MỤC LỤC

BÀI THỰC HÀNH SỐ 1

CYGWIN VÀ CÁC LỆNH LINUX CƠ BẢN

1. MỤC ĐÍCH

- Cài đặt và sử dụng cygwin.
- Sử dụng một số lệnh cơ bản của Linux

2. LÝ THUYẾT

2.1. Cygwin

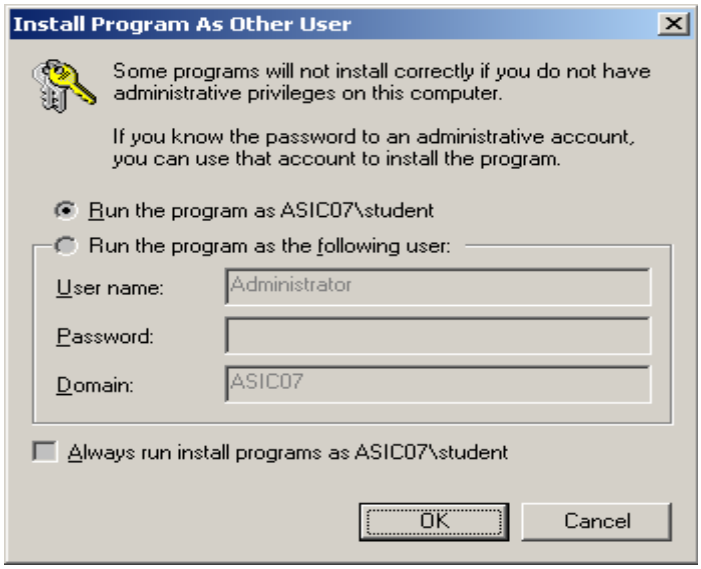
2.1.1. Khái niệm

Cygwin là một môi trường giống như Linux cho Window.Nó bao gồm hai phần:

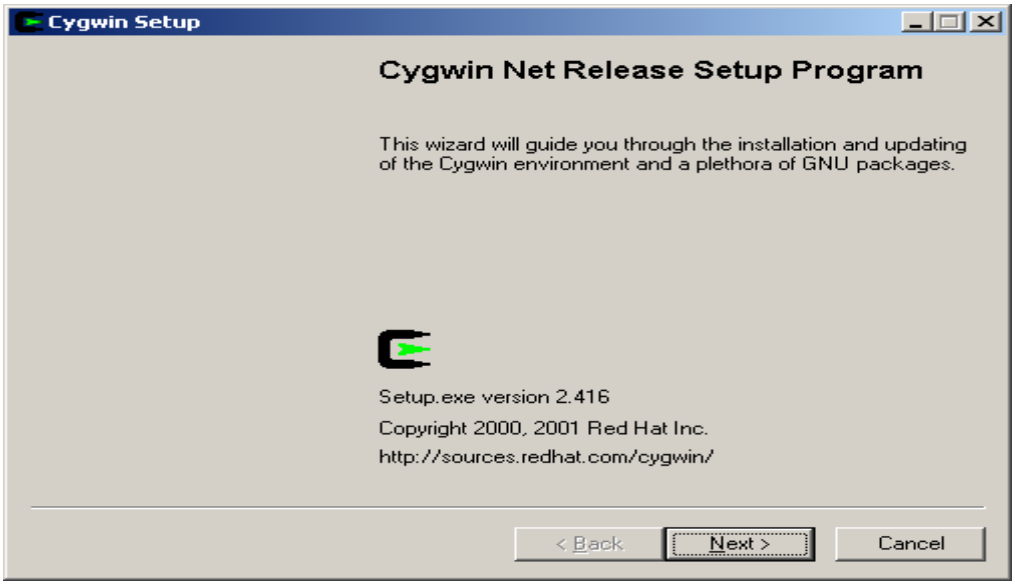
- Một DLL (cygwin1.dll) cung cấp các chức năng của Linux API.
- Một tập các công cụ,cung cấp cách nhìn và cảm giác như Linux.

2.1.2. Cài đặt

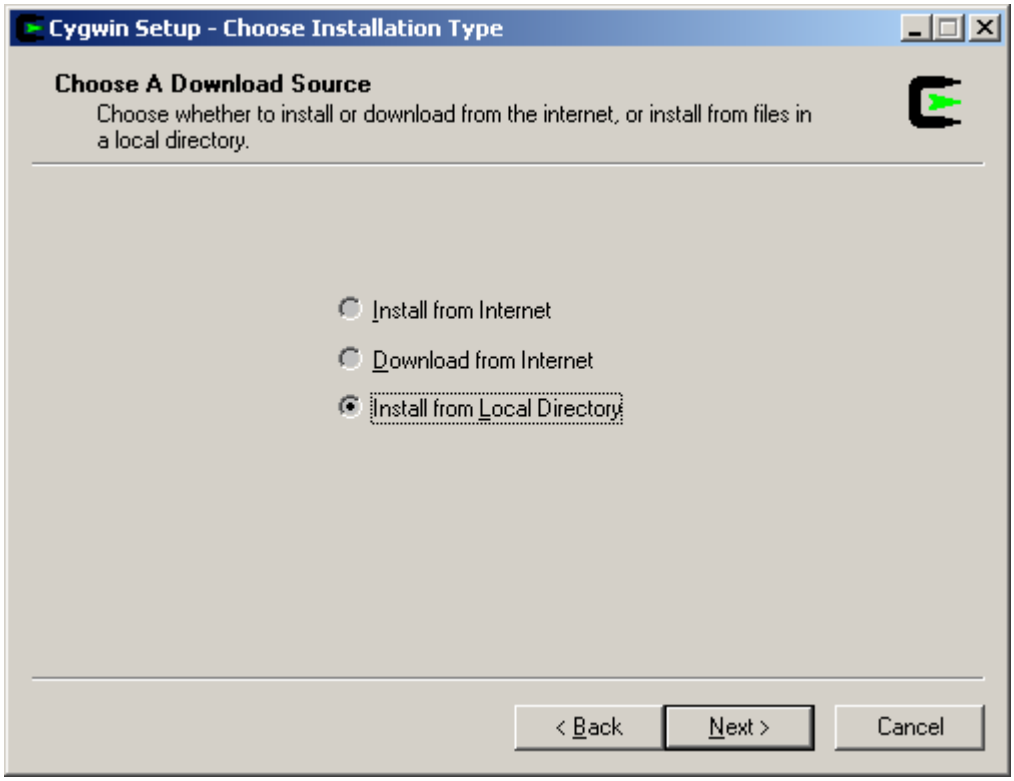
Chạy file setup.exe để cài đặt



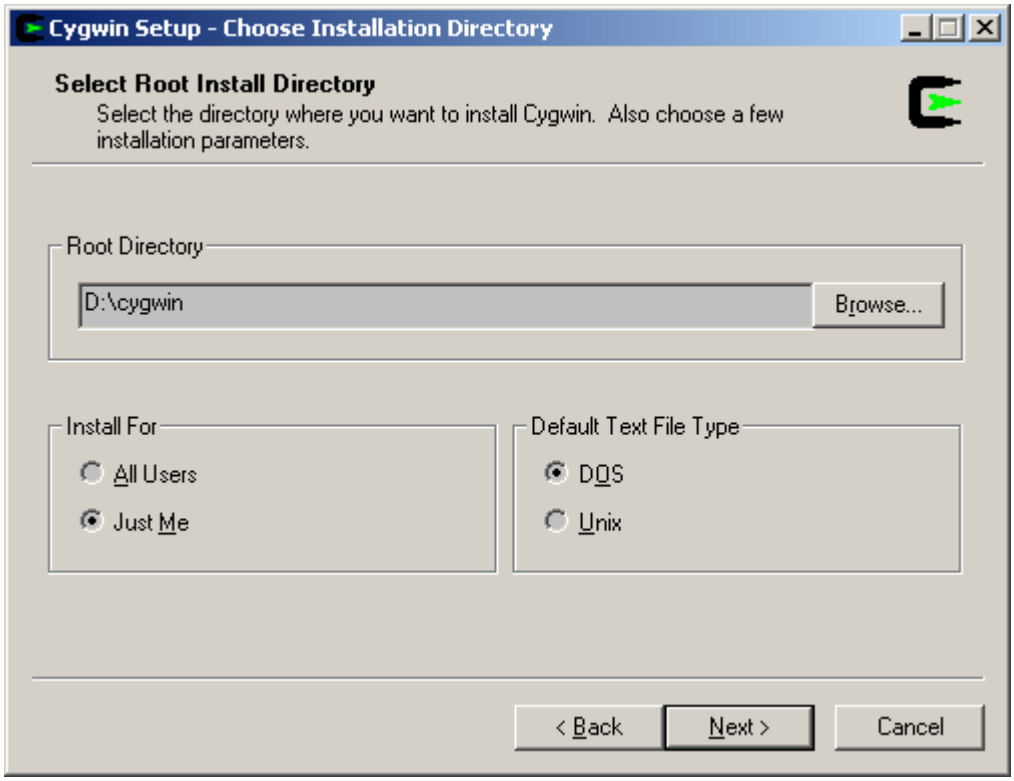
Nhấn OK để tiếp tục.



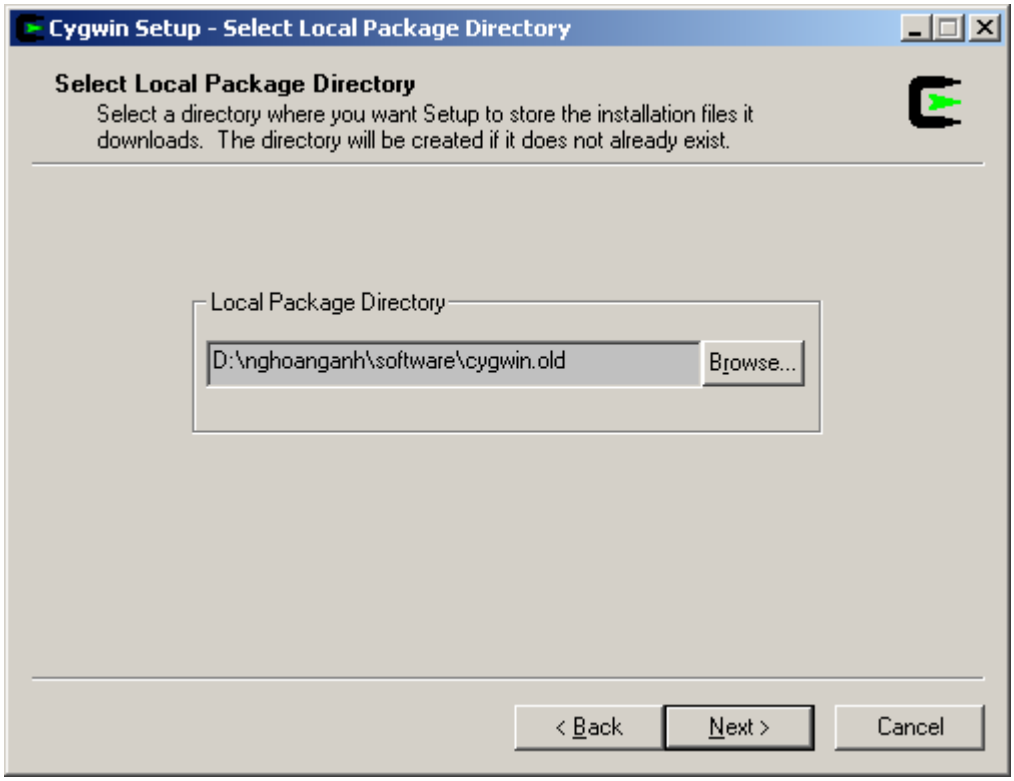
Nhấn Next để tiếp tục.



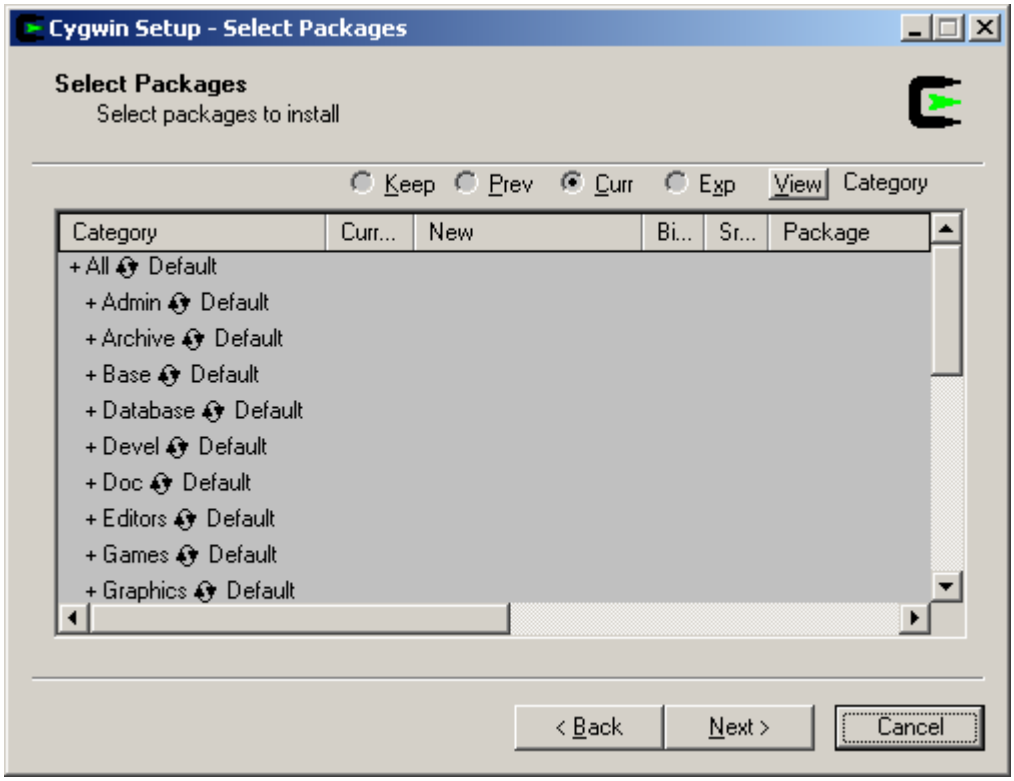
Chọn Install from Local Directory sau đó nhấn Next.



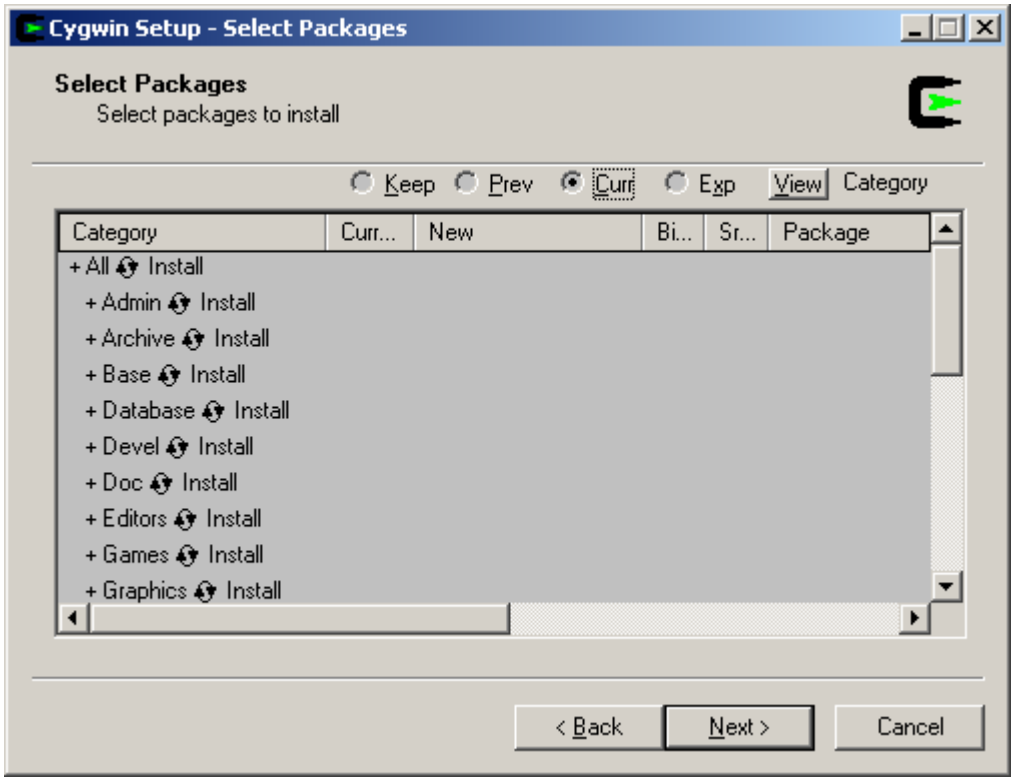
Chọn đường dẫn để cài đặt chương trình (ở đây là D:\cygwin) sau đó nhấn Next.



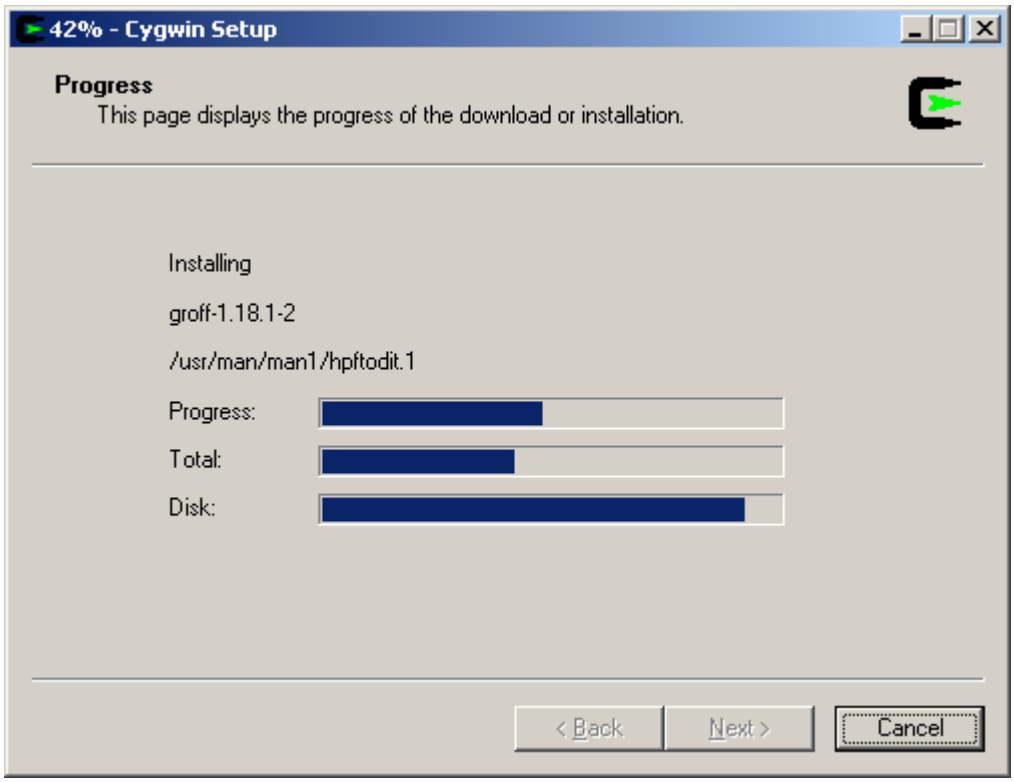
Bạn lựa chọn đường dẫn có chứa file setup.exe sau đó nhấn Next.

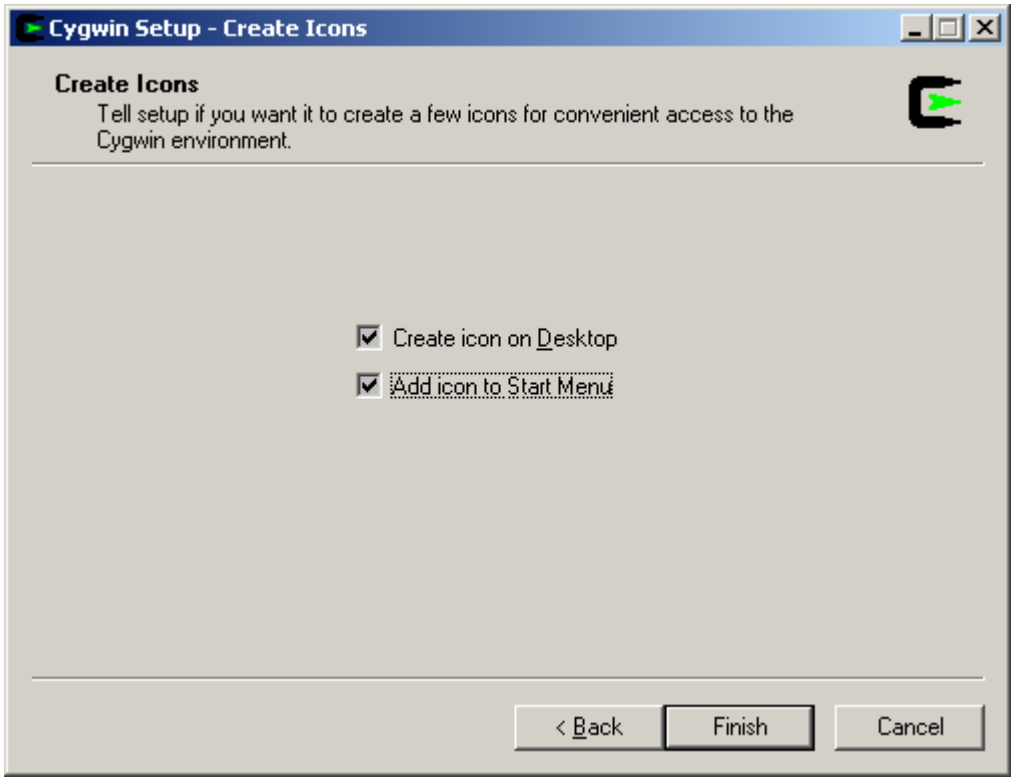


Nhấn chuột vào Default ở mục All chờ cho Default chuyển thành Install như hình sau



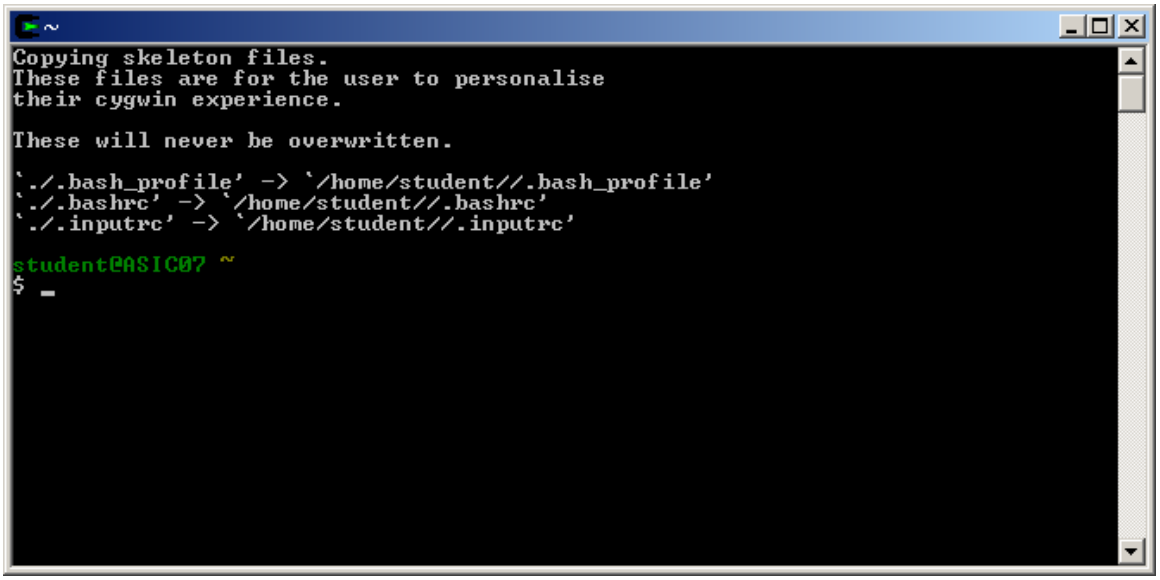
Nhấn Next để cài đặt.





Nhấn Finish để hoàn thành cài đặt.

Chạy thử Cygwin.



2.2. Tập lệnh cơ bản của Linux

- **Thay đổi thư mục**

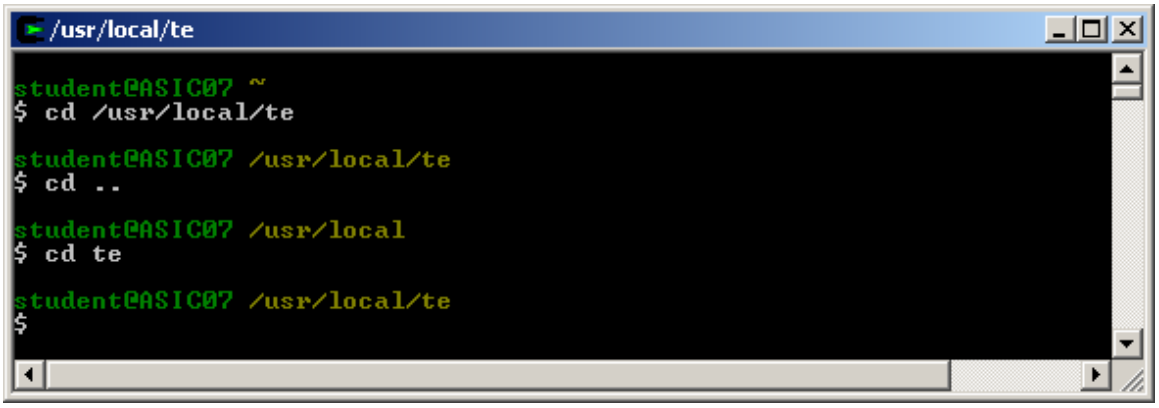
cd <pathname>

pathname cho biết thư mục cần chuyển tới. Đường dẫn có thể là tuyệt đối (tính từ thư mục gốc) hay tương đối (tính từ thư mục hiện hành).

Thư mục đặc biệt:

- Thư mục hiện hành : **.**
- Thư mục cha : **..**

Ví dụ:



- **Liệt kê nội dung thư mục**

ls –[option] directory_name

[option]:

-a liệt kê các file ẩn.

-d xem tên của thư mục hiện hành.

-F liệt kê các file và cho biết các kiểu của file qua ký hiệu ở cuối

Không có ký hiệu gì: file bình thường.

'/' : directories.

'*' : executable files.

"@" : linked file.

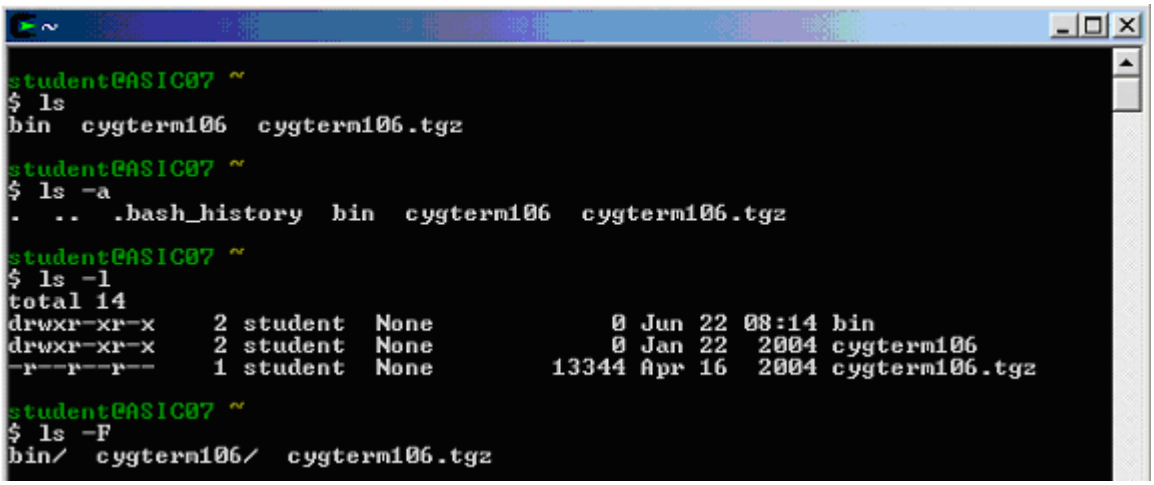
-i cho biết số inode của file.

-l liệt kê đầy đủ các thuộc tình của file.

-R liệt kê các thư mục con theo kiểu đệ quy.

-t sắp xếp theo thời gian cập nhật.

Ví dụ:



- **Tạo thư mục**

mkdir directory_name

Ví dụ:

- **Xóa file hay thư mục**
- **Xóa thư mục trống:**

rmdir directory_name(s)

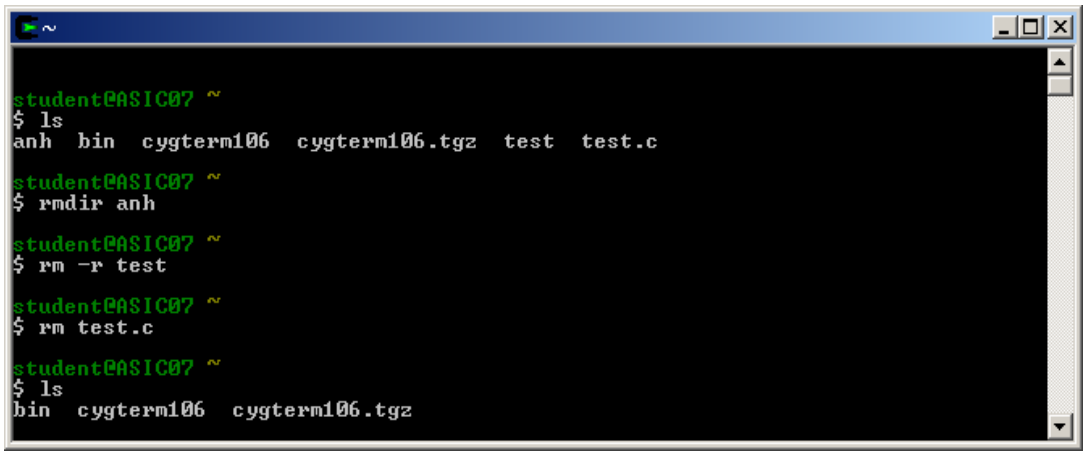
- **Xóa thư mục không trống**

rm –r directory_name(s)

- **Xóa file**

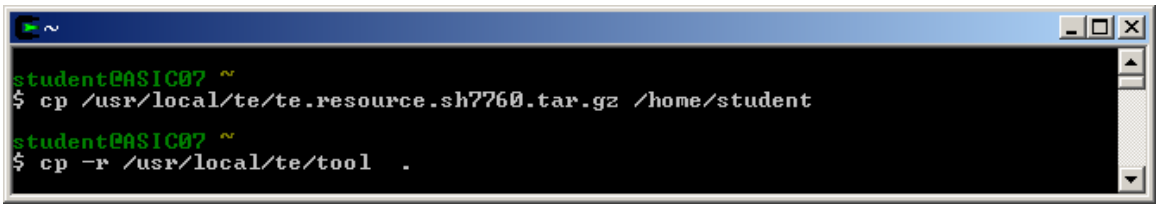
rm –option filename

Ví dụ:



- **Copy**
  - Copy files:
    cp [-option] source_file destination_file
    cp [-option] sorce_files destination_directory
  - Copy thư mục:
    cp –r source_directory(s) source_directory

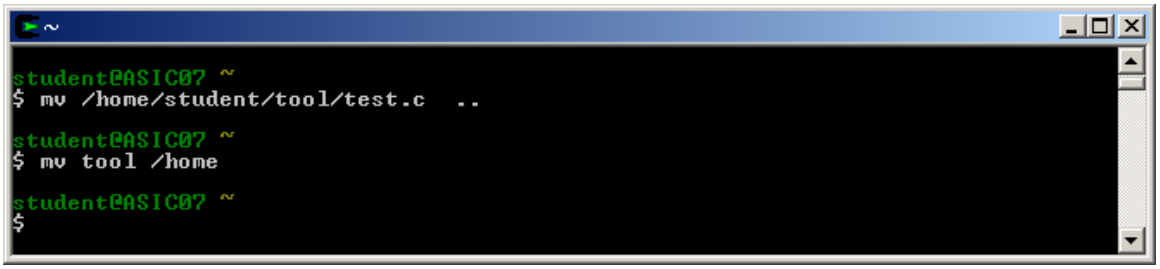
Ví dụ:



- **Di chuyển file hay thư mục**

mv [option] filename1 filename2
directory1 directory2
filename directory

Ví dụ:

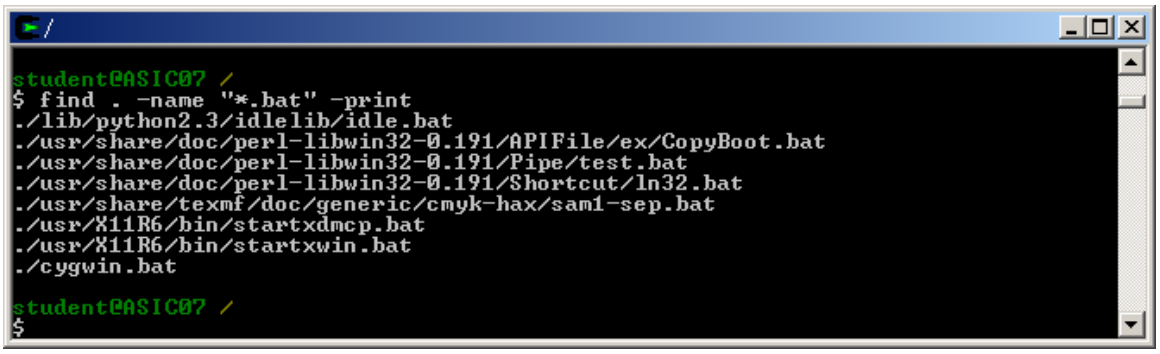


- **Tìm kiếm một file**

find pathname -name filename –print

(có thể dùng wildcard đặt trong dấu nháy kép)

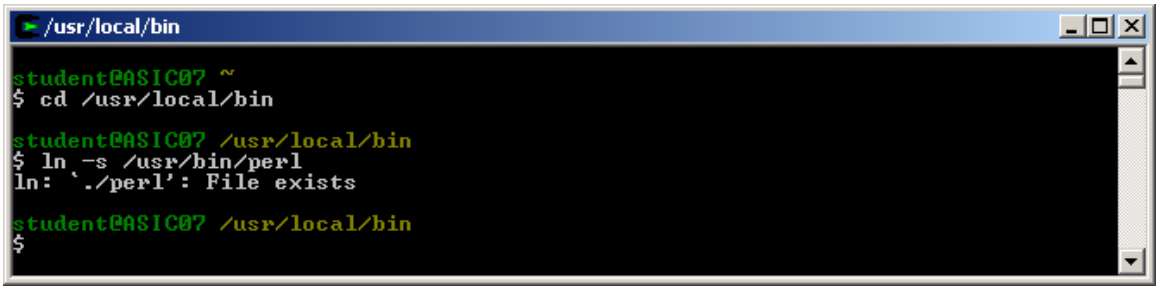
Ví dụ: Tìm những file .bat trong thư mục hiện hành



- **Tạo liên kết tắt mềm**

Tương tự như khái niệm shortcut của Windows UNIX cho phép tạo một file liên kết tắt đến một file hay thư mục khác.Có hai loại liên kết tắt cứng và mềm.Ở đây chỉ nói về liên kết tắt mềm: **ln -s** , chứa các thông tin trỏ đến file vật lý.

Ví dụ tạo một link đơn giản từ /usr/local/bin đến /usr/bin/perl



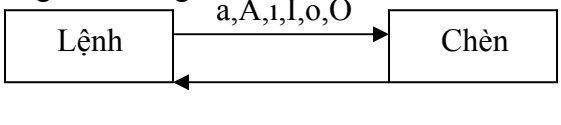
- **Xem và chỉnh sửa nội dung file bằng vi (visual interpreter)**

vi là trình soạn thảo thông dụng nhất trong UNIX.

Có hai chế độ hoạt động:

-Chế độ lệnh

-Chế độ chèn



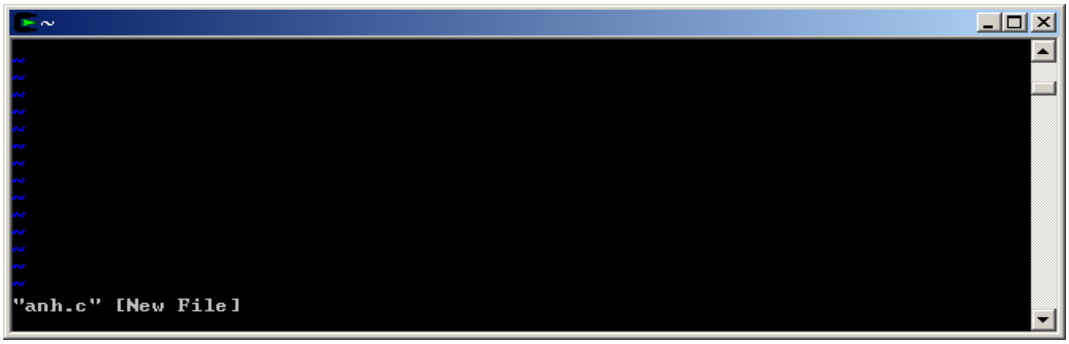
Chế độ lệnh:

Bắt đầu khi vào chương trình vi.Có thể thực hiện các thao tác di chuyển,thao tác trên các đối tượng text…

Muốn chuyển sang chế độ lệnh,nhấn Esc.

Chế độ chèn:

Cho phép nhập văn bản vào buffer.Có nhiều lệnh để vào chế độ chèn như:a,A,i,I,o,O.



- Các thao tác đơn giản:

Nhấn **a** (append) để vào chế độ chèn.
Nhập vào văn bản
Sau khi nhập xong,chuyển sang chế độ lệnh:Nhấn **Esc**
Lưu dữ liệu và thoát khỏi vi: Nhấn **:w** ,nhấn **Ente**r, nhấn **:q**

- Các lệnh di chuyển:

hlùi 1 ký tự
j xuống một dòng
k lên một dòng
l qua phải một ký tự
:n di chuyển đến dòng **n**

- Các lệnh chèn văn bản:

i: chèn trước cursor.
I: chèn ơ đầu dòng.
a: chèn sau cursor.
A: nối vào cuối dòng.
o: mở một dòng trống phía dưới.
O: mở một dòng trống phía trên.

- Tìm kiếm trên văn bản

Tìm kiếm về sau **:/pattern<Enter>**
Tìm về trước **:?pattern<Enter>**
Lặp lại lần tìm trước **n**
Lặp lại lần tìm trước chiều ngược lại **N**

- Các lệnh xóa

Xóa từ hiện hành **dw**
Xóa từ ở trước **db**
Xóa cả dòng **dd**
Xóa đến cuối dòng **d$**
Xóa đến đầu dòng **d0**
Xóa n dòng kế tiếp **p** hoặc **:u**

- Các thao tác khác

Ghi ra file **:w**
Ghi ra file có tên ‘filename’ **:w filename**

Thoát vi (khi nội dung file chưa thay đổi) **:q**
Thoát và không ghi **:q!**
Ghi và thoát **:x!**

3. THỰC HÀNH

Tạo cây thư mục /home/dir1/dir2.Soạn thảo chương trình helloworld.c trong usr/local, copy vào các thư mục dir1,dir2.Sau đó xóa dir1.

BÀI THỰC HÀNH SỐ 2

GIỚI THIỆU VỀ GCC VÀ GDB

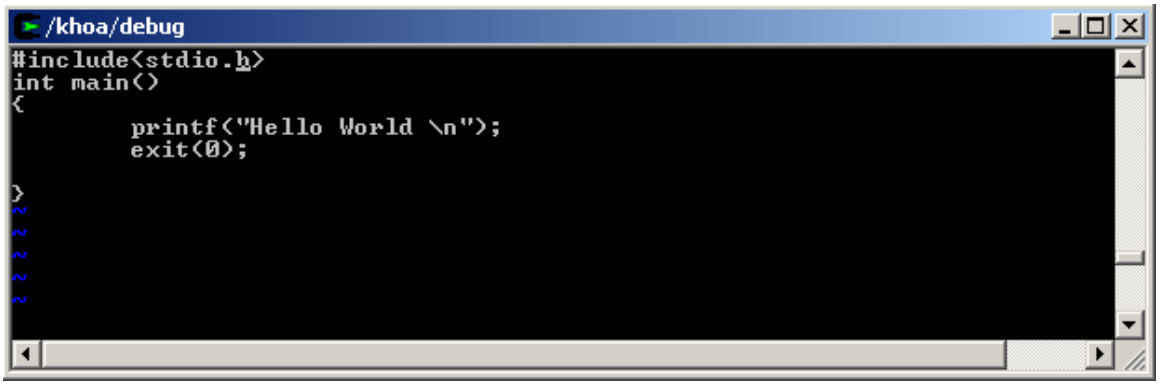
1. MỤC TIÊU

- Có khả năng viết , dịch và chạy một chương trình bằng C
- Biết cách debug trong môi trường Linux

2. LÍ THUYẾT

2.1 Làm quen với gcc

"Helloworld" là một chương trình kinh điển đối với hầu hết chúng ta khi bắt đầu tìm hiểu một ngôn ngữ lập trình mới. Không chỉ đơn giản in ra lời chào, chương trình cơ bản này còn giúp chúng ta hiểu được điều kiện và môi trường tối thiểu mà chương trình ứng dụng có thể hoạt động được. Helloworld.c dưới đây là một chương trình như vậy.



Để soạn thảo một mã nguồn trên thì bạn có thể dùng chương trình *vi* trên linux hoặc dùng một số trình soạn thảo khác. Nhưng do chúng ta dùng chương trình Cygwin nên gặp nhiều khó khăn cho việc sử dụng trình soạn thảo ***vi*** .Do đó để thuận lợi thì chúng ta sẽ sử dụng Notepad .

Trên UNIX để biên dịch một file C thông thường chúng ta sẽ sử dụng là ***cc*** hay ***gcc***. Việc sử dụng ***gcc*** và ***cc*** là hoàn toàn giống nhau , chúng không đòi hỏi cấu hình gì thêm cả nhưng ở đây chúng ta sẽ sử dụng ***gcc*** vì nó tương thích với các sản phẩm GNU cho họ Linux.

$ gcc Helloworld.c –o helloworld
$ ./helloworld
Hello World
$

Cách chương trình làm việc:

Trình biên dịch gcc và cc yêu cầu file chứa mã nguồn C trên thông số dòng lệnh để biên dịch. Ở đây ta chỉ định Helloworld.c ở đối số dòng lệnh thứ nhất. Tùy chọn –o yêu cầu trình biên dịch tạo ra file kết xuất (file chương trình thực thi) mang tên helloworld (bạn có thể chỉ định một tên kết xuất khác với tên file nguồn). Nếu bạn không chỉ định tùy chọn –o thì các trình biên dịch sẽ tạo ra file thực thi với tên là a.out . Khi đó thay vì gọi helloworld trên dòng lệnh bạn phải gọi a.out

Thường thì có 2 nguyên nhân chủ yếu khiến chương trình của bạn không chạy được sau khi đã biên dịch thành công. Thứ nhất nếu bạn gọi chương trình theo cách của DOS như:

$ helloworld

bash : helloworld : command not found

Lý do là Linux không tìm thấy đường dẫn đến nơi chứa file đã biên dịch helloworld . Linux không tìm file thực thi trong thư mục hiện hành như DOS. Để giải quyết vấn đề này bạn có thể chỉ định đường dẫn thư mục hiện hành **./** trước tên chương trình cần gọi.

Thứ hai ,chương trình không chạy khi gọi từ dòng lệnh là do file thực thi chưa được cấp quyền **x**.

$ ./helloworld

bash : ./ helloworld : Permission denied

Cho dù file thực thi của bạn đã biên dịch ra mã nhị phân nhưng Linux không quan tâm đến. Nó yêu cầu bất kì file nào muốn thực thi phải có quyền **x** trên thuộc tính phân quyền của file trước đã. Gặp trường hợp này bạn có thể gọi lệnh **chmod** để thêm quyền **x** cho helloworld như sau

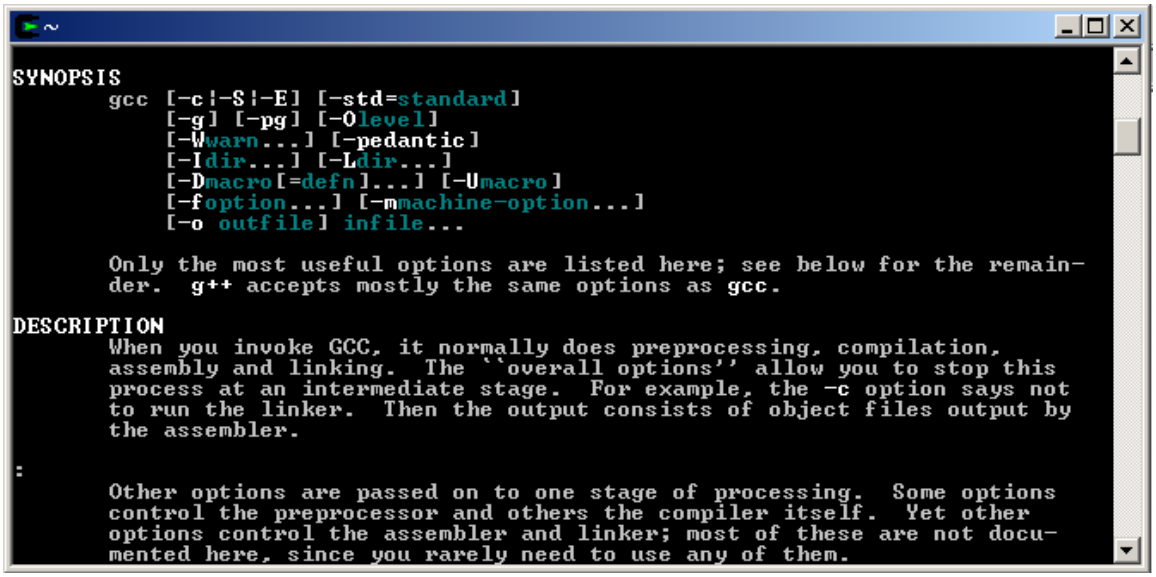
$ chmod o+x helloworld

Trong hệ thống nếu như người quản trị đã thiết lập cờ **umask** ,cho phép các file khi tạo ra mặc định có quyền **x** thì bạn không cần phải quan tâm tới vấn đề này.

Các tùy chọn của gcc:

Thật sư chúng ta dùng rất ít tùy chọn trên dòng lệnh biên dịch của trình biên dịch gcc (chủ yếu là –o, -g hoặc –l) . Tuy nhiên trình biên dịch gcc có rất nhiều tùy chọn. Bạn có thể tham khảo tài liệu **man** như sau:

$ man gcc



Bộ chương trình GNU cũng kèm theo trình **info** hướng dẫn sử dụng phần mềm rất chi tiết. Bạn có thể yêu cầu info hướng dẫn sử dụng bằng lệnh sau

$ info gcc

Sau đây sẽ là các tùy chọn của gcc:

-x *language* Đặc tả tường minh ngôn ngữ được sử dụng trong các file input (thay vì để trình biên dịch tự lựa chọn dựa vào phần mở rộng của các tên file). Đặc tả này có ý nghĩa với tất cả các file được đi theo sau nó cho đến khi gặp một –x kế tiếp nữa. Các ngôn ngữ mà trình biên dịch gcc có hỗ trợ là :

```
c  c-header  cpp-output
c++  c++-cpp-output
objective-c  objc-cpp-output
assembler  assembler-with-cpp
ada
f77  f77-cpp-input  ratfor
java
treelang
```

-x *none* Tắt bất kì đặc tả ngôn ngữ nào ở đây và trình biên dịch sẽ dựa vào phần mở rộng của các file input để dịch mà thôi

-c Compile hoặc assemble các file source nhưng không thực hiện quá trình link. Kết quả của quá trình biên dịch này sẽ tạo ra các file đối tượng. Mặc định thì các file đối tượng được tạo ra sẽ là các file input nhưng được thay thế phần mở rộng .c , .i ,.s,....
bằng .o

-o *file* Tạo kết quả biên dịch là một file thực thi mà có tên là *file*

-g Giống như tùy chọn –o nhưng file tạo ra cho phép ta có thể debug quá trình thực thi của nó.

-l*libraryname* Cho phép liên kết với các thư viện khi dịch

Ở đây có một lưu ý đó là thứ tự các tùy chọn khi biên dịch là không quan trọng.

2.2 Debug một chương trình bằng GDB

Tất cả các phần mềm đều chứa đựng lỗi. Thông thường thì 100 dòng lệnh là có khoảng 2-5 dòng lệnh bị lỗi (2-5%). Các lỗi thường gặp được phân loại và sử dụng một số phương pháp chung để loại bỏ chúng như sau:

- *Lỗi đặc tả* : Nếu như chương trình ngay từ đầu đã đặc tả sai thì chúng sẽ không hoạt động theo yêu cầu là điều tất yếu. Lập trình viên giỏi nhất thế giới cũng có thể viết ra một chương trình sai với ý tưởng của người dùng. Do đó mà trước khi bắt tay vào lập trình (hay thiết kế) bạn cần phải hiểu rõ mục đích mà chương trình mình phải thực hiện là gì.
- *Lỗi thiết kế* : Chương trình cho dù kích thước nhỏ đi chăng nữa thì cũng cần phải thiết kế .Thường thì ít có chương trình nào bắt đầu bằng việc bạn ngồi ngay vào máy gõ lệnh và thế là nó chạy tốt. Hãy dành thời gian để nghĩ xem chương trình của bạn cần được xây dựng theo cách nào, cấu trúc dữ liệu nào cần dùng đến và chúng sẽ được xử dụng ra sao? Nếu lỗi thiết kế xảy ra thì nó có thể làm cho bạn phải ngồi viết lại toàn bộ chương trình.
- *Lỗi viết mã* : Dĩ nhiên mọi người đều có thể gõ sai lệnh. Tuy nhiên, các trình biên dịch đủ sức bắt lỗi cú pháp và bạn không phải lo lắng về vấn đề này. Có một số trình thông dịch không bắt lỗi khi bạn soạn thảo, lỗi chỉ phát sinh khi chương trình đang chạy. Đây là vấn đề cần quan tâm. Lỗi viết mã khó bắt nhất là các lỗi thuộc về logiccủa chương trình. Ví dụ những vòng lặp vô tận không có điều kiện thoát, những lỗi đánh chỉ số mảng vượt quá phạm vi cho phép.

Quá trình gỡ lỗi một chương trình có thể được thực hiện bằng cách theo dõi quá trình hoạt động của chương trình bằng các công cụ tự động. Thường các trình debugger kèm theo trình biên dịch sẽ thực hiện công việc này. Chương trình của bạn vẫn được biên dịch ra mã nhị phân và chạy như một chương trình bình thường. Tuy nhiên, trình biên dịch sẽ thêm một số thông tin để trình bắt lỗi debugger có thể dựa vào đó hướng dẫn và thực thi chương trình theo yêu cầu tương tác của bạn. Một chương trình nhị phân muốn thêm vào các thông tin gỡ lỗi, bạn biên dịch với tùy chọn là **–g** (đối với trình biên dịch là gcc) . Chúng ta sẽ xử dụng trình gỡ lỗi **gdb** của GNU để thực thi và bắt lỗi các chương trình nhị phân được biên dịch với tùy chọn **–g**.

Để bắt đầu , chúng ta hãy thử với một chương trình gây ra lỗi sau. Đây là một đoạn chương trình chỉ phục vụ chức năng cho hàm **sort**.Việc sắp xếp dựa trên giải thuật Buble Sort. Mảng sắp xếp bao gồm các phần tử có cấu trúc **item**. Các phần tử được sắp xếp tăng dần theo khóa **key**.

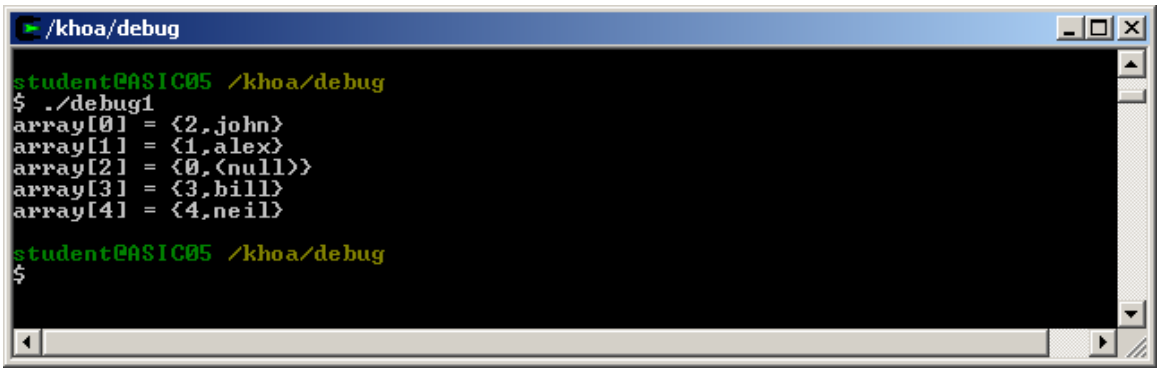
```
Chương trình debug1.c
/*******************************************/
typedef struct {
 char * data;
 int key;
} item;

item array[]={
 {"bill",3},
 {"neil",4},
 {"john",2},
 {"rick",5},
 {"alex",1},
};

void sort (item * a, int n)
{
 int i=0,j=0;
 for (;i<n ; i++)
 {
    for (j=0;j<n;j++)
    {
       if(a[j].key > a[j+1].key)
       {
          item t = a[j];
          a[j] = a[j+1];
          a[j+1] = t;
       }
    }
    n--;
 }
}
main()
{
 int i;
 sort(array,5);
 for (i=0; i<5;i++)
            printf("array[%d]= {%d,%s}\n" ,i,array[i].key, array[i].data);
      exit(0);
}
/*****************************************/
```

Chúng ta hãy thử biên dịch chương trình :

$ gcc debug1.c –o debug

Trình biên dịch thực hiện thành công và không đưa ra thông báo lỗi nào cả. Lỗi này thuộc về lỗi logic chỉ phát sinh lúc chương trình thực thi.

Khi chạy chương trình thì ta sẽ được kết quả sau :

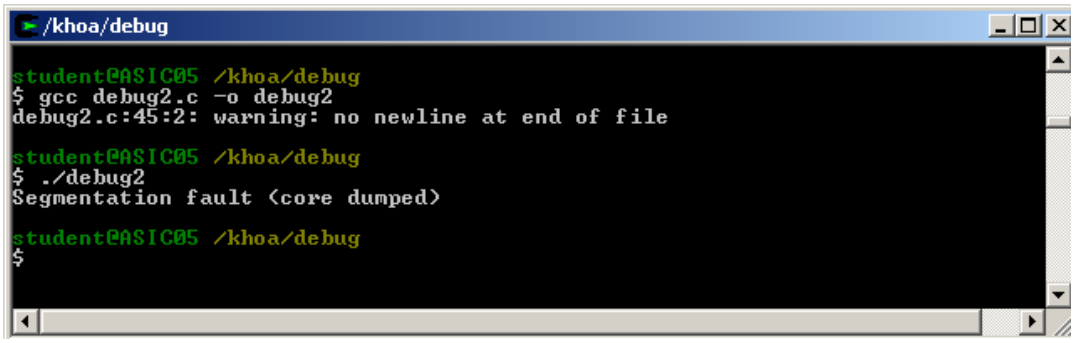


Chúng ta thấy rằng khi chạy chương trình này thì kết quả kết xuất không theo thứ tự mà chúng ta muốn sắp xếp với lại tại vị trí thứ 2 của mảng đã bị sai. Kết quả này là do quá trình chúng ta truy xuất vào các phần tử không có trên mảng. Tùy theo các phiên bản khác nhau của Linux và Unix thì có thể cho ra một thông báo lỗi khác nhau.

Do hệ điều hành luôn cấp phát một vùng nhớ dư cho một ứng dụng nào đó nên ở đây tại vị trí của phần tử thứ 2 thì có kết quả **(null)**. Nếu như bây giờ ta sửa lại đoạn chương trình nguồn thì nó sẽ kết xuất ra một thông báo lỗi khác

```
typedef struct {
 char  data[10000000];
 int key;
} item;
```

Thông báo lỗi bây giờ là

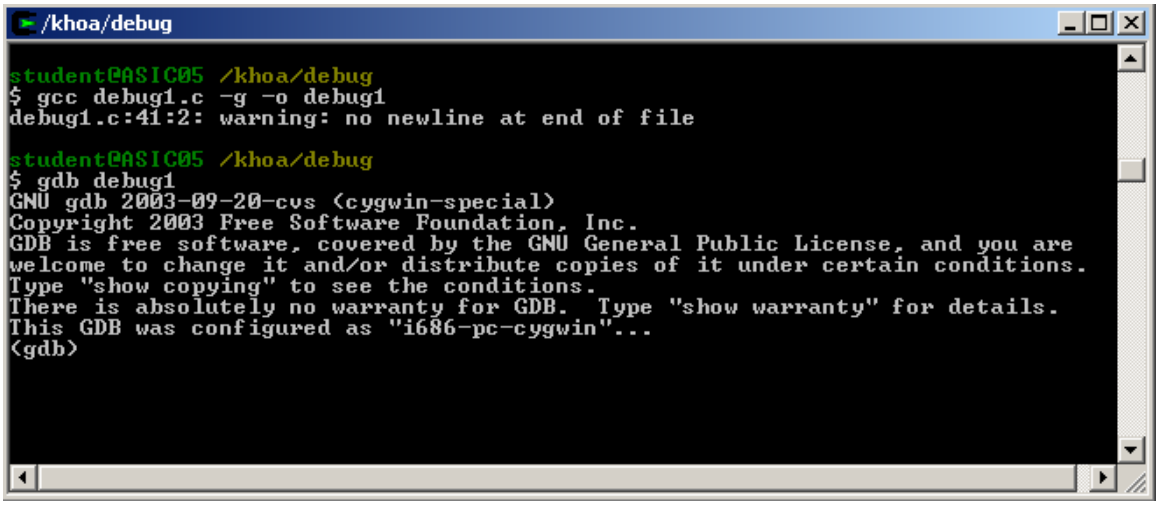


Lí do của việc này là khi kích thước của một phần tử quá lớn thì nó sẽ chiếm nhiều vùng nhớ , khi đó ta truy xuất vượt quá kích thước của mảng sẽ đồng thời vượt luôn vùng nhớ dư mà hệ điều hành cấp cho ứng dụng.

Để tìm lỗi cho chương trình, chúng ta bắt đầu quá trình biên dịch cho chương trình này lại với chế độ debug

$ gcc debug1.c –g –o debug1

$ gdb debug1



Bạn tương tác với **gdb** từ dấu nhắc của chương trình. Sau đây là các lệnh cơ bản trong **gdb**

run (r)	Thực thi chương trình đang debug cho đến khi gặp một breakpoint
break (b) num	Thiết lập breakpoint tại hàng thứ *num*
delete break num	Xóa breakpoint thứ num
disable break num	Tạm thời vô hiệu hóa điểm dừng này.
enable break num	Sử dụng lại breakpoint đã bị vô hiệu hóa.
help breakpoint	Xem các lệnh chi tiết về breakpoint
info breakpoint	Xem thông tin về các breakpoint hiện có trong chương trình
backtrace	Dò vết stack , chúng ta có thể dùng lệnh này để xem lại các bước mà chương trình đi đến lỗi bằng cách lần theo vết stack nơi lưu chứa các lời gọi hàm và trị trả về của hàm.
print j	In nội dung của một biến hay của một biểu thức nào đó. Để in giá trị của một mảng thì ta cũng làm tương tự *print* a[0]@5 . Với @5 chỉ ra số lượng phần tử kế tiếp cần in ra.
Display	In giá trị của một mảng bất kì khi chương trình thực thi tới breakpoint
list linenum	In *linenum* dòng code tiếp theo từ vị trí hiện tại, nếu không chỉ rõ số dòng thì gdb sẽ in ra một số lượng dòng nhất định nào đó.
cont (c)	Tiếp tục thực thi chương trình cho đến khi gặp một breakpoint mới
next (n)	Thực thi dòng lệnh kế tiếp
commands	Thực thi một số lệnh được chỉ định khi gặp một breakpoint
quit	Thoát khỏi **gdb**

3. THỰC HÀNH

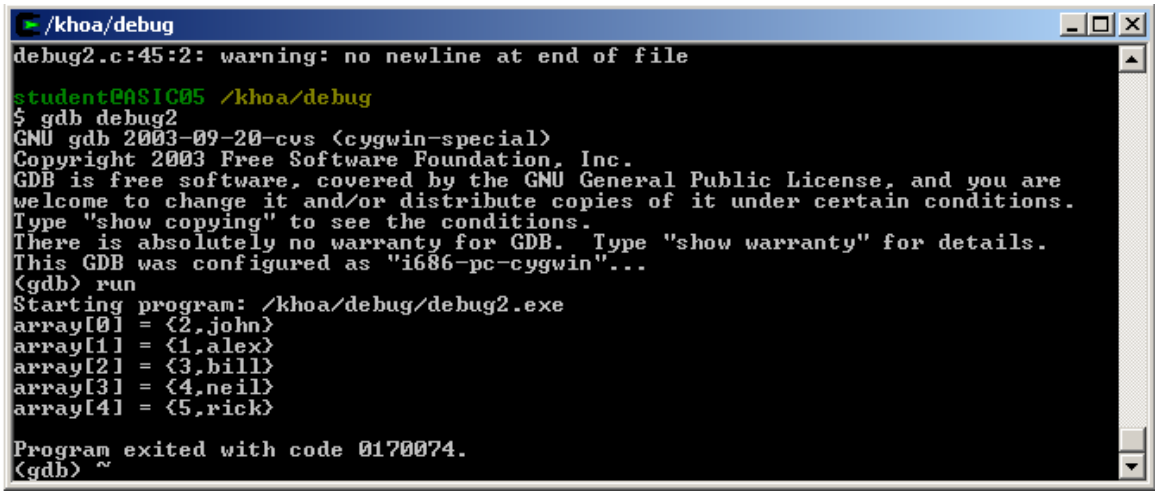
Hiện thực chương trình debug1.c bên trên . Dịch , chạy và dùng **dbg** để tìm lỗi cho chương trình đó.

- Dùng dbg để chạy chương trình debug1

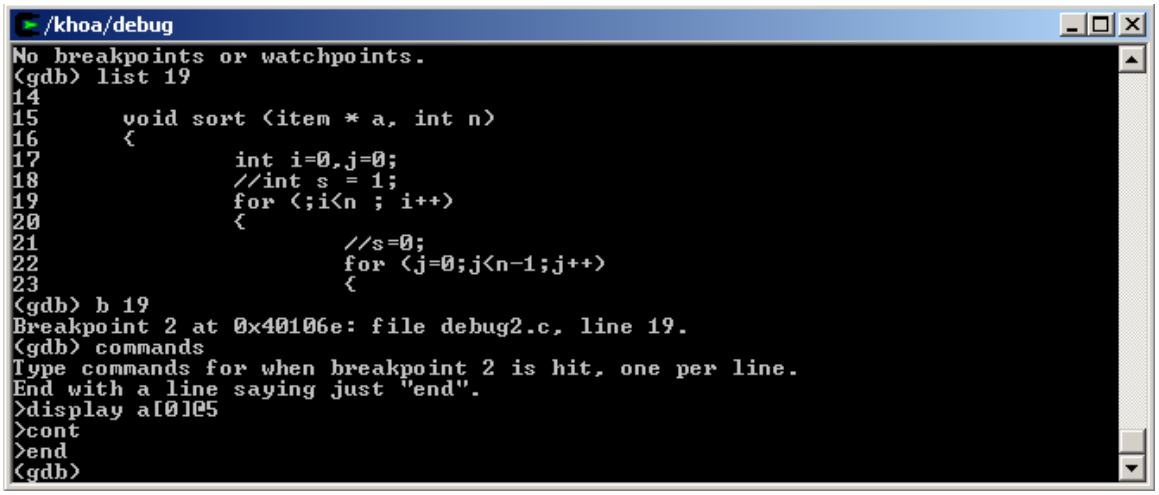
- Dùng lệnh *run* cho chương trình tự thực thi
- Dùng lệnh *backtrace* để thấy rằng lỗi xảy ra bên trong hàm sort
- Thiết lập breakpoint tại vị trí dòng lệnh

  a[j].key > a[j+1].key

sẽ thấy rằng ở đây a[j+1] là vượt quá trị của mảng , do đó mà vòng lặp phải sửa lại là $i < n-1$. Khi sửa lại thì dịch và chạy chương trình. Tuy chương trình không còn báo lỗi nữa nhưng kết quả xuất ra vẫn còn sai



- Để sửa lỗi tiếp tục ta sẽ đặt một breakpoint tại vòng lặp ngoài cùng và mỗi lần chương trình thực thi tới đó sẽ kiểm tra giá trị của mảng array.



Sau khi thiết lập như trên thì ta cho chương trình thực thi lại từ đầu , ta nhận thấy là số vòng lặp của chương trình ít hơn ta mong đợi. Vậy nguyên nhân nằm ở đâu???

Nguyên nhân chính là do ta đã giảm n-- đi. Có lẽ người lập trình nghĩ rằng cứ sau mỗi vòng lặp bên trong thì chương trình đã sắp xếp được một số rồi nhưng theo lẽ thì chỉ có thể giảm được số vòng lặp bên trong thôi chứ không thể giảm số vòng lặp bên ngoài Khi bỏ đi dòng lệnh này thì chương trình sẽ hoạt động một cách chính xác.

BÀI THỰC HÀNH SỐ 3

LÀM QUEN VỚI T-ENGINE

1. MỤC TIÊU

- Giới thiệu bộ Kit T-Engine và tổng quan về kiến trúc của nó.
- Cài đặt công cụ để phát triển và giải thích các thư mục được cài đặt.
- Giới thiệu và cài đặt các phần mềm giao tiếp giữa máy tính và T-Engine.
- Xây dựng và liên kết process.
- Khởi động T-Engine, truyền nhận chương trình và thực thi chương trình trên T-Engine.

2. LÝ THUYẾT

2.1 Tổng quan về T-Engine

2.1.1 T-Egine là gì

T-Engine là một platform được phát triển mở và chuẩn hoá cho những hệ thống nhúng. Nó đóng một vai trò quan trọng trong việc xây dựng những hệ thống tính toán thường gặp,mà thông thường tất cả đều được nhúng với máy tính thông minh và liên kết mạng.

T-Engine là sự kết hợp của chuẩn phần cứng (T-Engine board) và hệ điều hành thời gian thực(T-Kernel).T-Engine hổ trợ sự phân phối middleware.

T-Engine là kiến trúc tốt cho phát triển hệ thống nhúng nhanh chóng và hiệu quả cho các sản phẩm như cell phones và những ứng dụng thông tin.T-Engine hổ trợ eTRON, một kiến trúc bảo mật mạng phát triển dựa trên TRON Project, để đảm bảo những thông tin số sẽ đến an toàn mà không lo bị nghe trộm hay xáo trộn trong khi truyền.

2.1.2 Các sản phẩm T-Engine

T-Engine cung cấp bốn loại sản phẩm bao phủ phạm vi các thành phần tạo nên một môi trường tính toán thường gặp.

Hình 3.1: Dòng T-Engine

- **T-Engine chuẩn**
  Một sản phẩm có độ tương thích cao với người sử dụng,nhắm vào các thiết bị thông tin di động như cellular phone,PDA,DTV,..
- **µT-Engine (micro T-Engine)**
  Một sản phẩm giao tiếp đơn giản với người sử dụng chủ yếu dùng cho những thiết bị điều khiển,như những thiết bị điện gia dụng,thiết bị khoa học nghiên cứu ứng dụng.
- **nT-Engine (nano T-Engine)**
  Một platform dùng cho những thiết bị điện gia dụng nhỏ,các cảm biến kích thước cở một đồng xu.
- **pT-Engine (pico T-Engine)**
  Paltform cho những thiết bị là thành phần nhỏ nhất trong môi trường tính toán thông thuờng như đèn,công tắc,các cảm biến,van…

Standard T-Engine: Outline of CPU Board Specification

- 32-bit CPU with MMU
- 16MB/32MB RAM (increase possible with expansion board)
- 4MB Flash memory (increase possible with expansion board)
- Serial I/O - 384Kbps or more possible
- PCMCIA - Type II, 1 slot
- USB Host - Type A connector, 1 channel
- eTRON chip I/F SIM card connector)
- LCD panel I/F
- Sound CODEC - Input 1 channel, output 2 channels
- Extension bus I/F
- Calendar clock

µT-Engine: Outline of CPU Board Specification

- 32-bit CPU
- 2MB/4MB RAM (increase possible with expansion board)
- 4MB Flash memory (increase possible with expansion board)
- Serial I/O - 384Kbps or more possible
- CF card, Type II, 1 slot
- MMC card, 1 slot
- eTRON chip I/F (SIM card connector)
- Expansion bus I/F
- Calendar clock

Bảng 1

2.1.3 T-ENGINE/SH7760

Cấu hình hệ thống

Đặc điểm

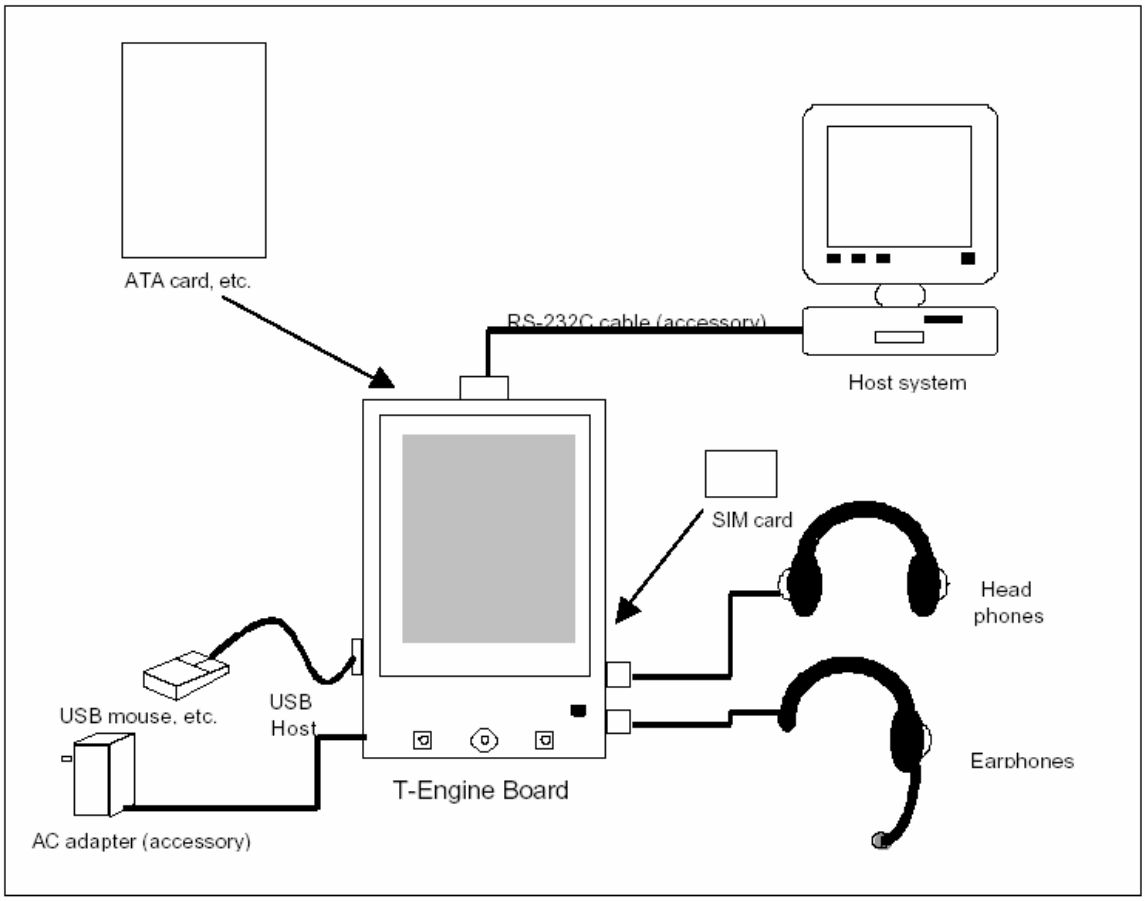
Chip ngoại vi LSI (PCMCIA controller và sound generator chip) sẵn có trên phương diện thương mại.

T-Engine bao gồm PCMCIA controller,sound generator chip,SIM card connector ,..tạo điều kiện thuận lợi để phát triển những ứng dụng của hệ thống.

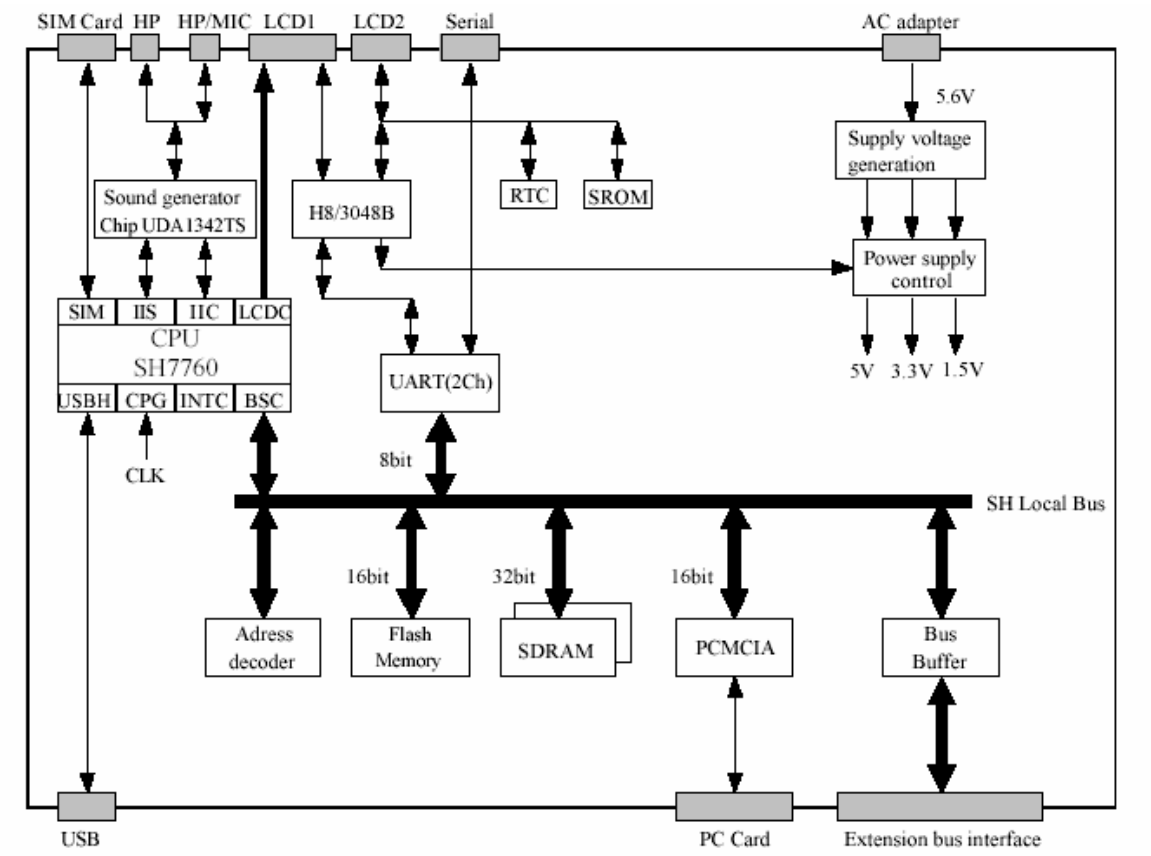
Board T-Engine có hai SH7760 bus (bus dữ liệu và bus địa chỉ) và khe cắm mở rộng điều khiển tín hiệu output , người sử dụng có thể kết nối các thiết bị đặc biệt.

Cấu hình T-Engine

Hình 1 cho thấy cấu hình hệ thống một T-Engine board và hình 2 là sơ đồ khối của T-Engine



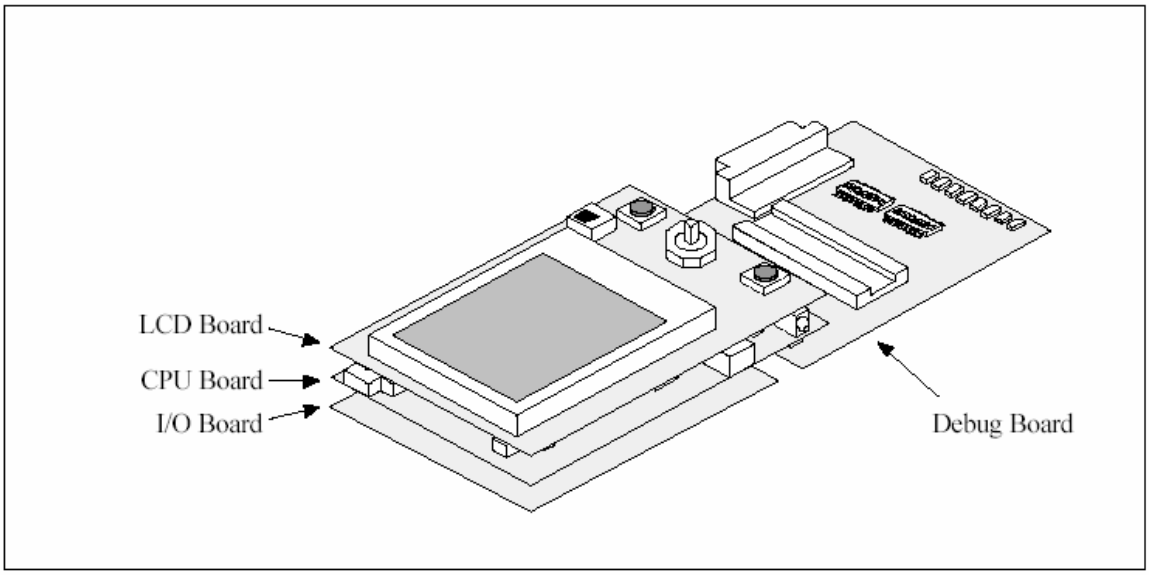
Hình 3.2: Cấu hình hệ thống.



Hình 3.3 : Sơ đồ khối

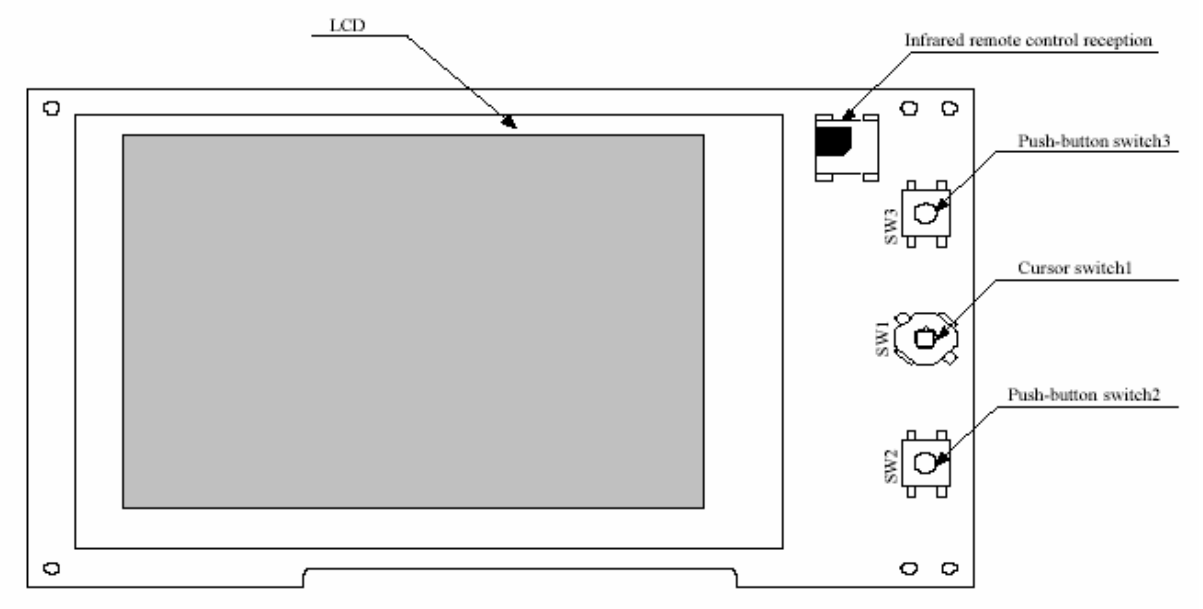
Hình dạng

T-Engine board bao gồm bốn board:CPU,LCD,debug,và I/O board.Hình 3 cho ta thấy cái nhìn bên ngoài của T-Engine.

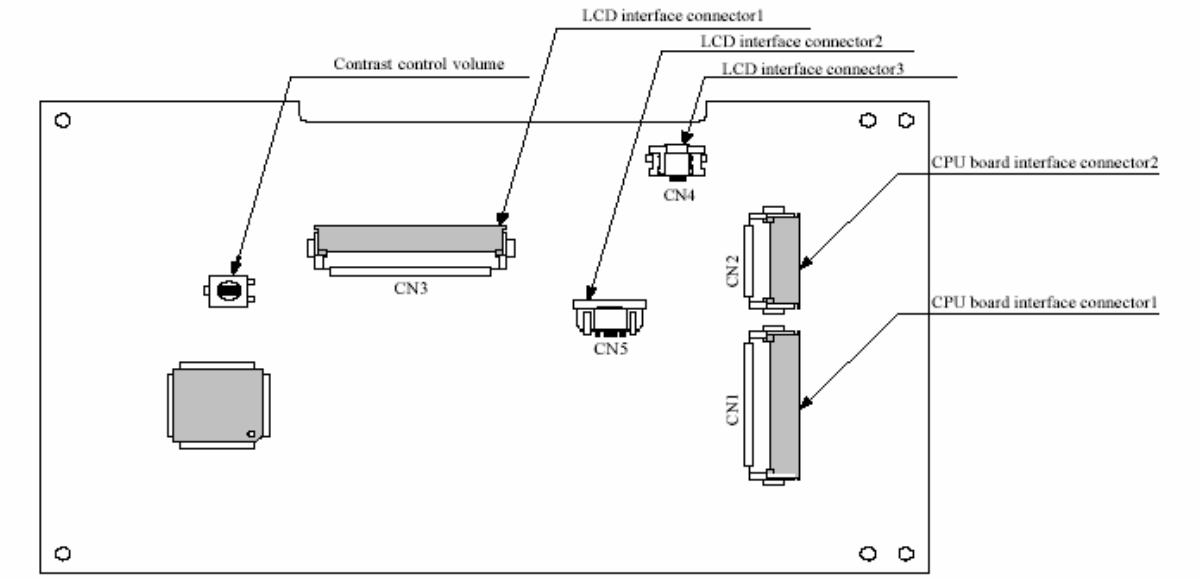


Hình 3.4**:** Hình dạng SH7760 T-Engine.

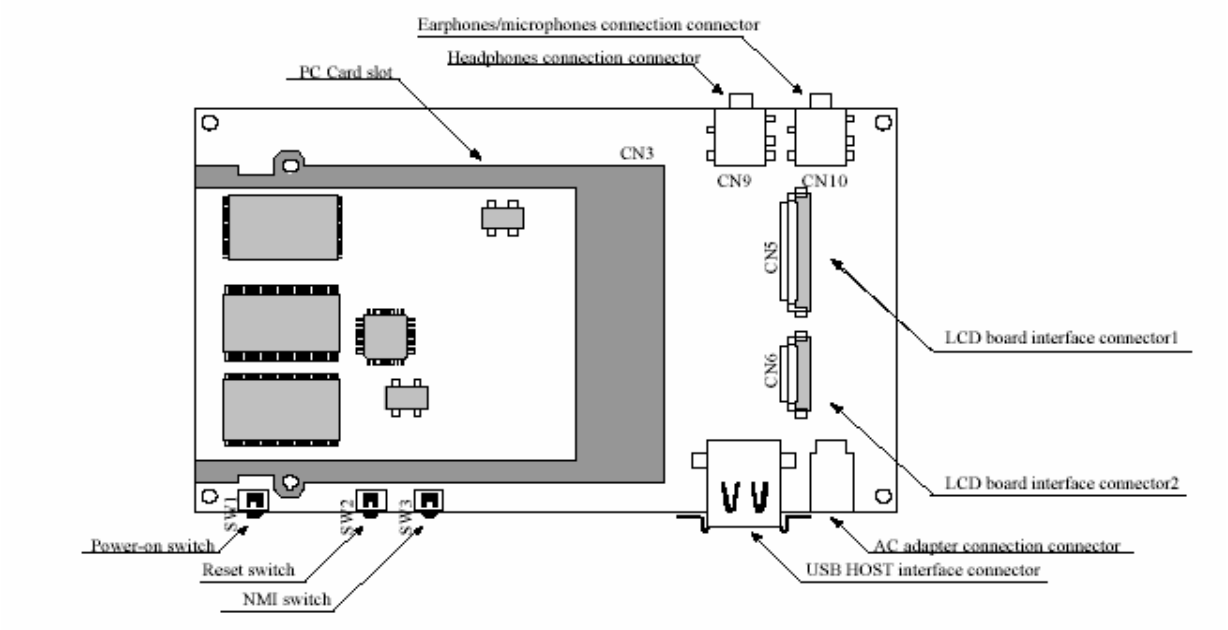
Các hình từ hình 4 đến hình 9 cho ta thấy hình ảnh của các board tương ứng (LCD,CPU , debug,và I/O board).



Hình 3.5**:** Mặt trước LCD board.

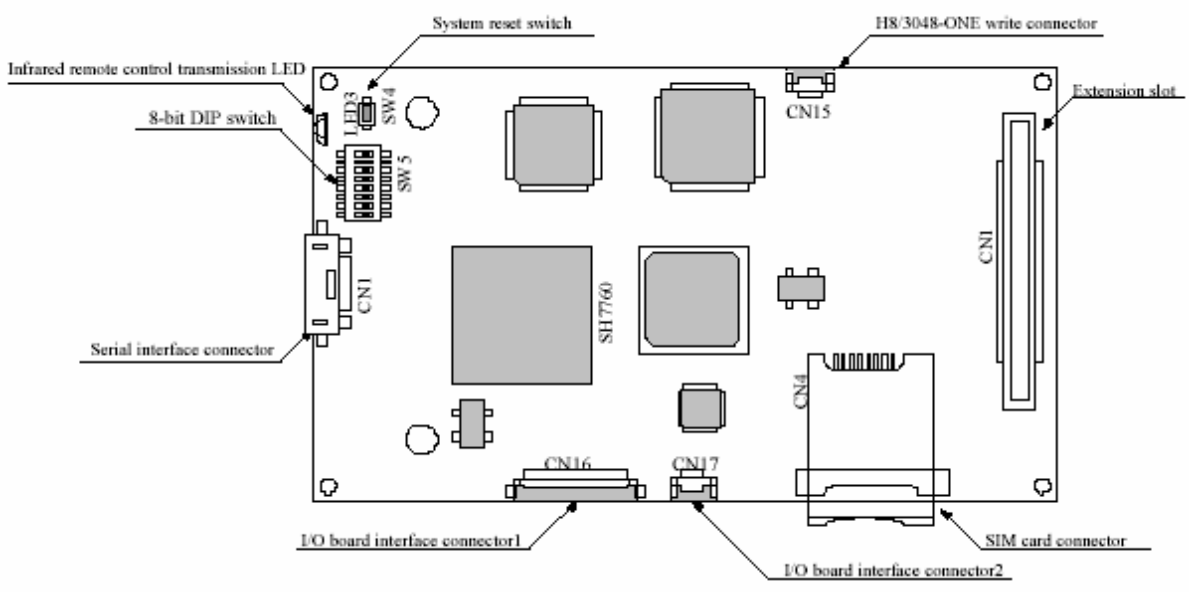


Hình 3.6**:** Mặt sau LCD board.



Lưu ý : kết nối CN15 được sử dụng cho kiểm tra mạch ưu tiên trong xưởng.Không sử dụng

Hình 3.7 **:** Mặt trước CPU board.



Hình 3.8: Mặt sau CPU board.

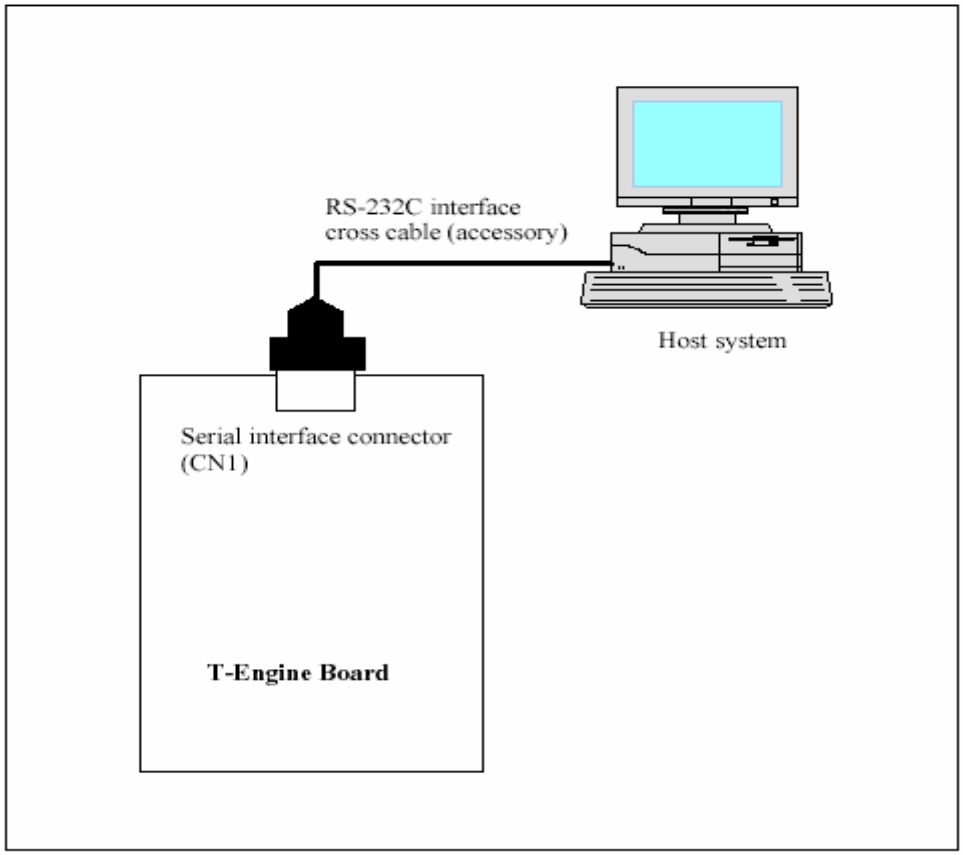
Item	Specifications	Target device
CPU	SH7760 Model name: HD6417760BP200D (RENESAS Technology) Input clock: 16.6667MHz Operating clock (Internal): 200MHz (x 12) (External): 66MHz (x 4) Package: 256-pin BGA	
Flash memory	Capacity: 8MB MBM29DL640E90TN (Fujitsu) x 1	
SDRAM	Capacity: 64MB EDS2516APTA-75 (ELPIDA) x 2	
PC Card I/F	One slot Controller: MR-SHPC-01 V2T (Marubun) Package: 144pin TQFP	
Serial I/F	2ch Controller: ST16C2550CQ48 (EXAR) Package: 48pin TQFP	ChA: H8/3048F-ONE I/F ChB: Monitor for debugging
Sound	Model name: UDA1342TS (Philips) Package: 28pin SSOP Earphone/microphone: 1ch Headphone output: 1ch - Microphone input Impedance: 2.2KΩ Sensitivity: -51dB/Pa - Headphone output Impedance: 32Ω	The SH7760 on-chip SSI is used to transmit data. The SH7760 on-chip IIC is used to select the mode.
USB Host	1ch Controller: SH7760 on-chip USB Host	
TFT color LCD module	NL2432DR22-02B (NEC) Display color: 262,144 colors Display area: 240(H) x 320(V) Controller:SH7760 on-chip LCDC	
Power supply controller	H8/3048F-ONE Model name: HD64F3048BVTE25 (Renesas Technology) Operating frequency: 7.3728MHz Package: 100-pin TQFP	The control SH7760 working for power supply control, RTC, or tablet interface infrared remote control must be interfaced via the serial chA.
RTC	Model name: RV5C348B (RICOH) Package: 10pin SSOP-G	Via the H8/3048F-ONE
Touch panel I/F	Model name: ADS7843 (TI) Package: 16pin SSOP	Via the H8/3048F-ONE (To be mounted on the LCD board)
Serial EEPROM	Capacity: 512 bytes Model name: S-29391AFJA (SII)	Via the H8/3048F-ONE
Infrared remote control	Transmission Model name: GL100MN0MP (SHARP) Transmission carrier: 38KHz Reception Model name: GP1UC101 (SHARP) Transmission carrier: 38KHz	Via theH8/3048F-ONE

Bảng 2**:** Đặc điểm chức năng của T-Engine.

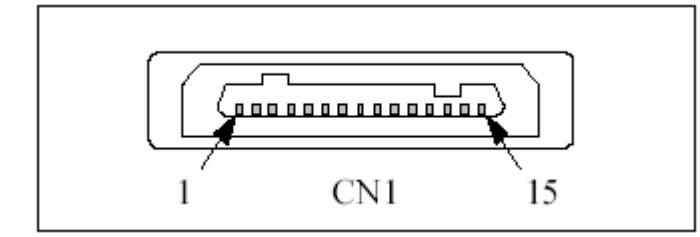
Cài đặt

Kết nối hệ thống

Để sử dụng T-minitor,ta sử dụng cáp chuẩn RS-232C nối với connector CN1 của T-Engine với máy tính.Hình 2.1 cho thấy cách thức kết nối với máy chủ và hình 2.2 cho thấy các chân của connector CN1



Hình 3.9 **:** Kết nối hệ thống**.**



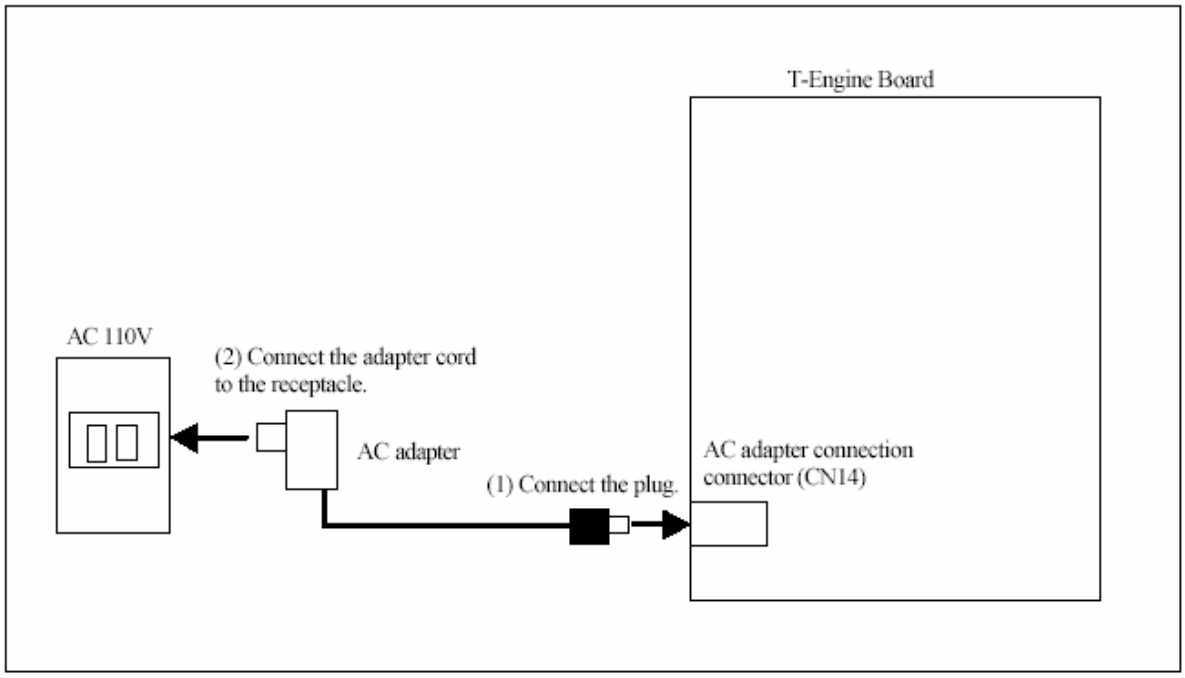
Hình Chân 3.10 **:**connector nối tiếp CN1.

Pin No.	Signal name	I/O	Remarks
1	GND	-	
2	TxD	Output	TXB(UART)
3	RxD	I	RXB(UART)
4	GND	-	
5	RTS	O	RTSB(UART)
6	CTS	I	CTSB(UART)
7	GND	-	
8	Reserved	-	ISP TCK(*)
9	Reserved	-	GND(*)
10	Reserved	-	ISP TMS(*)
11	Reserved	-	ISP Plug(*)
12	Reserved	-	ISP BScan(*)
13	Reserved	-	ISP TDI(*)
14	Reserved	-	ISP TDO(*)
15	Reserved	-	Vcc(3.3V) (*)

Bảng 3:Các tín hiệu của các chân connector CN1.

Lưu ý: (*) :Những chân này chỉ sử dụng để kiểm tra board trong xưởng,không sử dụng chúng với mục đích khác.

Kết nối Adapter



Hình 3.11 **:** Kết nối adapter.

Lưu ý:

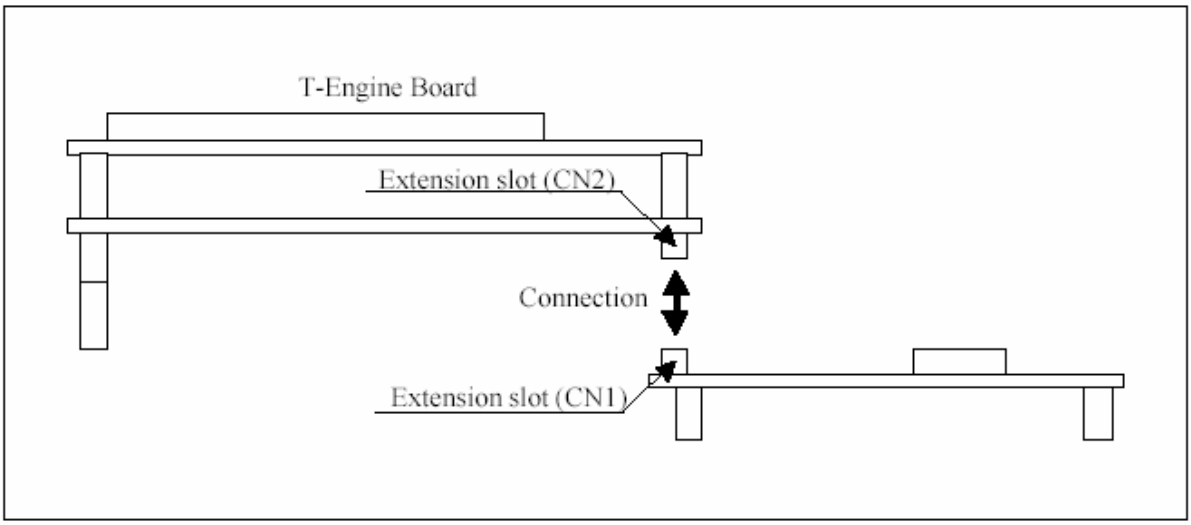
- Không để các vật nặng lên dây của AC adapter. Để tránh những rủi ro về rỉ điện,lửa,hay sốc điện,không gây hại dây của AC adapter.
- Để tránh những nguy hại về sốc điện,không rút dây cắm khi tay ướt,không kéo dây..

Mở ,tắt T-Engine board

Để mở hay tắt T-Engine nhấn công tắc SW1 trên CPU board. Để mở T-Engine nhấn và giữ công tắc trong 0.5 giây hay hơn. Để tắt nhấn và giữ công tắc ít nhất 2 giây.

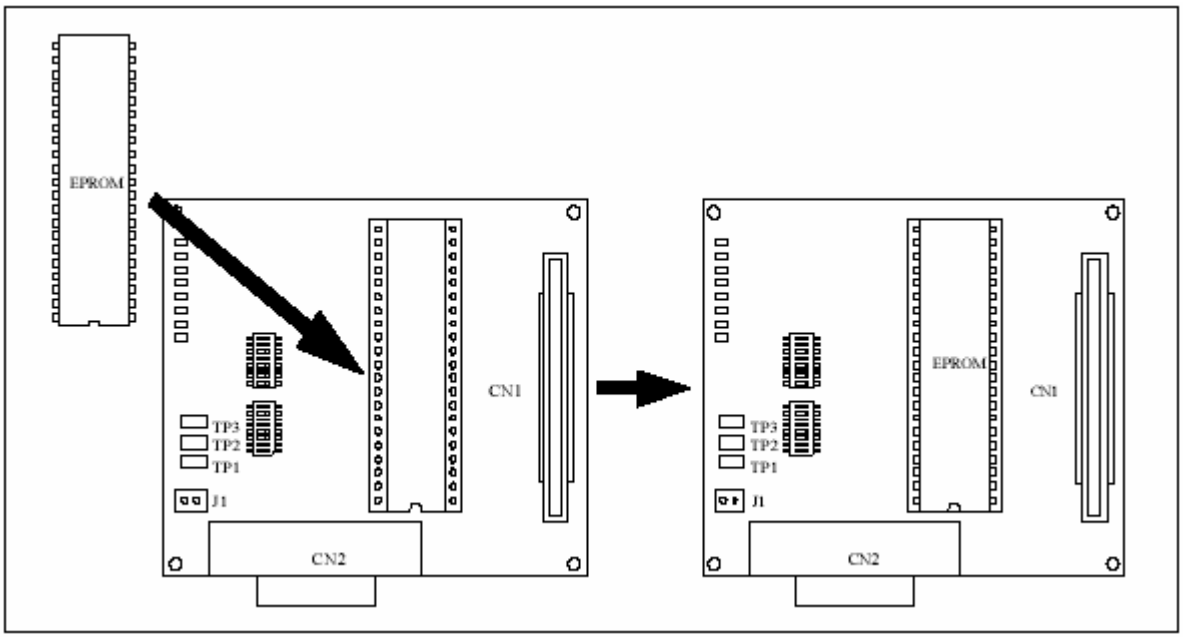
Kết nối debug board

Kết nối debug board ở khe mở rộng CN2 trên T-Engine board.



Hình 3.12**:** Kết nối debug board.

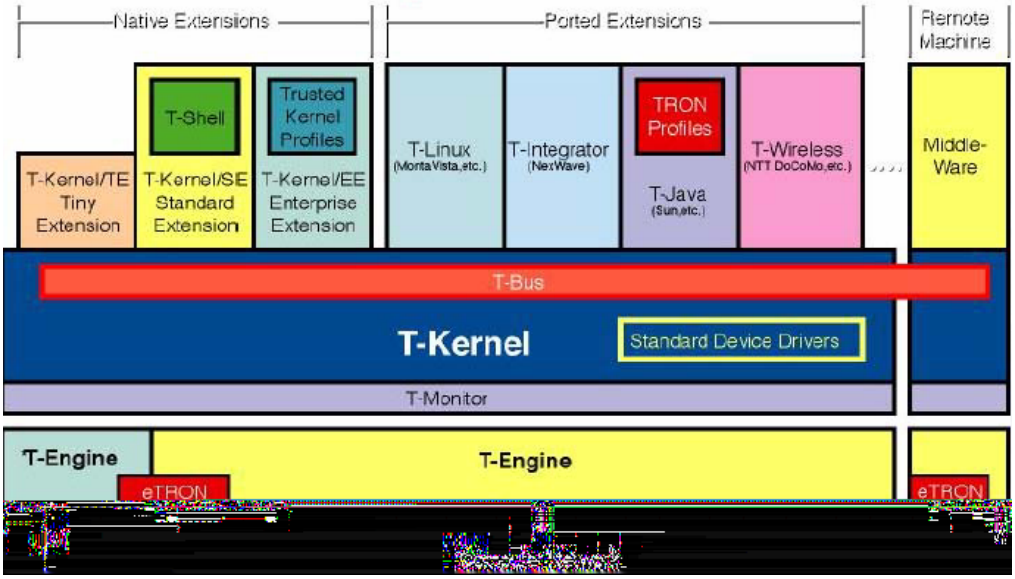
Lưu ý: Tắt nguồn T-Engine trước khi nối debug board hay tháo EPROM.Khi gắn lại EPROM kiểm tra chiều kết nối như hình 2.5



Hình 3.13 **:** Kết nối EPROM

❖Chi tiết hơn về các đặc điểm kỹ thuật của T-Engine SH7760 có thể tham khảo tài liệu: **SH7760 T-Engine Development Kit – User's Manual**

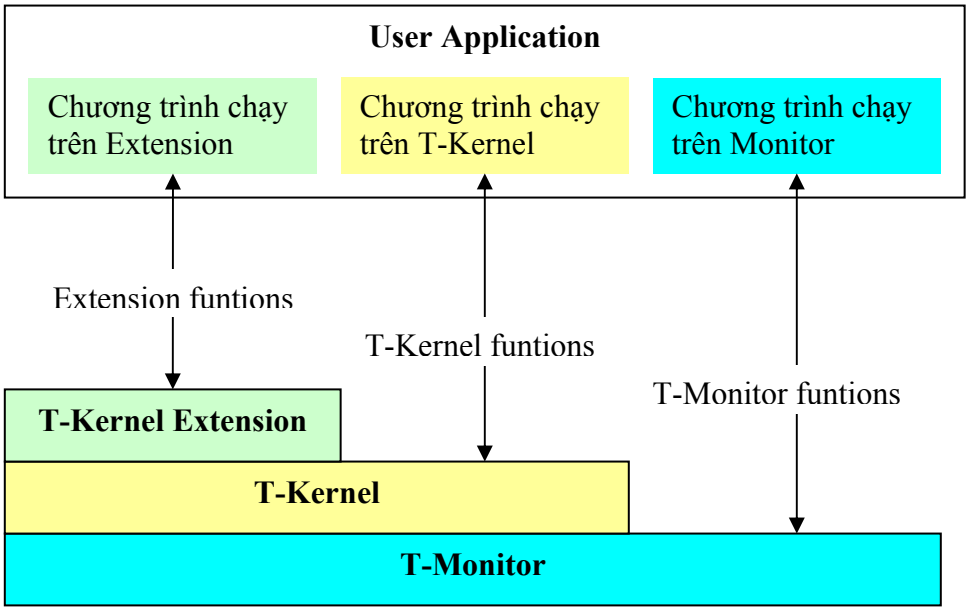
2.1.4 Kiến trúc của T-Engine :



Hình 3.14 Kiến trúc tổng quan T-Engine

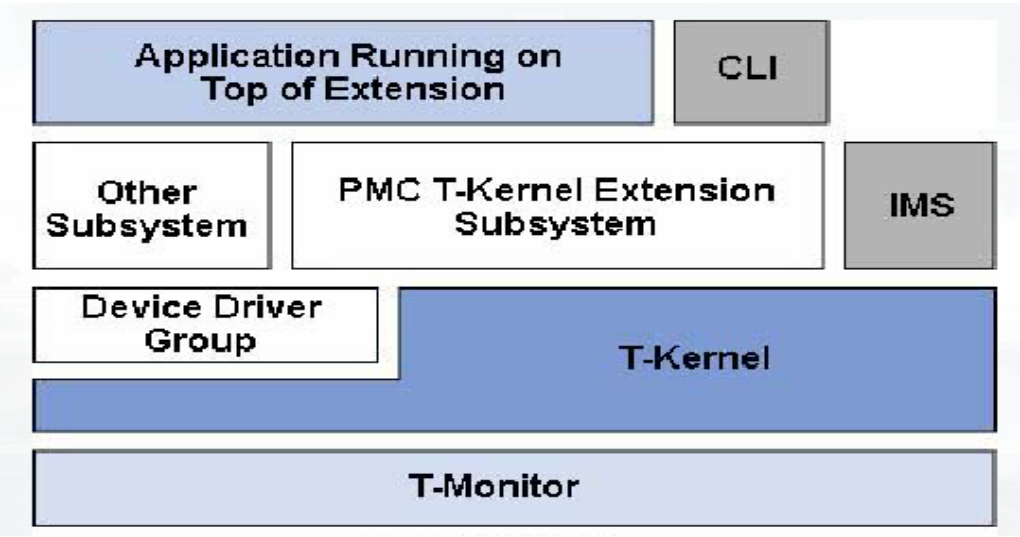
- **T-Monitor**
  - Có chức năng khởi tạo phần cứng và khởi động hệ thống, có thêm các hàm debug cơ bản.
  - Có chức năng tương tự như "PC BIOS" và thêm debug.
- **T-Kernel**
  - Hệ điều hành thời gian thực cho T-Engine.
  - Cung cấp các hàm quản lý và điều khiển cơ bản cho chương trình, không cung cấp các hàm cho hệ thống file và giao thức mạng, những hàm này thường được cung cấp bởi middleware.
- **Device drivers**
  - Điều khiển phần cứng.
  - Được đặt dưới sự quản lý của T-Kernel, và các ứng dụng sử dụng device driver thông qua các hàm dịch vụ của T-kernel.
- **T-Kernel Extension**
  - Mở rộng các chức năng của T-kernel,cung cấp các hàm của mức hệ thống cao hơn như quản lý hệ thống file.
- **User Applications**
  - Chương trình chạy trên T-Monitor : chủ yếu để kiểm tra phần cứng và debug (firmware).
  - Chương trình chạy trên T-Kernel : đây là hình thức cơ bản của một trình ứng dụng, nó chứa một hay nhiều task, và các hàm system call của T-Kernel là có thể được sử dụng trong mỗi task.
  - Chương trình chạy trên Standard Extension :
    - Đây là chương trình với đơn vị nhỏ nhất là các process, và có thể sử dụng các hàm system call do Extension cung cấp hơn là các hàm của T-Kernel.

- Các hàm system call bao gồm các hàm điều khiển process, file và giao tiếp giữa các process.



Hình 3.15 Những dạng của một trình ứng dụng

2.2 Tổng quan về Hệ thống



Hình 3.16 IMS và CLI

- **Công cụ phát triển IMS (Initial Monitor System)**

IMS là một chương trình chạy như là một task khởi tạo T-Kernel,xây dựng các hàm sau:

- Tham khảo,thao tác các trạng thái của T-Kernel bằng lệnh.
- Load và unload các chương trình hệ thống (subsystems)
- Thực thi chương trình hệ thống (processes).
- Thực thi command file.
- Tự động thực thi khởi tạo hệ thống bắt đầu lệnh STARTUP.CMD

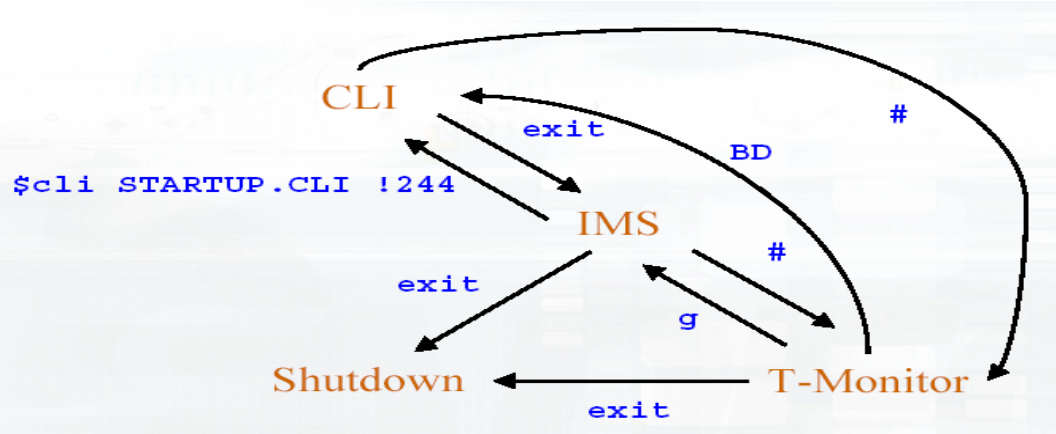
- **Công cụ phát triển CLI (Command Line Interpreter)**

CLI là một chương trình bắt đầu như một process của hệ thống.

CLI có những hàm

- Các loại khác nhau của những tác vụ file-centered tuỳ thuộc vào lệnh.
- Loading/unloading của subsystem.
- Thực thi các chương trình ứng dụng.
- Thực thi các command file.
- Lodspg và unlodspg có thể sử dụng để tự động load middleware và ứng dụng như một sự sắp xếp hệ thống để phân bố middleware.

Sự chuyển đổi giữa IMS,CLI,T-Monitor và shutdown modes.



2.3 Cài đặt môi trường phát triển (Cygwin)

- Cài đặt phần mềm terminal để giao tiếp : Tera_term.
- Cài đặt môi trường phát triển trên Cygwin (môi trường Linux trên Window)
- Kết nối T-Engine với máy tính.
- Tạo một đĩa boot và đĩa làm việc.

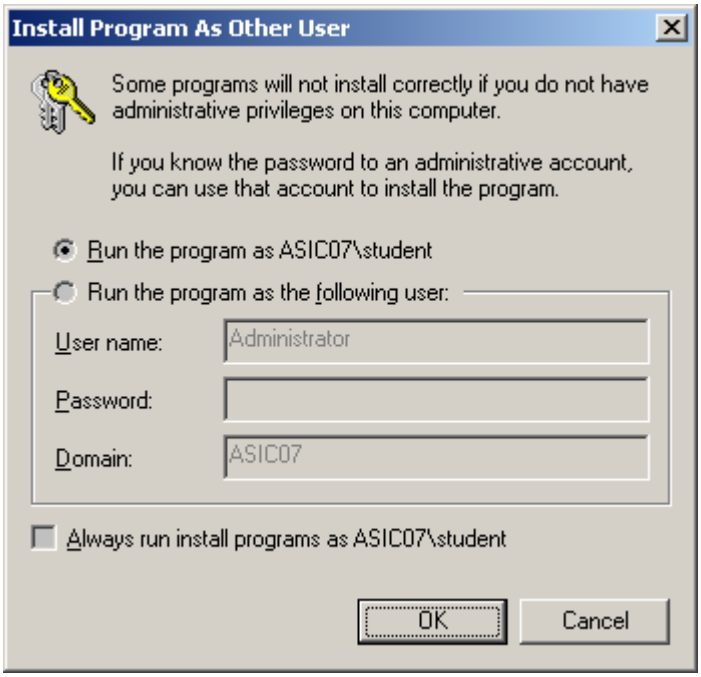
2.3.1 Cài đặt các phần mềm giao tiếp

TERA_TERM

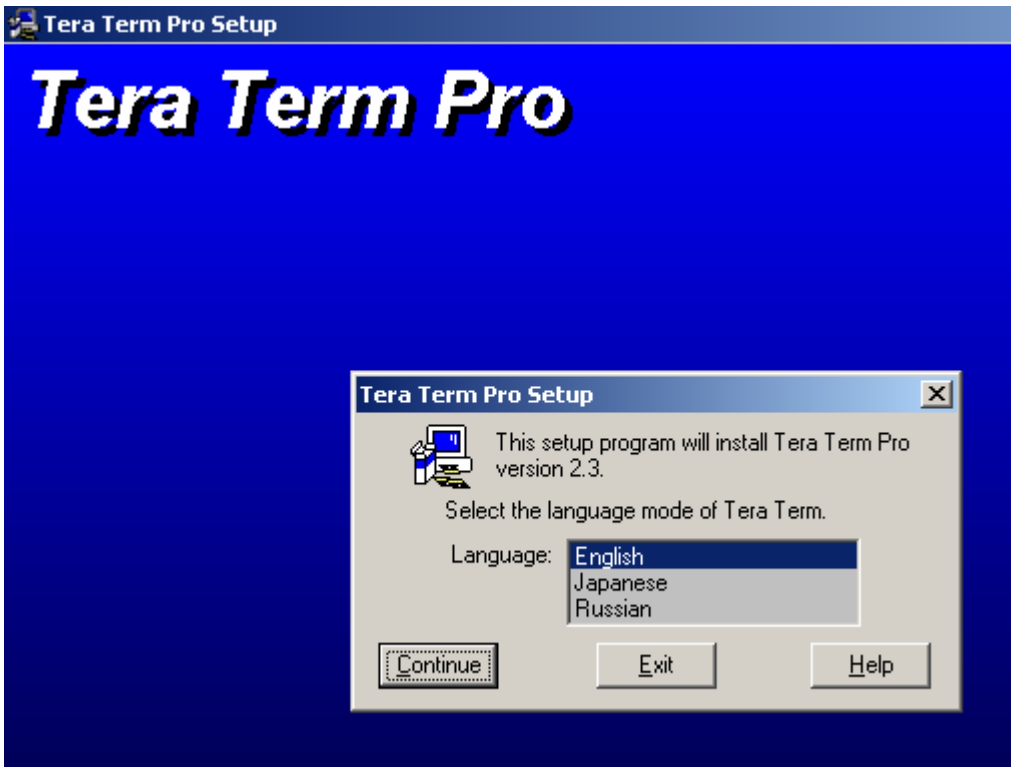
Tera Term (Pro) là một phần mềm mô phỏng đầu cuối cho Window. Nó có chức năng như là một telnet client kết nối với một telnet server chạy trên Cygwin. Phần mềm này còn được sử dụng để mô phỏng T-Engine. Tere term có thể được download tại http://hp.vector.co.jp/authors/VA002416/

Giải nén file **tterm23.zip**

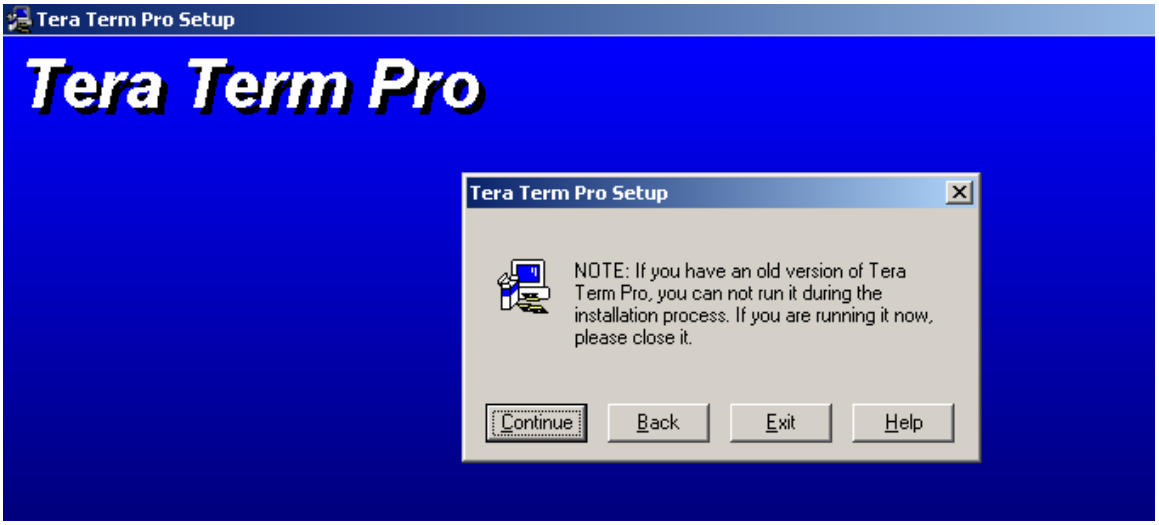
Chạy file **Setup.exe** để cài đặt



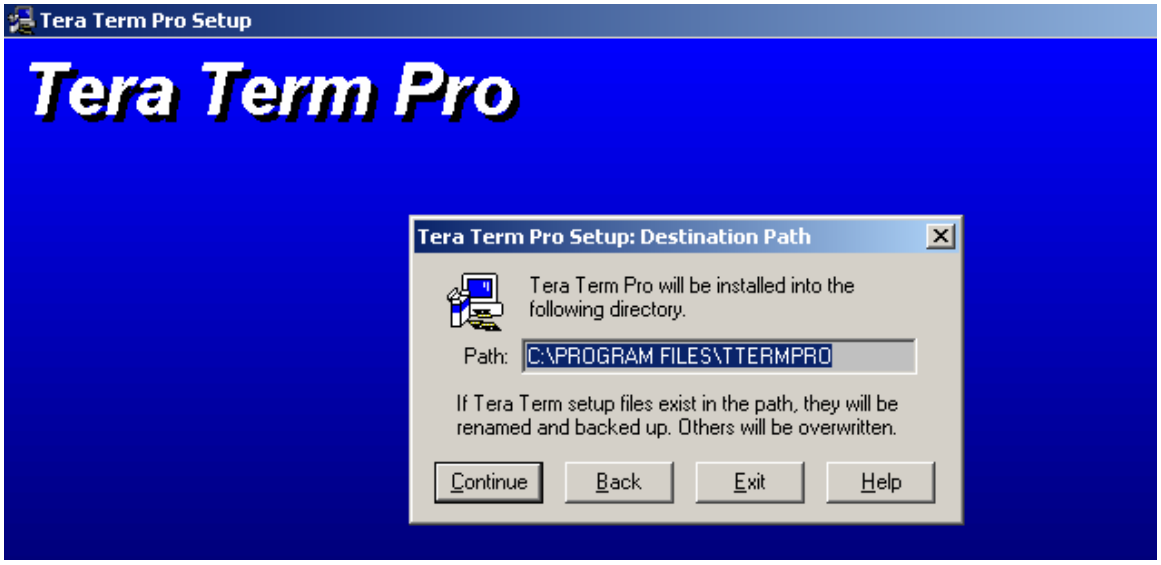
Nhấn OK để tiếp tục.



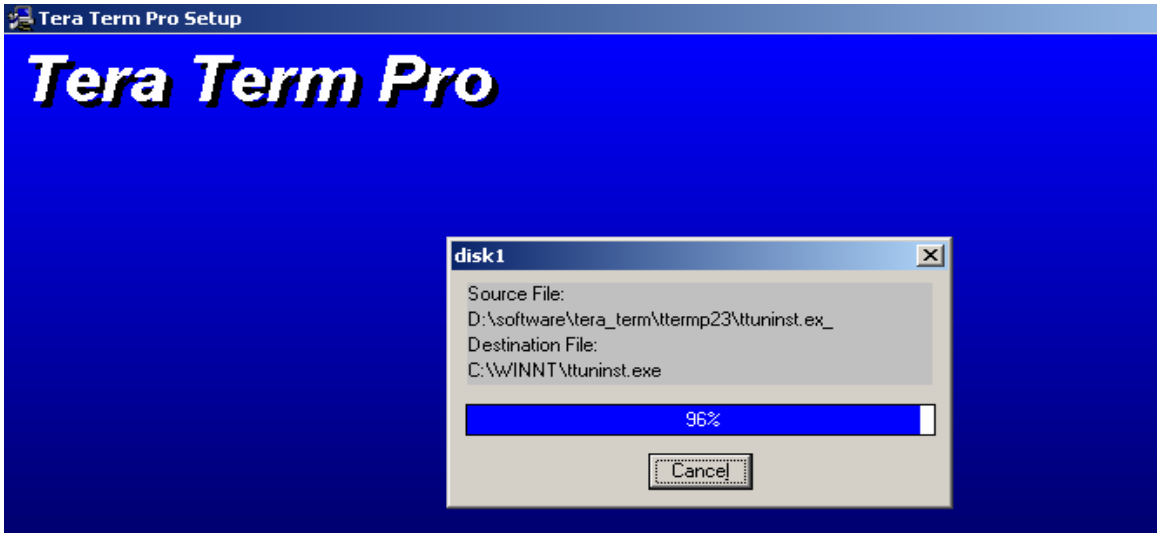
Chọn English và nhấn Continue để tiếp tục quá trình cài đặt.

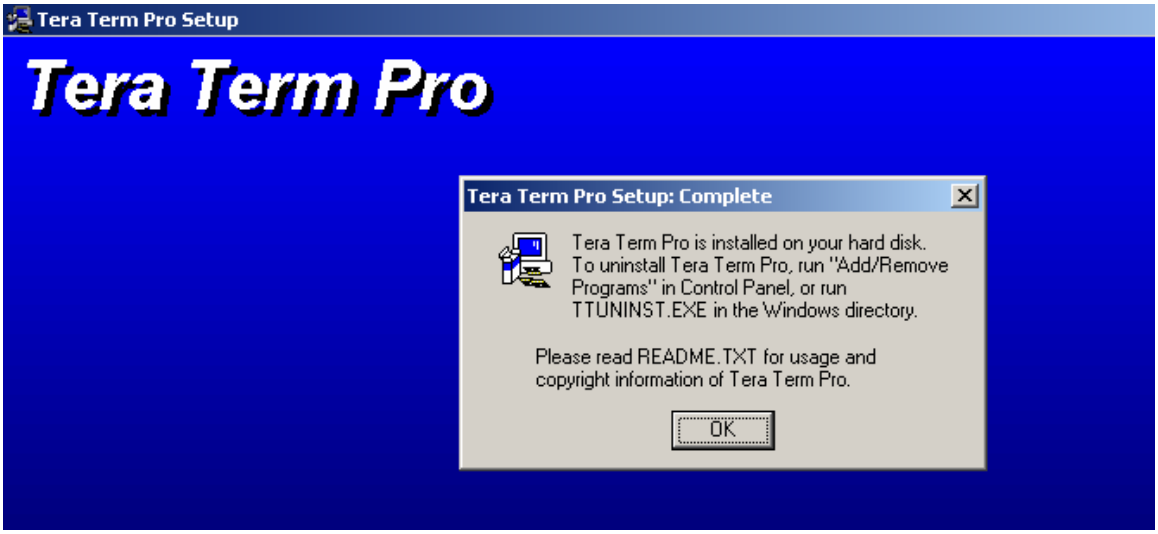


Nhấn Continue.



Chọn thư mục để cài đặt chương trình, sau đó nhấn Continue.





Nhấn OK kết thúc quá trình cài đặt.

2.3.2 Cài đặt môi trường phát triển trên Cygwin (môi trường Linux trên Window)

Khởi động Cygwin và Tera-term (from Start Menu):

Sau đó làm việc trên cửa sổ của Cygwin.

- Tạo một thư mục để cài đặt, ví dụ (nên dùng thư mục ten này) /usr/local/te.

  $ mkdir /usr/local/te

- Giải nén tất cả các packet GNU vào trong thư mục /usr/local/te.
  Nếu thư mục cài đặt là /usr/local/te, copy những packet dưới đây vào trong /usr/local/te (c:\cygwin\usr\local\te on Windows).

  te.resource.sh7760.tar.gz
  te.Cygwin-i686.common.04.tar.gz
  te.Cygwin-i686.sh.04.tar.gz

- Giải nén những file trên như sau :

  $ cd /usr/local/te

  $ tar zxpf te.resource.sh7760.tar.gz

  $ tar zxpf te.Cygwin-i686.common.04.tar.gz

  $ tar zxpf te.Cygwin-i686.sh.04.tar.gz

- Copy tool

  $cp –r /usr/local/te/tool/Cygwin-i686/etc /usr/local/te

- Những công cụ sau là cần thiết cho việc sử dụng môi trường phát triển GNU,không cần thiết phải cùng version.

  GNU make version 3.78.1
  perl version 5.005.03
  Đường thực thi của perl phải là /usr/local/bin/perl.Nếu không tìm thấy perl trong /usr/local/bin cần phải tạo link từ perl tới /usr/local/bin.

  $ cd/usr/local/bin; ln –s /usr/bin/perl

$ cd/usr/bin; ln –s make gmake

- Cập nhật lại các biến môi trường
  Sau khi cài đặt các công cụ phát triển cần phải cập nhật lại các biến môi trường cho phù hợp với môi trường hệ thống.Giá trị mặc định của thư mục cơ sở này là /usr/local/te
  Nếu công cụ phát triển được cài đặt trong /usr/local/te, trùng với thư mục mặc định thì các biến môi trường như BD,GNUs,GNU_BD,GNUsh,GCC_EXEC_PREFIX không cần cập nhật lại.
  Nếu các công cụ phát triển được cài đặt trong thư mục khác /usr/local/te các biến môi trường phải được cập nhật lại
  Sử dụng bash

  $ export BD= /usr/local/te

  $ export GNUs =/usr

  $ export GNU_BD= $BD/tool/Cygwin-i686

  $ export GNUsh =$GNU_BD/sh-unknown-tkernel

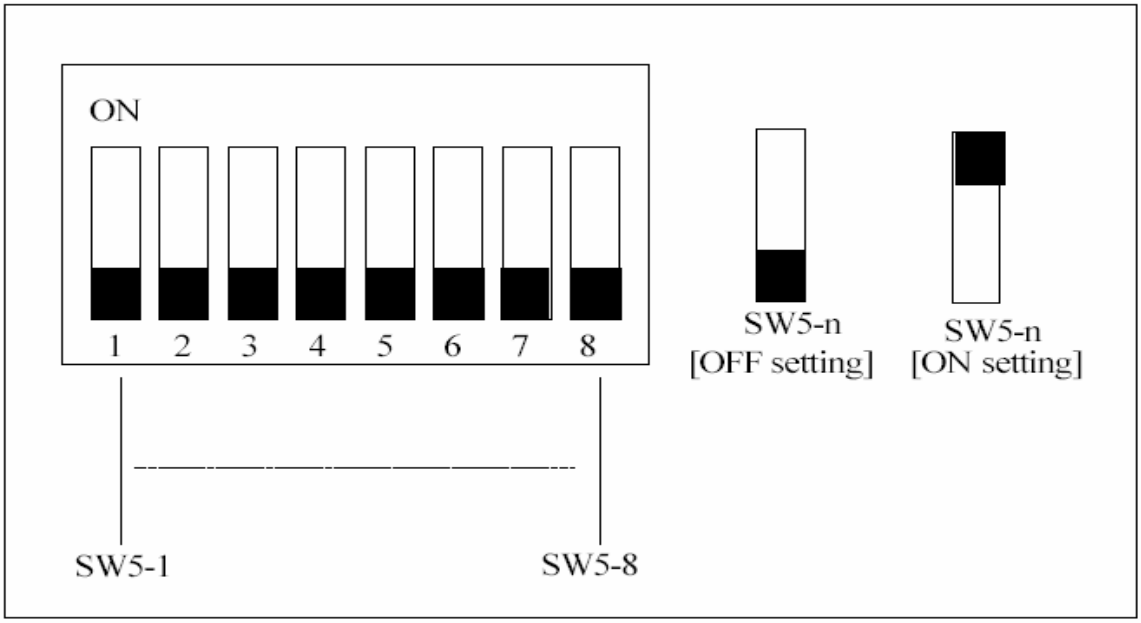
  $ export GCC_EXEC_PREFIX =$GNU_BD/lib/gcc-lib/

Những cập nhật trên có thể thực hiện trong .bashrc

2.3.3 Kết nối T-Engine với máy tính

Nối T-Engine với máy tính thông qua cổng COM1.

Thiết lập DIP-SW trên board



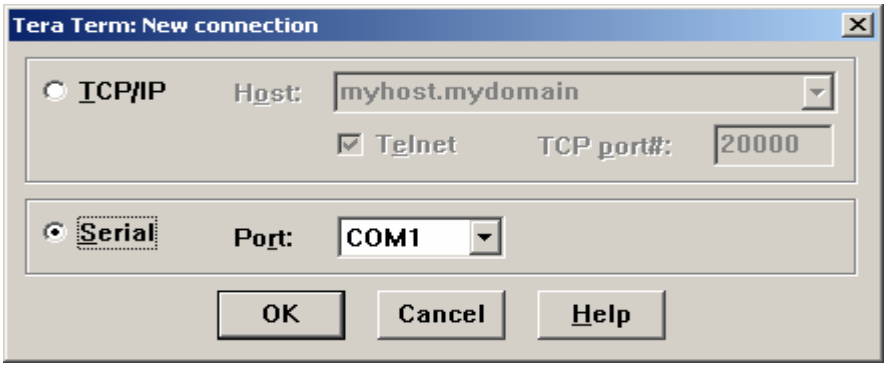
Hình : Thiết lập 8 bit DIP switch

- DIPSW5-1 : tự động boot (tắt: tự động boot,bật : T-Monitor)
- DIPSW5-2 : tốc độ port tuần tự (tắt :32400 bps,bật : 115200 bps)
- DIPSW5-4 : CPU clock (tắt :96MHz, bật :144 MHz)

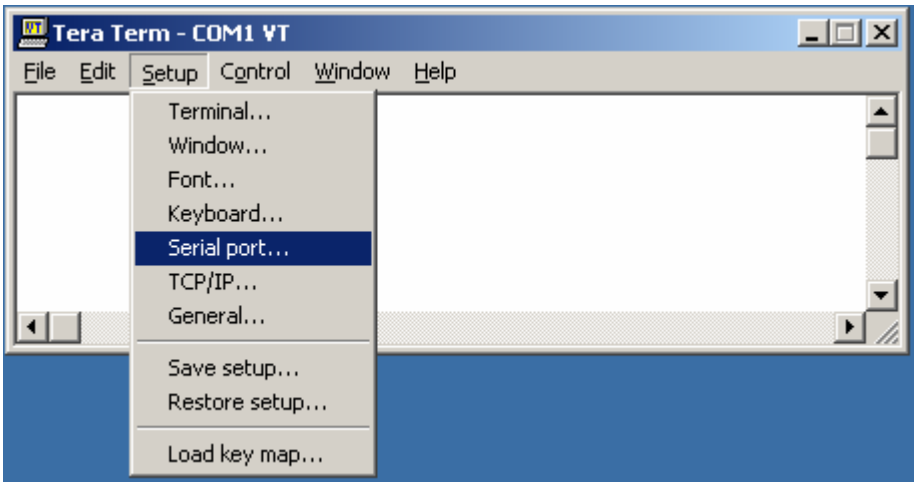
Khởi động Cygwin và Tera-term

Sau đó làm việc trên cửa sổ của Cygwin.

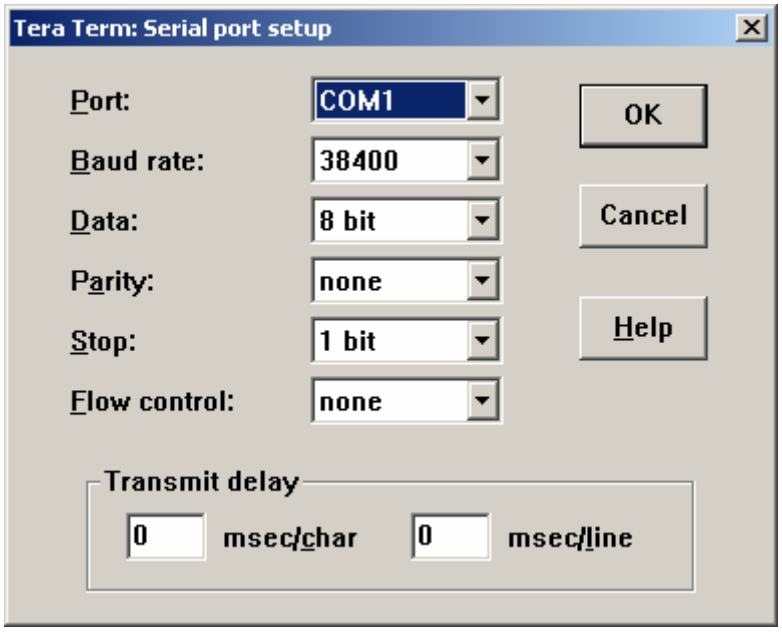
Khi khởi động Tera-term, một cửa sổ,ta chọn *"Serial"* và nhấn OK.



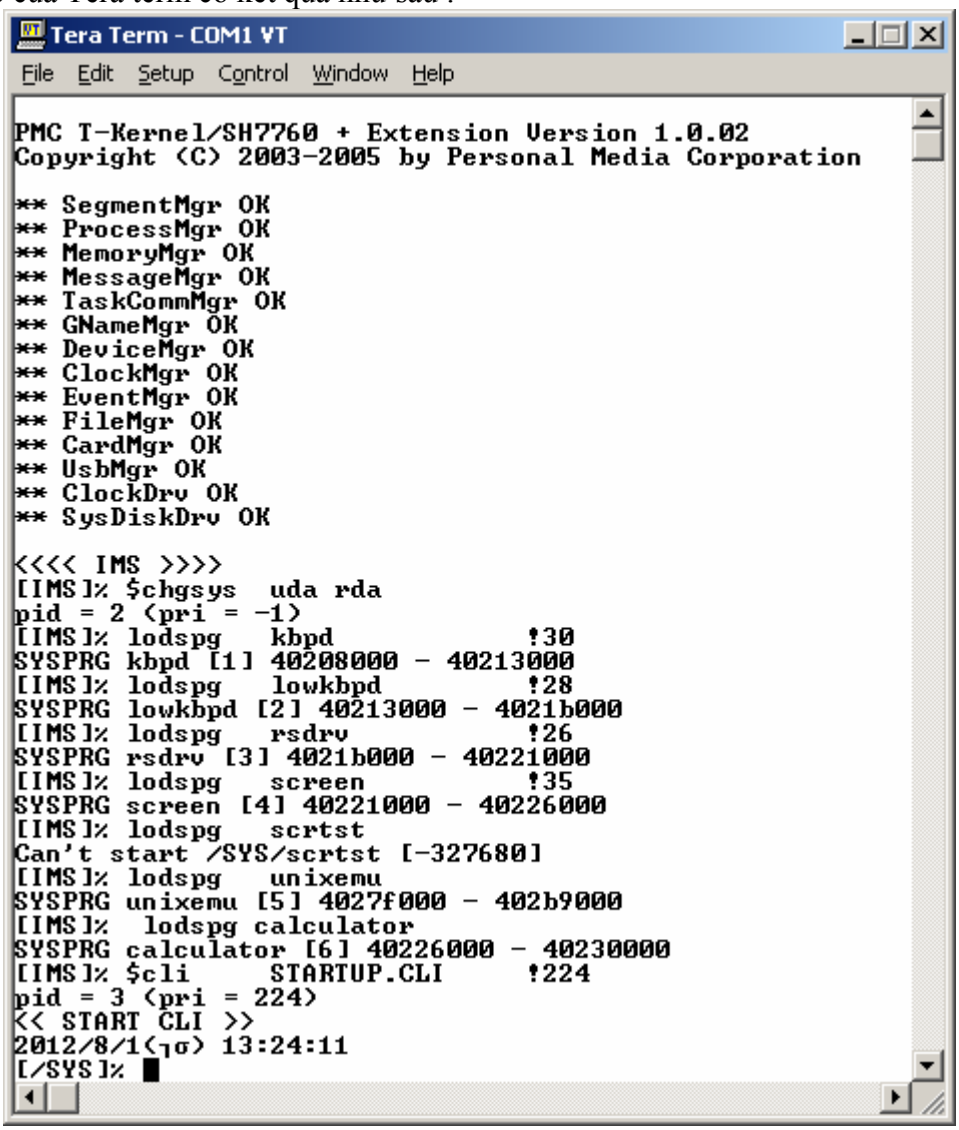
Sau đó cửa sổ mới của Tera term sẽ hiển thị lên, ta sẽ giao tiếp với T-Engine thông qua cửa sổ này. Trên cửa sổ "Tera term – COM1 VT", chọn *"Setup → Serial port"* như sau



Sau đó thiết lập các thông số như sau và nhấn OK :



Sau khi thiết lập thuộc tính, chọn menu *"Setup → Save setup"* để lưu file cài đặt với tên tương tự. Để khởi động T-Engine ta mở nguồn cho T-Engine (SW1), sau khi mở nguồn T-Engine dễ khởi dộng và trên cửa sổ của Tera term có kết quả như sau :



Khi hệ thống khởi động trình tự sau sẽ được thực thi

- (1) T-Monitor,T-Kernel,T-Extension và các drives cần thiết khởi động.
- (2) IMS khởi động như task ban đầu.
- (3) IMS thực thi /SYS/STARTUP.CMD.Kết quả tiếp theo CLI khởi động.

Khi CLI khởi động,IMS bản thân nó chuyển sang trạng thái wait exit.
Thực thi lệnh thoát CLI,quyền thực thi trả lại cho IMS. Nhập một file thực thi với '$' bắt đầu nó sẽ bắt đầu file thực thi như là một process trên IMS.

Ví dụ : $cli STARTUP.CLI !244

Các chỉ thị có thể xem nội dung của file /SYS/STARTUP.CMD:

[/SYS] % tp –a /SYS/STARTUP/CMD

Các thủ tục khởi động hệ thống

- T-Monitor tìm kiếm các thiết bị có thể boot từ đó.Sau đó tải và chạy boot block(chương trình boot chính –PBOOT) trên đĩa khởi động.
- PBOOT tìm các file hệ thống trên chương trình boot thứ cấp(SBOOT) ,tải nó vào bộ nhớ và chạy.
- SBOOT tải các file hệ thống sau vào bộ nhớ và chạy:
  KERNEL.SYS.
  KERNEL.SYS OS Kernel (T-Kernel,Extension,drivers)
  SYSCONF file cấu hình hệ thống
  DEVCONF file cấu hình thiết bị
  KERNEL.SYS khởi tạo hệ thống dựa trên các thông số trong SYSCONF,sau đó chạy T-Kernel và T-Kernel Extension.Nó cũng khởi động các driver sau:
  Quản lý PC Card,quản lý USB,Clock(RTC),Console(serial),System disk
  Sau khi tất cả các OS khởi tạo hoàn tất,IMS được bắt đầu như task đầu tiên,file STARTUP.CMD được đọc và sử lý được xây dựng dựa trên nội dung của nó.
  STARTUP.CMD khởi động các driver khác,và cuối cùng chạy CLI (STARTUP.CLI).

Xem nội dung của file STARTUP.CMD

```
[/SYS]% ed -a STARTUP.CMD
Record len = 289
>>% p40
+ 0: *
+ 1: *  @(#)STARTUP.CMD (T-Engine/SH7760)
+ 2: *
+ 3: *  IMS initial command script for booting system
+ 4: *  Copyright (C) 2004 by Personal Media Corporation
+ 5: *
+ 6: $chgsys   uda rda
+ 7: lodspg    kbpd        !30
+ 8: lodspg    lowkbpd     !28
+ 9: lodspg    rsdrv       !26
+10: lodspg    screen      !35
+11: lodspg    scrtst
+12: lodspg    unixemu
+13: $cli      STARTUP.CLI    !224
+14: exit      1
>>%q
```

Thêm chương trình của ta vào trước dòng này để thực thi nó tự động khi khởi động hệ thống.Ví dụ:lodspg abc

Tạo đĩa làm việc.

Các phần mềm không thể chứa trong ROM của flash memory do đó cần phải tạo đĩa làm việc trước tiên để lưu trữ các phần mềm phát triển.

Các thiết bị sau có thể được sử dụng như đĩa ghi:

- ATA card,hay CF card + PC-Card adapter
  Tên thiết bị : pca

- USB
  Tên thiết bị : uda

Để những đĩa này có thể sử dụng được cần phải phân vùng và format bằng các thủ tục cơ bản của CLI

```
[/SYS]% hdpart pca (or uda)
uda [C:249 H:4 S:16 B:16000 (7 MB)]
No      System      Boot   StartCHS   EndCHS      SecNo   SecCnt   MB
1       01 DOS      00     0: 1: 1    249: 3:16   16      15984    7
2       00 ------   00     0: 0: 0    0: 0: 0     0       0        0
3       00 ------   00     0: 0: 0    0: 0: 0     0       0        0
4       00 ------   00     0: 0: 0    0: 0: 0     0       0        0
** Create/Delete/Boot/Edit/Quit ? c
Create PartNo (1-4) ? 1
Size [MB] (<8MB) ?
No      System      Boot   StartCHS   End CHS     SecNo   SecCnt   MB
1       13 BTRON    00     0: 1: 1    249: 3:16   16      15984    7
2       00 ------   00     0: 0: 0    0: 0: 0     0       0        0
3       00 ------   00     0: 0: 0    0: 0: 0     0       0        0
4       00 ------   00     0: 0: 0    0: 0: 0     0       0        0
** Create/Delete/Boot/Edit/Update/Quit ? b
Boot PartNo (1-4) ? 1
No      System      Boot   StartCHS   End CHS     SecNo   SecCnt   MB
1       13 BTRON    80     0: 1: 1    249: 3:16   16      15984    d7
2       00 ------   00     0: 0: 0    0: 0: 0     0       0        0
3       00 ------   00     0: 0: 0    0: 0: 0     0       0        0
4       00 ------   00     0: 0: 0    0: 0: 0     0       0        0
** Create/Delete/Boot/Edit/Update/Quit ? u
** uda: Updated Master Boot Block
[/SYS]% format pca0 WORK
Format pca0 [STD] WORK
Logical Formatting...
Disk Format Success.
```

Lựa chọn –x trong lệnh format khi kích thước vùng chia lớn hơn hay bằng 2 GB.

```
[/SYS] % format  pca0 WORK
```

Sau khi format hoàn tất sử dụng các lệnh sau của CLI để có thể truy cập đĩa:

```
[/SYS] % att pca0 A

[/SYS] % cd /A
```

Tạo đĩa boot

Việc tạo đĩa boot cũng tương tự như như tạo đĩa làm việc. Nếu đĩa boot được tạo là PC-Card hay USB cắm vào T-Engine,thứ tự boot của hệ thống tùy thuộc vào cách thiết lập SW-1 và SW-3

DIP SW–1	DIP SW–3	Boot order
OFF	OFF	PC-Card → ROM disk
OFF	ON	PC-Card → USB storage → ROM disk
ON	OFF	T-Monitor
ON	ON	T-Monitor

Nếu boot từ USB,hệ thống sẽ boot từ ROM trước,sau đó hệ thống đĩa đượcchuyển từ ROM sang USB.Do đó những file sau trong USB không sử dụng

Lưu ý rằng chúng không ảnh hưởng đến việc boot của hệ thống.

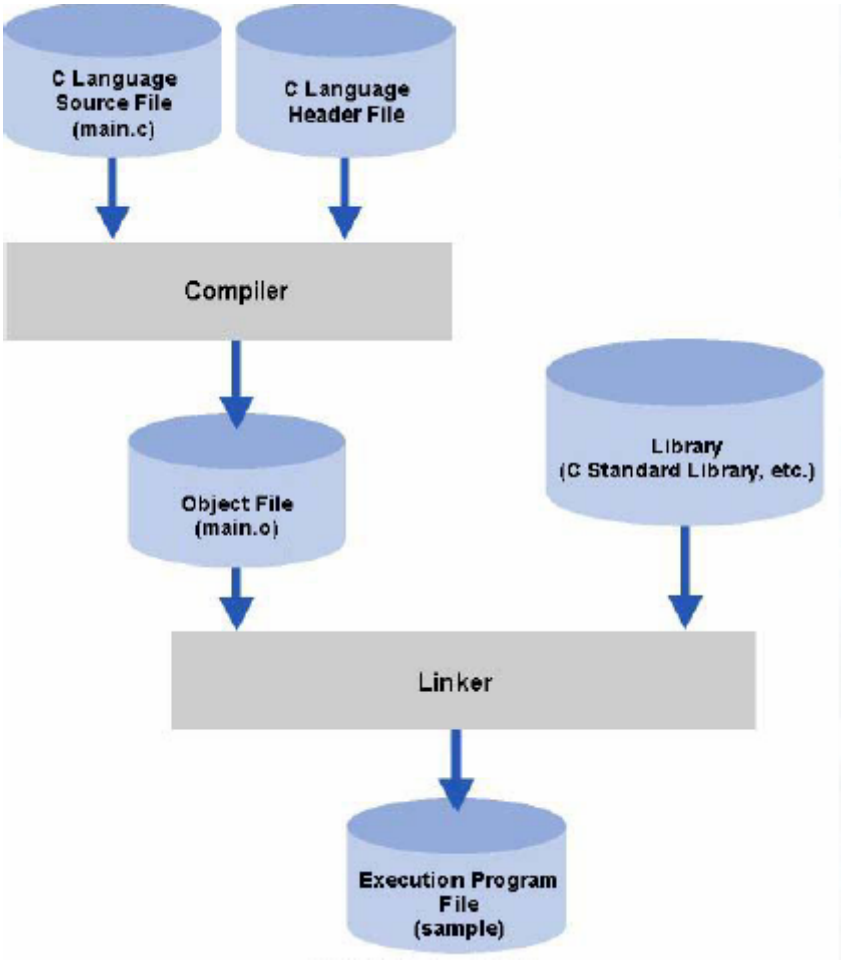
PBOOT,SBOOT,KERNEL.SYS,SYSCONF,DEVCONF,STARTUP.CMD,chgsys

Nếu DIP SW-3 được bật,hệ thống sẽ kiểm tra thiết bị USB.Nếu USB không được gắn việc boot sẽ tốn thời gian.Nếu hệ thống không cần boot từ USB DIP SW-3 nên tắt.

Các bước tạo đĩa boot:

- Chia vùng boot sử dụng tiện ích hpart.
  Trong lệnh format,chỉ định –b để yêu cầu ghi chương trình boot.Chỉ định lựa chọn –x nếu vùng chia có kích thước lớn hơn hay bằng 2 GB.
  [/SYS] % format –b pca0 SYSTEM
  Copy các chương trình từ ROM sang đĩa boot
  [/SYS] % att pca0 /A
  [/SYS] % cp –b –v –r * /A

2.4 Biên dịch và liên kết chương trình chạy trên t-kernel



Hình :Xây dựng process

Trước khi truyền nhận một chương trình T-Kernel sang T-Engine, thì cần phải tạo đĩa work hay boot bằng CF card (pca) hoặc USB (uda).

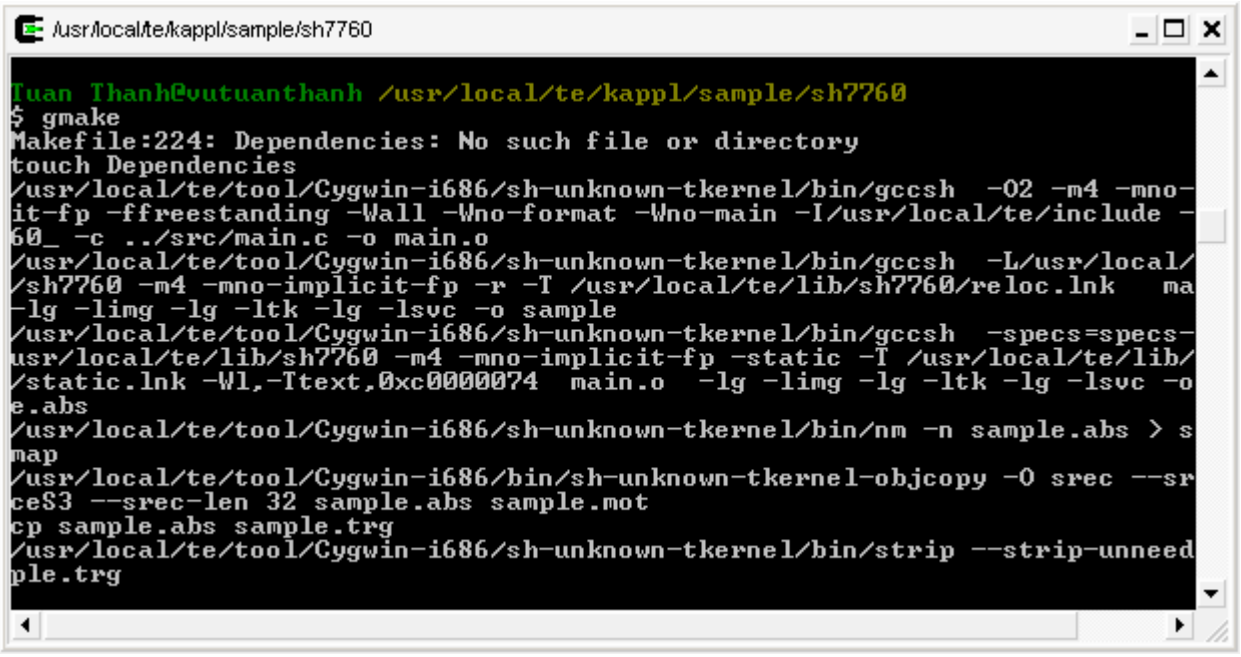
Thư mục chứa các ví dụ chương trình T-Kernel là /usr/local/te/kapple/, sau này khi tạo các chương trình mới cũng nên chứa trong thư mục này.

Compile một chương trình mẫu ví dụ có tên là sample:

$ cd /usr/local/te/kappl/drawsamp/sh7760

$ gmake

Khi biên dịch cửa sổ Tera term có kết quả sau



Sau khi biên dịch, ta chọn menu *"File → New conecction"* giống như thao tác ở phần kết nối với T_Engine.

Nếu T-Engine boot từ USB hay CF card (bạn đã tạo đĩa boot) thì có thể bỏ qua lệnh *att* mà làm việc ngay trên thư mục /SYS, nếu T-Engine boot từ ROM thì phải tạo đĩa làm việc WORK (như phần C.3.1) và thực hiện doạn lệnh sau trong cửa sổ "Tera term – COM1 VT"

[/SYS] % att pca0 A (hoặc uda0)

[/SYS] % cd /A

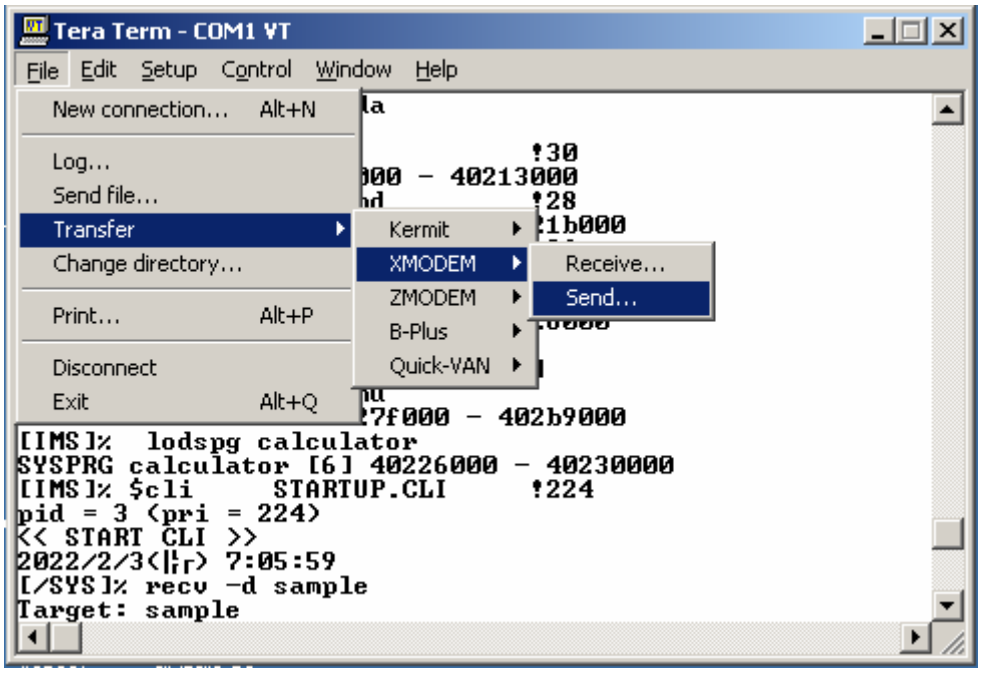
Chép chương trình vào đĩa làm việc

[/A] % recv –d sample

Hoặc nếu CF card la đĩa boot thì không cần att

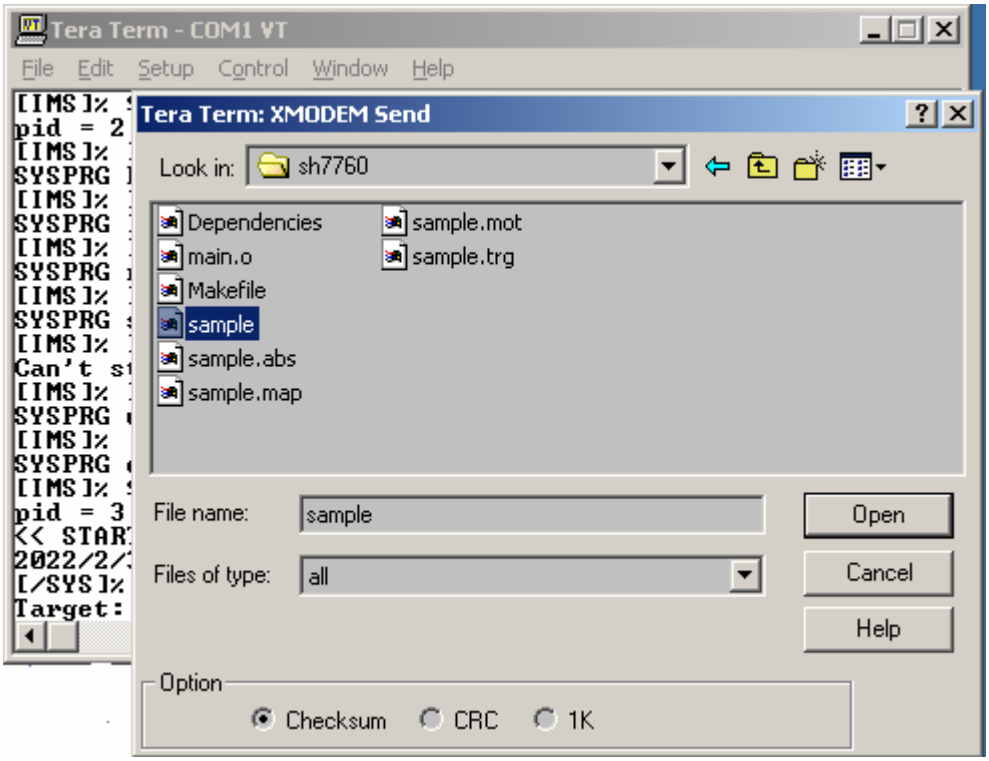
[/A] % recv –d sample

Lựa chọn –d trong lệnh recv dùng để ghi đè lên các chương trình cùng tên.

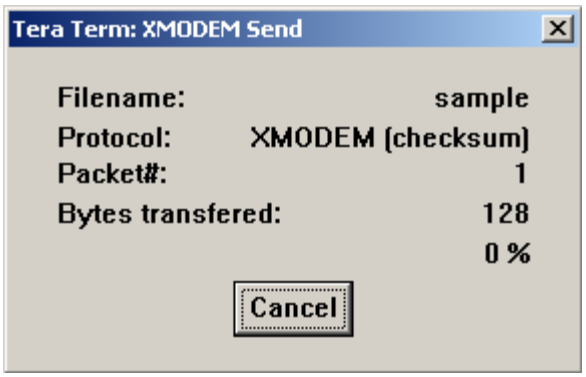


Bây giờ CLI đang chờ nhận chương trình, vào menu *"File"* → *"Tranfer"* → *"XMODEM"* → *"Send"*

Chọn file chương trình muốn gửi

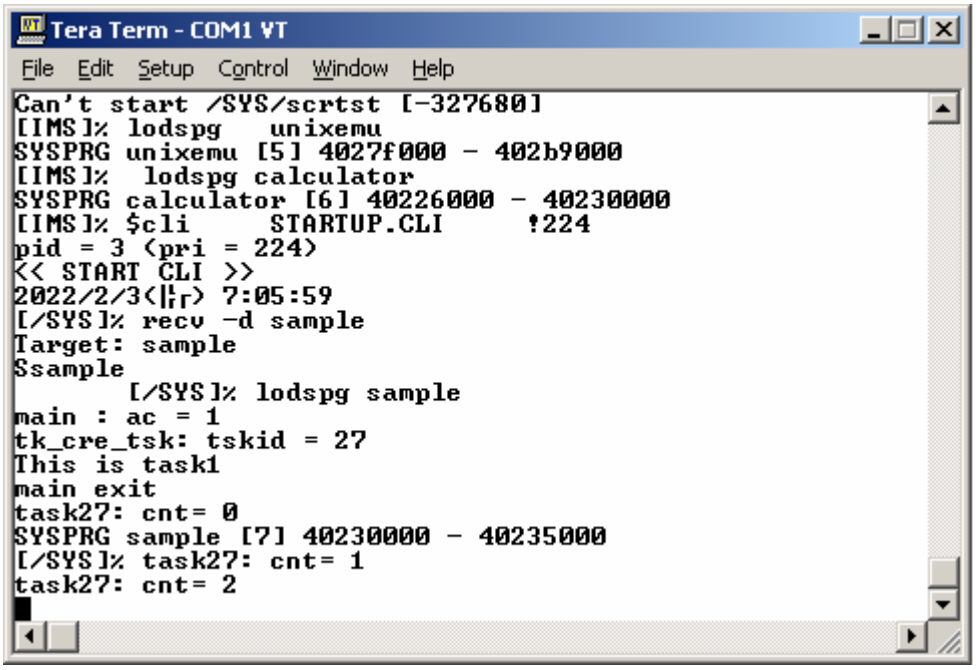


Sau đó hộp thọa truyền nhận sẽ hiển thị



Chi chương trình từ đĩa làm việc lên ROM sử dụng lệnh lodspg của CLI

[/SYS] % lodspg sample



Để kết thúc sự thực thi của chương trình sử dụng lệnh unlspg

[/SYS] % unlspg 7 (số bên trong [ ] của 'SYSPRG .. [7])

Để tắt T-Engine

[/SYS] % exit

[/IMS]% exit

2.5 Một số lệnh cơ bản của CLI và IMS : Xem phần phụ lục

3. THỰC HÀNH

3.1 Thực hiện cài đặt Tera Term và Cygterm :

Như hướng dẫn ở phần Lý thuyết 2.3.1.

3.2 Thực hiện cài đặt môi trường phát triển cho T-Engine :

Như phần 2.3.2.

3.3 Khởi động T-Engine, tạo đĩa boot và đĩa làm việc :

Như phần 2.3.3.

3.4 Dịch và thực thi chương trình *sample* trong phần 2.4 :

Đọc file main.c trong thư mục ../kapple/sample/src, nhận xét.

3.5 Dịch và thực thi chương trình *drawsamp* :

Thư mục chứa drawsamp là /usr/local/te/kapple/drawsamp, thực thi chương trình và nhận xét các chức năng do chương trình cung cấp

BÀI THỰC HÀNH SỐ 4

CẤU TRÚC MỘT TRÌNH ỨNG DỤNG CÁC HÀM QUẢN LÝ TASK

1. MỤC TIÊU

- Hiểu về kiến trúc của một chương trình ứng dụng trong T-Engine
- Có khả năng viết được một chương trình đơn giản trên T-Engine

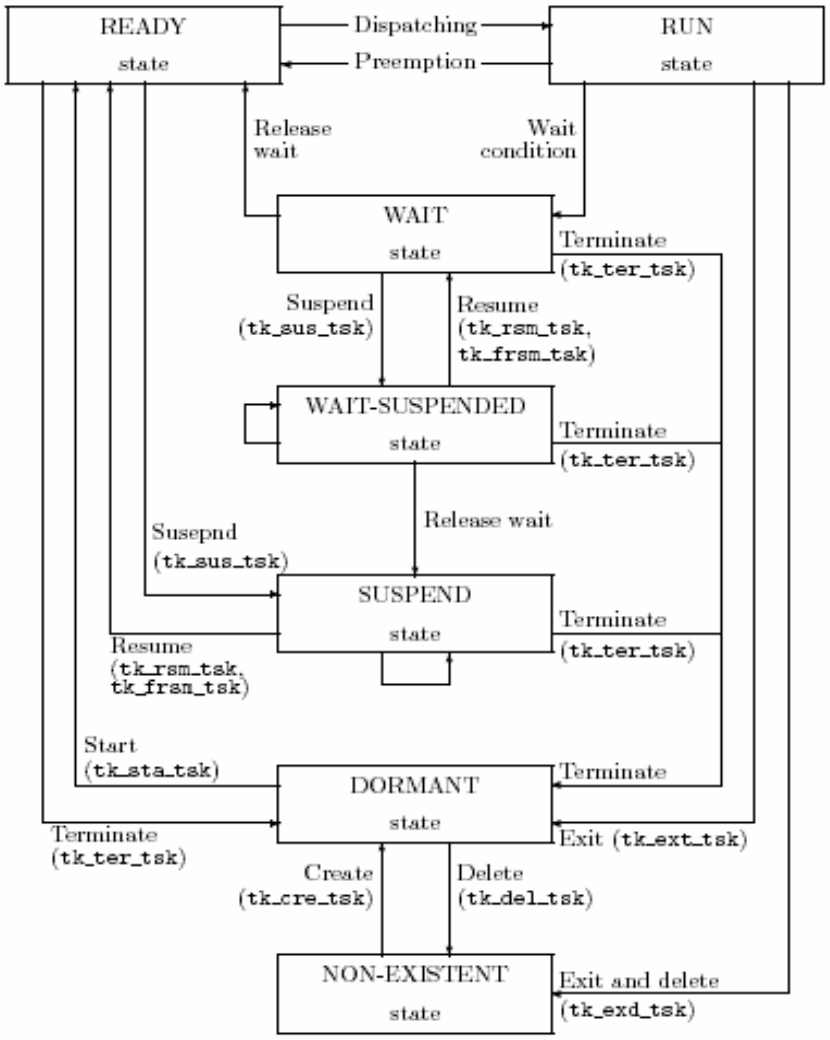
2. LÝ THUYẾT

2.1 Tổng quan về T-Kernel

T-Kernel là một hệ điều hành real-time do đó trong mỗi công việc của T-Kernel ta điều có thể thiết lập một thông số là thời gian timeout cho nó.

Đơn vị luận lí cơ bản của một chương trình đang thực thi trong T-Kernel được gọi là một task. T-Kernel định thời giữa các task theo cơ chế "preemptive" dựa trên độ ưu tiên giữa các task. Độ ưu tiên trong T-Kernel của các task có tổng cộng là 140.

Các trạng thái của một task được quản lý bởi T-Kernel được mô tả bởi hình bên dưới



Trạng thái của task được phân làm 5 loại chính ,trong đó thì trạng thái WAIT được phân làm 3 loại nhỏ hơn.

(a) RUN state

Một task đang ở trạng thái RUN state có nghĩa là task đó hiện thời đang được thực thi. Khi mà một task độc lập với ngữ cảnh (Task-independent portion) được thực thi thì task mà thực thi ngay trước khi bắt đầu sự thực thi của Task-independent portion được nói là đang ở trong trạng thái Run state.

(b) READY state

Task mà đã hoàn tất quá trình chuẩn bị cho sự thực thi nhưng chưa chuyển sang trạng thái RUN được vì task có độ ưu tiên cao hơn đang thực thi. Trong trường hợp này , thì task sẽ được thực thi bất cứ khi nào nó trở thành task có độ ưu tiên cao nhất giữa các task trong trạng thái READY

(c) WAIT states

Task không thể được thực thi bởi vì những điều kiện cho sự thực thi chưa có sẵn. Nói cách khác là task đang chờ đợi những điều kiện cho sự thực thi được thỏa mãn. Trong khi một task đang ở trạng thái WAIT thì các giá trị trong các thanh ghi và của program counter cũng như một số thông tin khác

của chương trình đang đươc thực thi sẽ được lưu lại. Khi task được đánh thức để tiếp tục thực thi từ trạng thái này thì các thông tin được lưu lại sẽ được phục hồi như trước khi task đi vào trạng thái WAIT. Trạng thái này được chia làm 3 trạng thái nhỏ hơn.

(c.1) WAIT state

Sự thực thi sẽ bị ngừng lại bởi vì task đang thực thi vừa mới triệu gọi một hàm system call . Và sự thực thi chỉ có thể tiếp tục khi hàm system call này hoàn thành.

(c.2) SUSPEND state

Sự thực thi buộc phải ngưng lại bởi vì một task khác.

(c.3) WAIT-SUSPEND state

Task cùng lúc vừa ở trong trạng thái WAIT vừa ở trạng thái Suspend. WAIT-SUSPEND là kết quả khi một task nào đó yêu cầu Suspend một task đã ở trong trạng thái WAIT .T-Kernel phân biệt rõ ràng giữa 2 trạng thái WAIT và SUSPEND. Một task thì không thể tự nó đi vào trạng thái SUSPEND.

(d) DORMANT state

Task mà chưa được bắt đầu sự thực thi hay đã hoàn thành sự thực thi rồi. Trong khi một task đang ở trạng thái DORMANT thì tất cả thông tin thể hiện sự thực thi của nó sẽ không được lưu lại. Khi một task bắt đầu sự thực thi của nó từ trạng thái DORMANT thì sự thực thi của nó sẽ bắt đầu tại địa chỉ bắt đầu của task. Ngoại trừ những đặc tả khác thì giá trị của thanh ghi sẽ không được lưu lại.

(e) NON-EXISTENT state

Đây là một trạng thái ảo trước khi task được tạo ra , hoặc sau khi nó bị deleted hoặc chưa được đăng kí với hệ thống.

Phụ thuộc vào sự hiện thực mà ở đây có thể có thêm 1 số trạng thái tạm thời khác mà không thuộc vào bất kì trạng thái nào trong 5 trạng thái được liệt kê ở trên.

Khi một task chuyển sang trạng thái READY mà có độ ưu tiên cao hơn task đang thực thi hiện tại thì quá trình "dispatching" có thể xảy ra ngay khi task chuyển sang trạng thái READY và sau đó nó sẽ chuyển sang trạng thái RUN. Trong trường hợp này thì task mà đang thực thi cho tới thời điểm "dispatching" xảy ra được nói là nó đã bị tước đoạt quyền thực thi bởi task mới.

Một task được nói là bắt đầu sự thực thi có nghĩa là nó chuyển từ trạng thái DORMANT sang trạng thái READY. Do đó mà khi nói một task đã thực thi có nghĩa là task đang ở trong một trạng thái bất kì nào đó ngoài trạng thái DORMANT hay NON-EXISTENT . Và một task được nói là thoát khỏi sự thực thi có nghĩa là task đã đi đến trạng thái DORMANT.

Một task được nói là đã được giải phóng khỏi trạng thái "wait" có nghĩa là nếu task đang ở trạng thái WAIT thì sẽ chuyển sang trạng thái READY , còn nếu task đang ở trạng thái WAIT-SUSPEND thì sẽ chuyển sang trạng thái SUSPEND. Sự tiếp tục thực thi của một task ở trạng thái "suspend" có nghĩa là nếu task đang ở trạng thái SUSPEND thì sẽ chuyển sang trạng thái READY, và nếu như task đang ở trạng thái WAIT-SUSPEND thì sẽ chuyển sang trạng thái WAIT.

2.2 Các hàm API được cung cấp cho việc quản lý task

Đây là các hàm system call cho phép ta thao tác hay tham khảo trực tiếp các trạng thái của Task. Những hàm được cung cấp để tạo và xoá một Task, khởi động và thoát, huỷ bỏ một yêu cầu khởi động Task, thay đổi độ ưu tiên, và tham khảo trạng thái của một Task. Mỗi Task được định danh bởi một số ID gọi là *Task ID* .

Mỗi Task có một độ ưu tiên cơ bản và độ ưu tiên hiện tại để system kiểm soát thứ tự thực thi của các Task. Thông thường khi ta chỉ nói đến độ ưu tiên của Task mà không đề cập loại ưu tiên nào thì ta ngầm hiểu là độ ưu tiên hiện tại. Độ ưu tiên cơ bản của một Task được khởi tạo là độ ưu tiên khởi động khi khởi động một Task. Khi ta không sử dụng mutex thì độ ưu tiên hiện tại cũng chính là độ ưu tiên cơ bản, do vậy độ ưu tiên hiện tại được khởi tạo trực tiếp là độ ưu tiên khởi động của Task.

Kernel không thực hiện công việc giải phóng các tài nguyên do Task yêu cầu (semaphore resources, memory blocks…) khi thoát Task, công việc này là nhiệm vụ của ứng dụng.

Sau đây là các hàm API của hệ thống dùng để quản lý task.

❖ **Các hàm đồng bộ Task-Dependent:**

1. tk_cre_tsk : Tạo task

[C Language Interface]

```
ID  tskid  =  tk_cre_tsk ( T_CTSK *  pk_ctsk ) ;
```

[Parameters]

T_CTSK * pk_ctsk Thông tin về Task được tạo

Chi tiết của pk_ctsk :

VP	exinf	Thông tin phụ được thêm vào
ATR	tskatr	Các thuộc tính của Task
FP	task	Địa chỉ của Task
PRI	itskpri	Độ ưu tiên khởi động của Task
INT	stksz	Kích thước của stack (bytes)
INT	sstksz	Kích thước của stack hệ thống (bytes)
VP	stkptr	Pointer chỉ tới stack của người dùng
VP	uatb	Bảng không gian trang của Task
INT	lsid	ID không gian luận lý
ID	resid	ID tài nguyên

[Return Parameters]

ID tskid Task ID
hoặc Mã lỗi (Error codes)

[Error Codes]

E_NOMEM	Thiếu bộ nhớ (không thể định vị vùng nhớ cho khối điều khiển hay user stack)
E_LIMIT	Số lượng Task vượt quá giới hạn của hệ thống
E_RSATR	Lỗi thuộc tính
E_NOSPT	Chức năng không được cung cấp(khi TA_USERSTACK hay TA_TASKSPACE không được hỗ trợ)
E_PAR	Lỗi tham số
E_ID	ID tài nguyên (*resid*) không hợp lệ
E_NOCOP	Coprocessor được chỉ định không sử dụng được

[Description]

- Khởi tạo một Task và gán một số ID cho nó. Hàm này định vị một TCB (Task Control Block) để khởi tạo Task dựa theo các tham số *itskpri, task, stksz* …Trạng thái của Task sau khi khởi tạo là DORMANT.
- *itskpri* chỉ định độ ưu tiên khởi tạo tại thời điểm tạo Task. *Itskpri* có giá trị từ 1 đến 140, theo thứ tự độ ưu tiên giảm dần.
- *exinf* có thể được sử dụng tự do bởi người dùng để chèn thêm thông tin về Task, và được tham khảo bằng hàm *tk_ref_tsk*. Nếu thông tin của Task là một vùng lớn, ứng dụng phải xin cấp phát vùng nhớ riêng biệt và đặt địa chỉ vào exinf.
- *tskatr* cho biết các thuộc tính hệ thống trong các bit thấp của nó và thông tin hiện thực trong các bit cao.

tskatr := (TA_ASM | | TA_HLNG)
| [TA_SSTKSZ] | [TA_USERSTACK] | [TA_TASKSPACE] | [TA_RESID]
| (TA_RNG0 | | TA_RNG1 | | TA_RNG2 | | TA_RNG3)
| [TA_COP0] | [TA_COP1] | [TA_COP2] | [TA_COP3] | [TA_FPU]

TA_ASM	Cho biết chương trình được viết bằng hợp ngữ
TA_HLNG	Cho biết chương trình được viết bằng ngôn ngữ cấp cao
TA_SSTKSZ	Cho biết thuộc tính *sstksz* được sử dụng, nếu TA_SSTKSZ không được chỉ định thì *sstksz* sẽ bị bỏ qua và kích thước mặc dịnh sẽ được thiết đặt.
TA_USERSTACK	Khi thuộc tính này được chỉ định thì *sstkptr* sẽ hợp lệ. Trong trường hợp này OS không cung cấp stack người dùng nhưng người dùng phải xin cấp phát, và *sstksz* phải được thiết lập là 0. Nếu TA_USERSTACK không được chỉ định, *sstkptr* sẽ bị bỏ qua. Nếu TA_RNG0 được thiết lập thì TA_USERSTACK không thể được chỉ định.
TA_TASKSPACE	Khi thuộc tính này được chỉ định thì *uatb* và *lisd* sẽ hợp lệ và được thiết lập như là không gian của Task.Nếu TA_TASKSPACE không được chỉ định, *uatb* và *lisd* sẽ bị bỏ qua, không gian của Task sẽ không được định nghĩa. Trong suốt thời gian này, ta chỉ truy cập được không gian hệ thống. Cho dù TA_TASKSPACE được hay không được chỉ định, không gian Task vẫn có thể được thay đổi sau khi tạo Task
TA_RESID	Khi thuộc tính này được chỉ định thì *resid* sẽ hợp lệ, và chỉ ra nhóm resources nào mà Task sẽ phụ thuộc.
TA_RNG*n*	Cho biết Task chạy ở cấp bảo vệ cấp *n.*
TA_COP*n*	Cho biết có sử dụng Coprocessor thứ n(bao gồm floating point Coprocessor hay DSP).
TA_FPU	Cho biết có sử dụng FPU.

```
#define TA_ASM      0x00000000 /* Assembly program */
#define TA_HLNG     0x00000001 /* High-level language program */
#define TA_SSTKSZ   0x00000002 /* System stack size */
```

```
#define TA_USERSTACK    0x00000004 /* User stack pointer */
#define TA_TASKSPACE    0x00000008 /* Task space pointer */
#define TA_RESID        0x00000010 /* Task resource group */
#define TA_RNG0         0x00000000 /* Run at protection level 0 */
#define TA_RNG1         0x00000100 /* Run at protection level 1 */
#define TA_RNG2         0x00000200 /* Run at protection level 2 */
#define TA_RNG3         0x00000300 /* Run at protection level 3 */
#define TA_COP0         0x00001000 /* Use ID=0 Coprocessor */
#define TA_COP1         0x00002000 /* Use ID=1 Coprocessor */
#define TA_COP2         0x00004000 /* Use ID=2 Coprocessor */
#define TA_COP3         0x00008000 /* Use ID=3 Coprocessor */
```

- Khi TA_HLNG được chỉ định, khi Task bắt đầu nó sẽ nhảy đến địa chỉ của Task không trực tiếp mà thông qua sự hỗ trợ của môi trường ngôn ngữ cấp cao. Hàm xử lý Task có dạng như sau :

```
void Task( INT stacd, VP exinf )
{
    /*
    (processing)
    */
    tk_ext_tsk(); hoặc tk_exd_tsk(); /* Exit Task */
}
```

Những thông số khởi động bao gồm cả mã khởi động Task *stacd* trong hàm *tk_sta_tsk*, và thông tin thêm về Task *exinf.*

Một Task không thể kết thúc bởi một sự trả về đơn giản mà ta phải gọi các hàm khác tùy thuộc vào hiện thực.

Định dạng của Task khi TA_ASM được chỉ định phụ thuộc vào sự hiện thực và các thông số cũng phải dược truyền như là thông số khởi động.

- Mỗi Task chạy ở một cấp bảo vệ được chỉ định trong thuộc tính TA_RNG*n.* Khi một hàm system call hay hàm dịch vụ mở rộng được gọi thì cấp bảo vệ sẽ về 0, và trở lại như cũ khi các hàm được thực hiện xong.
- Mỗi Task có hai vùng stack : stack hệ thống và stack người dùng, Stack người dùng được sử dụng ở cấp bảo vệ được chỉ định trong TA_RNG*n*, trong khi stack hệ thống được sử dụng ở cấp bảo vệ 0. Khi TA_RNG*0* được chỉ định thì cả hai stack sẽ được nhập lại làm một stack.

2. tk_del_tsk : Xóa task

[C Language Interface]

ER ercd = tk_del_tsk (ID tskid) ;

[Parameters]

ID	tskid	Task ID

[Return Parameters]

ER ercd Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*tskid* không hợp lệ hay không được sử dụng
E_NOEXS	Task được chỉ định trong *tskid* không tồn tại
E_OBJ	Trạng thái của Task không phải là DORMANT

[Description]

- Xóa bỏ Task có ID là *tskid* .
- Hàm system call này thay đổi trạng thái của Task từ DORMANT thành NON-EXSTENT (không còn tồn tại trong hệ thống), giải phóng TCB và vùng stack. Task ID cũng được giải phóng. Khi trạng thái Task không là DORMANT mà gọi hàm này thì mã lỗi E_OBJ sẽ được trả về.
- Hàm này không thể xóa bỏ Task gọi nó, bởi vì Task gọi nó không nằm trong trạng thái DORMANT. Ta sẽ dùng hàm *tk_exd_tsk* để xóa bỏ chính nó.

3. tk_sta_tsk : Kích khởi sự thực thi của một task

[C Language Interface]

ER ercd = tk_sta_tsk (ID tskid, INT stacd) ;

[Parameters]

ID tskid Task ID

INT stacd Mã khởi động Task

[Return Parameters]

ER ercd Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*tskid* không hợp lệ hay không được sử dụng
E_NOEXS	Task được chỉ định trong *tskid* không tồn tại
E_OBJ	Trạng thái của Task không phải là DORMANT

[Description]

- Khởi động Task có ID là *tskid* .
- Hàm system call này thay đổi trạng thái của Task từ DORMANT thành READY. Những thông số được truyền cho Task khi khởi động được thiết lập trong *stacd.* Những thông số này có thể được tham khảo từ Task được khởi động , có thể sử dụng đặc điểm này cho việc truyền nhận thông điệp đơn giản.
- Độ ưu tiên của Task được chỉ định trong thuộc tính *itskpri* khi khởi tao Task đó.

- Các yêu cầu khởi động Task được gọi bởi hàm này không được xếp hàng. Nếu Task đích không nằm trong trạng thái DORMANT thì mã lỗi E_OBJ sẽ được trả về cho Task goi hàm này.

4. tk_ext_tsk : Thoát khỏi sự thực thi của task

[C Language Interface]

void tk_ext_tsk ( ) ;

[Description]

- Hàm system call này làm nhiệm vụ yêu cầu hệ thống làm thoát Task gọi nó và chuyển trạng thái của Task đó thành DORMANT.
- Khi một Task kết thúc bởi hàm *tk_ext_tsk ( )* , những tài nguyên do Task đang nắm giữ sẽ không tự động được giải phóng, mà người dùng phải tự giải phóng chúng.
- Khi trạng thái của Task chuyển xuống DORMANT thì độ ưu tiên và những thông tin khác trong TCB sẽ được tái khởi tạo.

5. tk_exd_tsk : Thóat khỏi sự thực thi của task và đồng thời xóa task

[C Language Interface]

void tk_exd_tsk ( ) ;

[Description]

- Hàm system call này làm nhiệm vụ yêu cầu hệ thống kết thúc thực thi chính Task gọi nó xóa bỏ Task, khi đó trang thái của Task đó là NON-EXISTENT (không còn tồn tại trên hệ thống).
- Khi một Task kết thúc bởi hàm *tk_exd_tsk ( )* , những tài nguyên do Task đang nắm giữ sẽ không tự động được giải phóng, mà người dùng phải tự giải phóng chúng.

6. tk_ter_tsk : Kết thúc sự thực thi của một task

[C Language Interface]

ER ercd = tk_ter_tsk (ID tskid) ;

[Parameters]

ID	tskid	Task ID

[Return Parameters]

ER	ercd	Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*tskid* không hợp lệ hay không được sử dụng
E_NOEXS	Task được chỉ định trong *tskid* không tồn tại
E_OBJ	Trạng thái của Task không phải là DORMANT

[Description]

- Kết thúc sự thực thi của Task có ID là *tskid* .
- Hàm system call này thay đổi trạng thái của Task có ID là *tskid* thành DORMANT.
- Ngay cả khi Task đích đang ở trạng thái đợi (bao gồm cả trạng thái SUSPEND) hay nằm trong các hàng đợi (semaphore, message …), thì nó được giải phóng khỏi trạng thái đợi và bị kết thúc.
- Hàm này không thể kết thúc chính nó. Nếu Task gọi hàm được chỉ định là Task đích thì mã lỗi E_OBJ sẽ được trả về cho Task goi hàm này.
- Mối quan hệ giữa trạng thái của Task đích và kết quả trở về của hàm *tk_ter_tsk* được tóm tắt trong bảng sau

Trạng thái của Task đích	Mã lỗi trả về của *tk_ter_tsk*	Kết quả thực thi của hàm
RUN hoặc READY (ngoại trừ Task gọi hàm)	E_OK	Kết thúc thực thi Task đích
RUN (Task gọi hàm)	E_OBJ	Không thực hiện
WAIT	E_OK	Kết thúc thực thi Task đích
DORMANT	E_OBJ	Không thực hiện
NON-EXISTENT	E_NOEXS	Không thực hiện

- Khi một Task kết thúc bởi hàm *tk_ext_tsk ( )* , những tài nguyên do Task đang nắm giữ sẽ không tự động được giải phóng, mà người dùng phải tự giải phóng chúng.
- Khi trạng thái của Task chuyển xuống DORMANT thì độ ưu tiên và những thông tin khác trong TCB sẽ được tái khởi tạo.

7. tk_chg_pri : Thay đổi độ ưu tiên của task

[C Language Interface]

ER ercd = tk_chg_pri (ID tskid, PRI tskpri) ;

[Parameters]

ID tskid Task ID

PRI tskpri Độ ưu tiên mới của Task

[Return Parameters]

ER ercd Mã lỗi (Error Codes)

[Error Codes]

E_OK Kết thúc bình thường

E_ID *tskid* không hợp lệ hay không được sử dụng

E_NOEXS Task được chỉ định trong *tskid* không tồn tại

E_PAR Lỗi tham số (*tskpri* không hợp lệ hay không được dùng)

E_ILUSE *tskpri* vượt quá giới hạn của hệ thống

[Description]

- Thay đổi độ ưu tiên cơ bản có Task ID la *tskid* đến giá trị được chỉ định trong *tskpri,* đồng thời độ ưu tiên hiện tại của Task cũng thay đổi theo.
- Khi TSK_SELF (= 0) được gán cho *tskid* thì Task đích cũng chính là Task gọi hàm. Tuy nhiên , khi *tskid* = TSK_SELF được chỉ định trong hàm system call được gọi từ một Task-independent portion, thì mã lỗi E_ID sẽ được trả về.
- Khi TPRI_INI (= 0) được gán cho *tskpri*, thì độ ưu tiên cơ bản của Task đích bị thay dổi về giá trị khởi tạo khi Task được bắt đầu thực thi (*itskpri*).
- Độ ưu tiên bị thay đổi bởi hàm này vẫn có giá trị cho tới khi Task bị kết thúc (terminated). Khi một Task trở lại trạng thái DORMANT, độ ưu tiên của Task trước khi thoát bị loại bỏ và khi đó nó sẽ được thiết lập lại giá trị khởi tạo khi Task dó được bắt đầu thực thi (*itskpri*). Tuy nhiên sự thay đổi độ ưu tiên trong khi Task đang ở trong trạng thái DORMANT sẽ có hợp lệ khi nó được khởi động vào lần kế tiếp.

8. tk_chg_slt : Thay đổi slicetime của task

[C Language Interface]

ER ercd = tk_chg_slt (ID tskid, RELTIM slicetime) ;

[Parameters]

ID tskid Task ID

RELTIM slicetime Khe thời gian (Time slice)

[Return Parameters]

ER ercd Mã lỗi (Error Codes)

[Error Codes]

E_OK Kết thúc bình thường

E_ID *tskid* không hợp lệ hay không được sử dụng

E_NOEXS Task được chỉ định trong *tskid* không tồn tại

E_PAR Lỗi tham số (*slicetime* không hợp lệ hay không được dùng)

[Description]

- Thay đổi khe thời gian của một Task có Task ID la *tskid* bằng giá trị được chỉ định trong *slicetime,* chức năng khe thời gian được sử dụng trong việc định thời các Task theo Round Robin. Khi một Task thực thi liên tục trong một khoảng thời gian là *slicetime* hay dài hơn , độ ưu tiên (thứ tự thực thi trước sau) của nó chuyển về thấp nhất giữa các Task có cùng độ ưu tiên (Priority), và tự động nhường sự thực thi cho Task kế tiếp.
- Thiết lập *slicetime = 0* cho biết thời gian không giới hạn, và Task sẽ không tự động nhường sự thực thi của nó cho Task khác.
- Khi một Task được khởi tạo thì *slicetime* của nó được thiết lập mặc định là 0.

- Khi TSK_SELF (= 0) được gán cho *tskid* thì Task đích cũng chính là Task gọi hàm. Tuy nhiên , khi *tskid* = TSK_SELF được chỉ định trong hàm system call được gọi từ một Task-independent portion, thì mã lỗi E_ID sẽ được trả về.
- Khe thời gian bị thay đổi bởi hàm này vẫn có giá trị cho tới khi Task bị kết thúc (terminated). Khi một Task trở lại trạng thái DORMANT, khe thời gian của Task trước khi thoát bị loại bỏ và khi đó nó sẽ được thiết lập lại giá trị khởi tạo khi Task đó được bắt đầu thực thi (*slicetime = 0*). Tuy nhiên sự thay đổi độ ưu tiên trong khi Task đang ở trong trạng thái DORMANT sẽ có hợp lệ khi nó được khởi động vào lần kế tiếp.
- Một Task bị tước đoạt sự thực thi bởi một Task có độ ưu tiên cao hơn thì thời gian thực thi của nó sẽ bị bỏ qua và chỉ được đếm lại từ đầu trong lần thực thi kế tiếp.
- Nếu Task đích là Task duy nhất đang thực thi tại độ ưu tiên của nó thì khe thời gian sẽ không có ý nghĩa, và Task đó được thực thi liên tục.
- Nếu trong nhiều Task có cùng độ ưu tiên có một Task mà *slicetime* = 0, thì Task đó sẽ được thực thi ngay khi có thể và định thời theo Round Robin cũng kết thúc.

9. tk_inf_tsk : Lấy thông tin về task

[C Language Interface]

ER ercd = tk_inf_tsk (ID tskid, T_ITSK * pk_itsk, BOOL clr) ;

[Parameters]

ID	tskid	Task ID
T_ITSK*	pk_itsk	Địa chỉ của gói thông tin về Task để trả về
BOOL	clr	Xóa thông tin về Task

[Return Parameters]

ER	ercd	Mã lỗi

Chi tiết của pk_itsk :

RELTIM	stime	Thời gian chạy tích lũy ở cấp hệ thống (ms)
RELTIM	utime	Thời gian chạy tích lũy ở cấp người dùng (ms)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*tskid* không hợp lệ hay không được sử dụng
E_NOEXS	Task được chỉ định trong *tskid* không tồn tại
E_PAR	Lỗi tham số (địa chỉ của gói thông tin trả về không dùng được)

[Description]

- Thu thập các thông tin thống kê của một Task có Task ID là *tskid.*
- Nếu *clr* = TRUE (≠ 0) , thì thông tin tích lũy thì được khởi tạo lại giá trị ban đầu sau khi thu thập thông tin.

- Khi TSK_SELF (= 0) được gán cho *tskid* thì Task đích cũng chính là Task gọi hàm. Tuy nhiên, khi *tskid* = TSK_SELF được chỉ định trong hàm system call được gọi từ một Task-independent portion, thì mã lỗi E_ID sẽ được trả về.
- Thời gian chạy ở cấp hệ thống là thời gian mà Task chạy ở cấp bảo vệ 0 (TA_RNG0), còn thời gian chạy ở các cấp bảo vệ khác được tính cho thời gian chạy ở cấp người dùng.Do vậy khi một Task được tạo và chạy ở cấp bảo vệ 0 (TA_RNG0) thì thời gian chạy chủ yếu là thời gian ở cấp hệ thống.

10. tk_ref_tsk : Tham khảo đến trạng thái của task

[C Language Interface]

ER ercd = tk_ref_tsk (ID tskid, T_RTSK * pk_rtsk) ;

[Parameters]

ID	tskid	Task ID
T_RTSK*	k_rtsk	Địa chỉ của gói thông tin về Task để trả về

[Return Parameters]

ER	ercd	Mã lỗi
Chi tiết	pk_rtsk :	
VP	exinf	thông tin thêm về Task
PRI	tskpri	Độ ưu tiên hiện tại của Task
PRI	tskbpri	Độ ưu tiên cơ bản
UINT	tskstat	Trạng thái của Task
UINT	tskwait	Thừa số đợi
ID	wid	ID của đối tượng đang đợi
INT	wupcnt	Số lượng yêu cầu đánh thức
INT	suscnt	Số lượng yêu cầu suspend
RELTIM	slicetime	Thời gian thực thi liên tục tối đa cho phép (ms)
UINT	waitmask	Thừa số hủy đợi
UINT	texmask	Cho phép những ngoại lệ xuất hiện
UINT	tskevent	Sự kiện của Task

(Những tham số khác tùy theo hiện thực có thêm vào sau)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*tskid* không hợp lệ hay không được sử dụng
E_NOEXS	Task được chỉ định trong *tskid* không tồn tại
E_PAR	Lỗi tham số (địa chỉ của gói thông tin trả về không dùng được)

[Description]

- Thu thập trạng thái của một Task có Task ID là *tskid.*
- Thuộc tính *tskstat* có thể có những giá trị sau đây :

tskstat:

TTS_RUN 0x0001 RUN
TTS_ RDY 0x0002 READY
TTS_ WAI 0x0004 WAIT
TTS_ SUS 0x0008 SUSPEND
TTS_ WAS 0x000c WAIT-SUSPEND
TTS_ DMT 0x0010 DORMANT
TTS_ NODISWAI 0x0080 Trạng thái WAIT bị vô hiệu

- Khi hàm *tk_ref_tsk* được thực thi trong một *interrupted task* từ một chương trình quản ký interrupt thì RUN sẽ được trả về cho *tskstat.*
- Khi *tskstat* là TTS_WAI (bao gồm cả TTS_WAS), giá trị của *tskwait* và *wid* được mô tả như sau:

tskwait	Giá trị	Mô tả	*wid*
TTW_SLP	0x00000001	Đợi do *tk_slp_tsk*	0
TTW_DLY	0x00000002	Đợi do *tk_dly_tsk*	0
TTW_SEM	0x00000004	Đợi do *tk_wai_sem*	semid
TTW_FLG	0x00000008	Đợi do *tk_wai_flg*	flgid
TTW_MBX	0x00000040	Đợi do *tk_rcv_mbx*	mbxid
TTW_MTX	0x00000080	Đợi do *tk_loc_mtx*	mtxid
TTW_SMBF	0x00000100	Đợi do *tk_snd_mbf*	mbfid
TTW_RMBF	0x00000200	Đợi do *tk_rcv_mbf*	mbfid
TTW_CAL	0x00000400	Đợi do gọi rendezvous	porid
TTW_ACP	0x00000800	Đợi do chấp nhận rendezvous	phorid
TTW_RDV	0x00001000	Đợi do hoàn thành rendezvous	0
(TTW_CAL|TTW_RDV)	0x00001400	Đợi do gọi rendezvous hay do hoàn thành rendezvous	0
TTW_MPF	0x00002000	Đợi do *tk_get_mpf*	mpfid
TTW_MPL	0x00004000	Đợi do *tk_get_mpl*	mplid
TTW_EV1	0x00008000	Đợi do task event #1	0
TTW_EV2	0x00010000	Đợi do task event #2	0
TTW_EV3	0x00020000	Đợi do task event #3	0
TTW_EV4	0x00040000	Đợi do task event #4	0
TTW_EV5	0x00080000	Đợi do task event #5	0
TTW_EV6	0x00100000	Đợi do task event #6	0
TTW_EV7	0x00200000	Đợi do task event #7	0
TTW_EV8	0x00400000	Đợi do task event #8	0

- Khi *tskstat* không là TTS_WAI (bao gồm cả TTS_WAS), giá trị của *tskwait* và *wid* đều bằng 0.
- Nếu trạng thái của Task là DORMANT thì *wupcnt* = 0*, suscnt* = 0*, and tskevent* =0
- Khi TSK_SELF (= 0) được gán cho *tskid* thì Task đích cũng chính là Task gọi hàm. Tuy nhiên , khi *tskid* = TSK_SELF được chỉ định trong hàm system call được gọi từ một Task-independent portion, thì mã lỗi E_ID sẽ được trả về.
- Khi Task đích không tồn tại thì mã lỗi E_NOEXS được trả về.

2.3 Cấu trúc của một chương trình ứng dụng

2.3.1 Makefile

Khi viết một chương trình nhỏ, thường lập trình viên sẽ biên dịch lại toàn bộ mã nguồn sau khi chỉnh sửa đôi chút trên file gốc. Tuy nhiên với những chương trình lớn, một số vấn đề sẽ phát sinh khi sử dụng cách biên dịch lại toàn bộ như trên. Thời gian để thực hiện công đoạn sửa mã – biên dịch lại- kiểm tra tăng lên rất lớn. Ngay cả những lập trình viên kiên nhẫn nhất vẫn cũng cố gắng tránh biên dịch lại những file không có chỉnh sửa gì trên đó cả. Chúng ta chỉ cần biên dịch lại những file có bổ sung hay sửa đổi là đủ.

Để giải quyết vấn đề trên thì GNU có cung cấp cho chúng ta một công cụ đó là make, chương trình này đảm bảo chỉ những file có chỉnh sửa mới được biên dịch lại mà thôi. Trình make không chỉ dùng phục vụ cho việc biên dịch chương trình mà còn sử dụng để tạo ra một file kết xuất từ nhiều file nguồn khác nhau. make còn chứa đựng các macro để gọi các trình xử lý văn bản hoặc sử dụng cho mục đích cài đặt chương trình.

Tuy nhiên, ở đây chúng ta sẽ sử dụng lệnh gmake để biên dịch cho chương trình của chúng ta (không có sự khác biệt giữa gmake và make , tuy nhiên phụ thuộc vào phiên bản gnu mà hệ thống đang có mà chúng ta dùng gmake hay make).

Mặc dù trình như bạn sẽ thấy gmake có chứa rất nhiều lệnh điều khiển nội tại nhưng bản thân nó thì không biết làm cách nào để xây dựng ứng dụng hộ bạn. Bạn phải tạo ra một file chứa các thông tin hướng dẫn trình gmake cách ứng dụng của bạn sẽ được xây dựng và biên dịch. Trên UNIX và Linux, file này thường mang tên là Makefile.

Tập tin Makefile thường nằm cùng thư mục gốc với nơi chứa mã nguồn cần biên dịch của dự án. Bạn có thể có nhiều tập tin Makefile khác nhau. Thực tế nếu dự án của bạn lớn thì bạn có thể tạo các file Makefile với nhiều các biên dịch khác nhau. Sự kết hợp giữa trình gmake và tập tin Makefile là công cụ cực kỳ mạnh để bạn quản lý dự án. Không những quản lý mã nguồn và biên dịch dự án , gmake còn dùng để sinh tài liệu hướng dẫn, cài đặt hoặc tháo bỏ các chương trình khỏi hệ điều hành Linux.

Trình gmake bản thân có rất nhiều tùy chọn nhưng có 3 tùy chọn thông dụng nhất đó là :

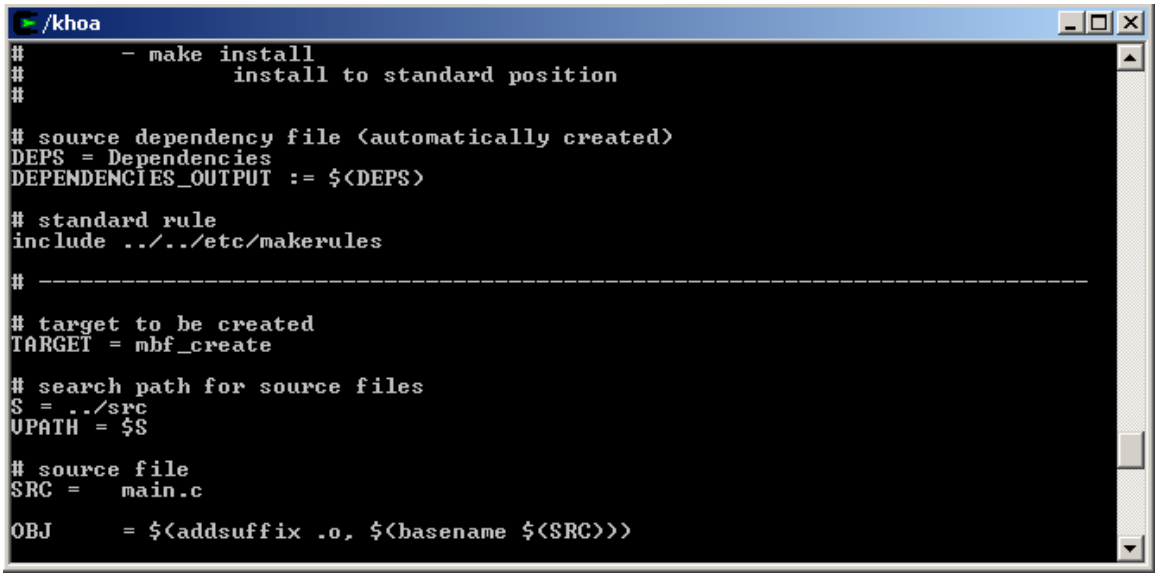
-k tùy chọn này yêu cầu make ‘cứ tiếp tục’ khi một lỗi nào đó phát sinh thay vì phải dừng quá trình xử lý ở ngay lỗi đầu tiên.

-n tùy chọn này yêu cầu gmake in ra các thao tác mà nó sẽ thực hiện nhưng không thực sự thực thi ra kết quả.

-f <file name> tùy chọn này cho phép bạn chỉ định tên tập tin Makefile cần diễn dịch bởi trình gmake. Thông thường nếu không có tùy chọn –f trình gmake sẽ tìm tập tin mang tên makefile hay Makefile ngay trong thư mục hiện hành nơi bạn gõ lệnh make. Ví dụ

$ gmake tương đương với $gmake Makefile

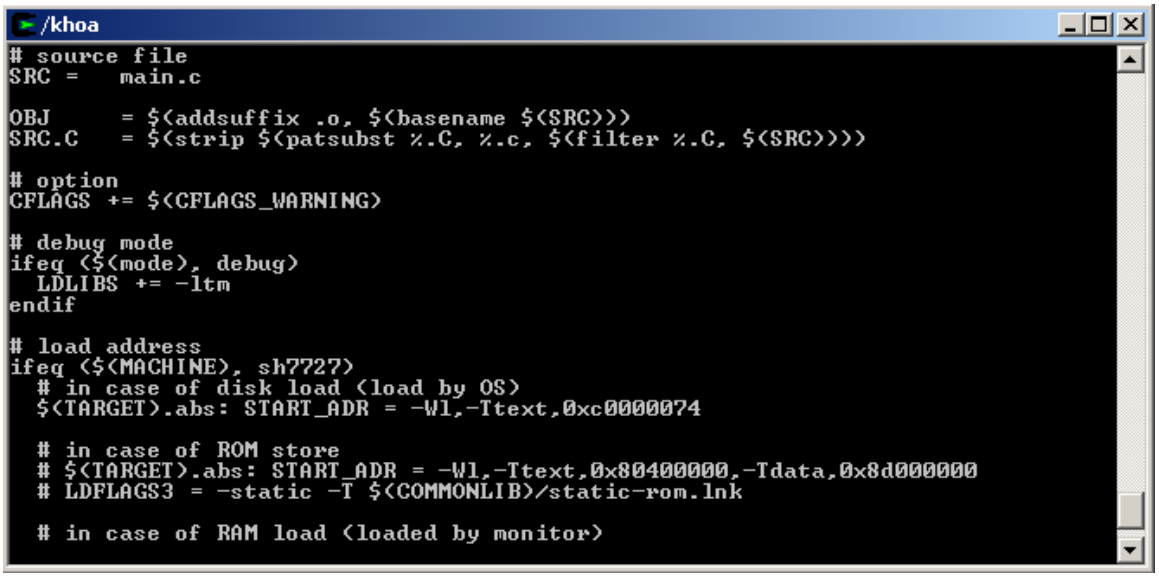
Trên T-engine , để xây dựng một chương trình ứng dụng cụ thể ta không cần phải xây dựng một file makefile mới mà chỉ cần chỉnh sửa nội dung của file makefile có sẵn là được rồi. Sau đây là những chỉ dẫn cần phải thay đổi trong tập tin makefile.



Tên file thực thi sẽ được đặt trong TARGET = <tên file thực thi>

Những file source của chương trình sẽ được đặt trong

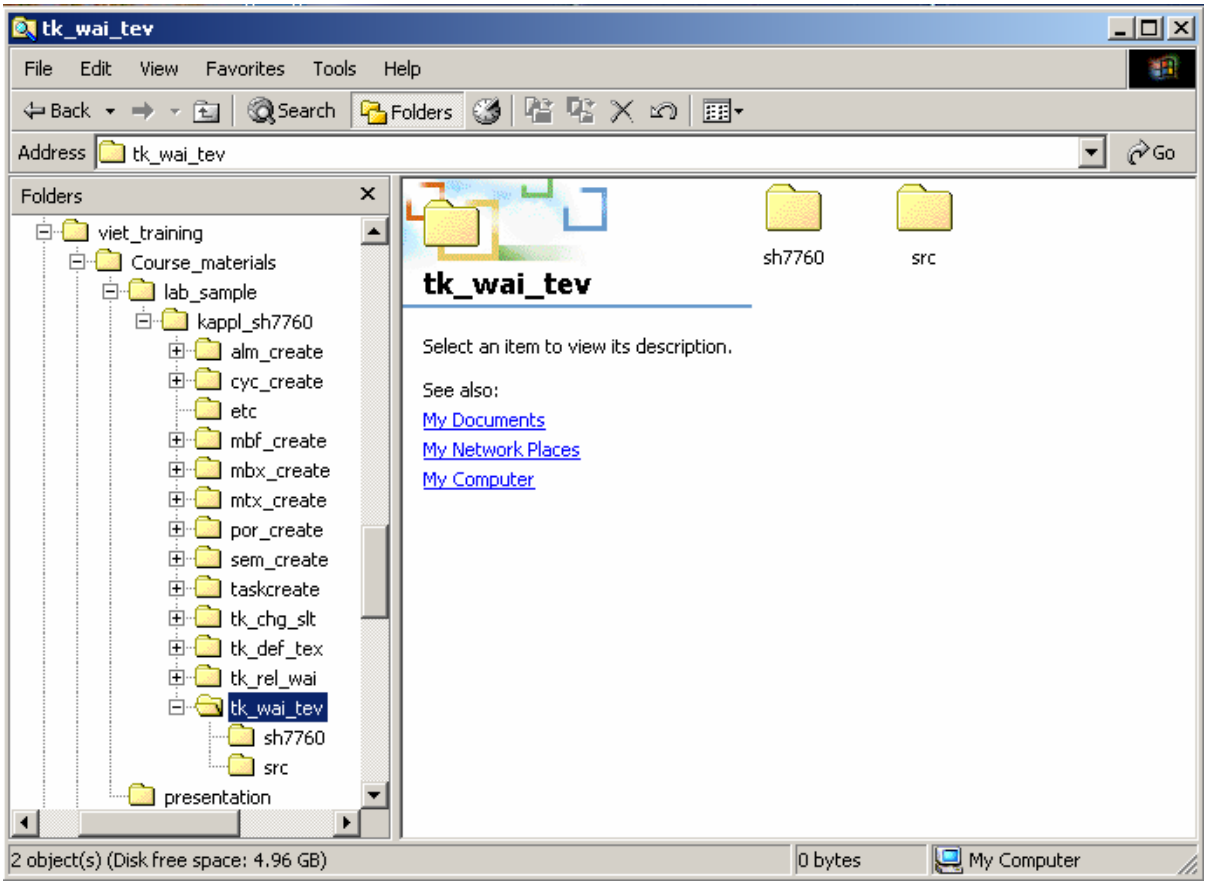
SRC = <file1.c> <file2.c> ….. (chú ý : giữa các file chỉ là khoảng trắng và chỉ cần chỉ ra các file.c , không cần các file.h)



Nếu như muốn thêm thư viện nào khác thì thêm ở LDLIBS = <tên thư viện> …

2.3.2 Cấu trúc của một chương trình ứng dụng trên T-Engine

Xây dựng một chương trình trên T-Engine ta cần phải xây dựng 2 thư mục chính đó là **sh7760** và thư mục **src**. Trong đó thư mục **src** dùng để chứa các file source như những file.h ,file.c và Makefile còn thư mục **sh7760** dùng để lưu giữ các file kết xuất mà chúng ta có được trong quá trình biên dịch.



Một điểm quan trọng ở đây đó là trong các file source thì phải có duy nhất một file chứa hàm main() và đó cũng là chương trình thực thi chính của ứng dụng. Và từ chương trình này , chúng ta có thể kích khởi sự thực thi của các task khác.

3. THỰC HÀNH

Đọc hiểu và thực thi quá trình biên dịch của chương trình sau :

//----------main.c----------

```
#include <basic.h>
#include <tk/tkernel.h>
#include <stdio.h>

IMPORT void task1(INT, VP);
ID taskid1 = -1;

/*======================================================================
main
======================================================================*/
EXPORT ER main( INT ac, UB *av[] )
{
   T_CTSK ctsk;
   ER ercd = 0;
   ID main_id;
   printf("main: (ac=%d)\n", ac);
   if (ac < 0)
   {
```

```
     if (taskid1 >= 0)
     {
       ercd = tk_ter_tsk(taskid1);
      if (ercd < 0)
        printf("tk_ter_tsk err = %x\n", ercd);
       ercd = tk_del_tsk(taskid1);
      if (ercd < 0)
        printf("tk_del_tsk err = %x\n", ercd);
     }
     goto ext;
   }
   ctsk.exinf = (VP)0x74736574; ///thong tin mo rong cua task
   ctsk.tskatr = TA_HLNG | TA_RNG1; ///thiet lap thuoc tinh cho task
   ctsk.task = task1; // thiet lap dia chi bat dau cua task
   ctsk.itskpri = 80; // do uu tien cua task
   ctsk.stksz = 1024 * 4; //kich thuoc can co' cua task de task thuc thi
   taskid1 = tk_cre_tsk(&ctsk); /*tao task voi cac thuoc tinh da~ duoc
thiet lap truoc*/
   printf("tk_cre_tsk: (taskid = %d)\n", taskid1);
   if (taskid1 < E_OK)
     goto ext;

   main_id = tk_get_tid();
   printf("Id cua man()   = %d\n", main_id);
   tk_sta_tsk(taskid1, 0);
   // now change priority of task1
   ercd = tk_chg_pri(taskid1,100);
   if (ercd < E_OK)
   {
      printf("Change priority has a problem \n");
      goto ext;
   }
   /*
   you must enter your code here to print the priority of task1
   */
   /*end*/
 ext:
   printf("main ended\n");
   return 0;
}

/*****************************************************************
What does the task1 do???
 *****************************************************************/
IMPORT void task1(INT stacd, VP exinf)
{
  int cnt = 0;
  ID task_id = 0;
  T_RTSK pkrtsk; //con tro chua goi du lieu ve task duoc tra ve

  task_id = tk_get_tid();
  printf("Id cua task1 la` = %d\n", task_id);
  if(ercd > 0)
  {
   tk_ref_tsk(ercd , &pkrtsk); /// tham khao trang thai cua task
   printf("Do uu tien hien tai cua task1 la %d \n", pkrtsk.tskpri);
   }
  /*
you must enter your code here to show the slicetime of task1 and change it
```

```
*/
      printf("exit and del task now\n");
   tk_exd_tsk();
  }
```

Thêm code vào những chỗ được chỉ định bên trên (2 chỗ) để thực hiện những yêu cầu được đưa ra.

BÀI THỰC HÀNH SỐ 5

ĐỒNG BỘ TASK-DEPENDENT

1. MỤC TIÊU

- Nắm bắt một cách cơ bản việc đồng bộ giữa các task độc lập thông qua việc điều khiển trạng thái của các task.

2. LÝ THUYẾT

Các hàm đồng bộ Task-Dependent nhằm đạt tới sự đồng bộ giữa các Task bằng các thao tác trực tiếp lên trạng thái của các Task. Những hàm được cung cấp để đưa Task về trạng thái nghỉ (sleep)hay đánh thức (wakeup) một Task; hủy bỏ các yêu cầu đánh thức Task; giải phóng Task khỏi trạng thái WAIT một cách bắt buộc; chuyển trạng thái của Task về trạng thái SUSPEND; trì hoãn sự thực thi của một Task; hoặc làm vô hiệu trạng thái WAIT của một Task.

Các yêu cầu đánh thức cho một Task thì được xếp trong hàng đợi (queue). Điều này có nghĩa là khi nó cố gắng đánh thức một Task mà Task này không ở trạng thái nghỉ thì các yêu cầu đánh thức này được ghi nhớ, và khi Task đó đi vào trang thái nghỉ ở lần kế tiếp (đang đợi để thức dậy) thì nó sẽ không chuyển về trạng thái nghỉ đó. Hàng đợi các yêu cầu đánh thức được nhận ra bởi mỗi Task có một số đếm để chứa số các yêu cầu đánh thức được xếp hàng, khi Task được khởi động số đếm này được xoá về 0. Và các yêu cầu làm treo (suspend) một Task cũng được xếp hàng tương tự như WAIT.

❖ **Các hàm đồng bộ Task-Dependent:**

1. tk_slp_tsk : Sleep Task

[C Language Interface]

```
ER ercd = tk_slp_tsk ( TMO tmout ) ;
```

[Parameters]

TMO	tmout	Thời gian Timeout được chỉ định

[Return Parameters]

ER	ercd	Mã lỗi (Error codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_PAR	Lỗi tham số (*tmout* ≤ (-2))
E_RLWAI	Trạng thái WAIT của Task được giải phóng (do Task nhận được tk_rel_wai khi đang ở trạng thái WAIT)
E_DISWAI	Trạng thái WAIT của Task được giải phóng bởi trạng thái WAIT của Task bị vô hiệu
E_TMOUT	Thăm dò bị lỗi hay timeout

E_CTX	Lỗi ngữ cảnh (hàm được gọi từ một *Task độc lập với ngữ cảnh* hay *dispatch* bị vô hiệu)

[Description]

- Thay đổi trạng thái của Task gọi hàm từ trạng thái RUN thành trạng thái nghỉ (WAIT, đợi cho tới khi được đánh thức bởi hàm *tk_wup_tsk* từ một Task khác).
- Nếu một Task khác gọi hàm *tk_wup_tsk* đánh thức Task gọi hàm *tk_slp_tsk* trước thời gian timeout (*tmout*), thì hàm *tk_slp_tsk* sẽ kết thúc bình thường. Nếu thời gian timeout đến trước khi hàm *tk_wup_tsk* được gọi thì mã lỗi E_TMOUT sẽ được trả về.
- Nếu chỉ định *tmout* = TMO_FEVR = (-1) thì Task gọi hàm sẽ đợi vô tận cho tới khi được đánh thức bởi hàm *tk_wup_tsk* từ một Task khác.

2. tk_wup_tsk : WakeupTask

[C Language Interface]

ER ercd = tk_wup_tsk (ID tskid) ;

[Parameters]

ID	tskid	Task ID

[Return Parameters]

ER	ercd	Mã lỗi (Error codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*tskid* không hợp lệ hay không được sử dụng
E_PAR	Lỗi tham số (*tmout* ≤ (-2))
E_NOEXS	Task được chỉ định trong *tskid* không tồn tại
E_OBJ	Trạng thái của đối tượng không hợp lệ (đánh thức chính Task gọi hàm hay đánh thức một Task đang ở trạng thái DORMANT)
E_QOVR	Hàng đợi các yêu cầu đánh thức bị tràn (quá nhiều yêu cầu đánh thức được xếp hàng trong *wupcnt*)

[Description]

- Nếu Task được chỉ định trong *tskid* được đặt trong trạng thái WAIT bởi hàm *tk_slp_tsk*, thì hàm *tk_wup_tsk* sẽ giải phóng trạng thái WAIT cho Task đó.
- Hàm system call này không thể được gọi cho chính Task gọi nó, nếu không thì lỗi E_OBJ sẽ được trả về.
- Nếu Task đích không gọi hàm *tk_slp_tsk* và không ở trạng thái WAIT thì các yêu cầu đánh thức bởi hàm *tk_wup_tsk* sẽ được lưu nhớ trong hàng đợi thông qua một biến đếm (*wupcnt)* các yêu cầu đánh thức mà chưa được thực hiện. Mỗi Task đều lưu trữ *wupcnt* trong TCB của nó, và giá trị khởi tạo (khi hàm *tk_sta_tsk* được thực thi) cho *wupcnt* là 0. Khi hàm *tk_wup_tsk* yêu cầu đánh thức một Task mà Task đó không ở trong trạng thái WAIT thì *wupcnt* sẽ tăng lên 1,ngược lại mỗi khi hàm *tk_slp_tsk* được thự thi thì *wupcnt*

sẽ giảm 1 và Task đó sẽ không đi vào trạng thái WAIT. Khi hàm *tk_slp_tsk* được thực thi trong khi *wupcnt* của Task đó bằng 0, thì *wupcnt* sẽ không thay đổi và Task đó sẽ đi vào trạng thái WAIT.

- Khi việc gọi hám *tk_wup_tsk* làm cho *wupcnt* vượt quá giới hạn cho phép, lỗi E_QOVR sẽ được trả về.

3. tk_can_wup: Hủy bỏ WakeupTask

[C Language Interface]

INT wupcnt = tk_can_wup (ID tskid) ;

[Parameters]

ID	tskid	Task ID

[Return Parameters]

INT	wupcnt	Số lượng các yêu cầu đánh thức đang đợi thực thi
hoặc		Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*tskid* không hợp lệ hay không được sử dụng
E_PAR	Lỗi tham số (*tmout* ≤ (-2))
E_NOEXS	Task được chỉ định trong *tskid* không tồn tại
E_OBJ	Trạng thái của đối tượng không hợp lệ (đánh thức chính Task gọi hàm hay đánh thức một Task đang ở trạng thái DORMANT)

[Description]

- Hàm system call này trả về giá trị *wupcnt* của Task được chỉ định trong *tskid*, đồng thời hủy bỏ tất cả các yêu cầu đánh thức Task đó và xóa giá trị *wupcnt* của nó về 0.
- Khi TSK_SELF (=0) được gán cho *tskid* thì Task đích cũng chính là Task gọi hàm.Tuy nhiên, khi *tskid* = TSK_SELF được chỉ định trong hàm system call được gọi từ một Task-independent portion, thì mã lỗi E_ID sẽ được trả về.

4. tk_rel_wai: Realease Wait

[C Language Interface]

ER ercd = tk_rel_wai (ID tskid) ;

[Parameters]

ID	tskid	Task ID

[Return Parameters]

ER	ercd	Mã lỗi (Error codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*tskid* không hợp lệ hay không được sử dụng
E_NOEXS	Task được chỉ định trong *tskid* không tồn tại
E_OBJ	Trạng thái của đối tượng không hợp lệ (gọi cho một Task khôngở trong trạng thái WAIT hay gọi cho chính Task gọi hàm hay cho một Task đang ở trạng thái DORMANT)

[Description]

- Nếu Task được chỉ định trong *tskid* đang ở trong một vài loại của trạng thái đợi (không bao gồm trạng thái SUSPEND) thì hàm *tk_rel_wai* sẽ giải phóng trạng thái đợi cho Task đó.
- Hàm system call này trả về mã lỗi E_RLWAI cho Task được giải phóng trạng thái WAIT.
- Các yêu cầu giải phóng trạng thái đợi bởi hàm *tk_rel_wai* thì không được ghi nhớ như hàm *tk_wup_tsk*.
- Hàm system call *tk_rel_wai* không giải phóng trạng thái SUSPEND. Nếu nó được gọi cho Task đanh ở trạng thái WAIT-SUSPEND, Task đó sẽ vào trạng thái SUSPEND. Nếu cần thiết giải phóng trạng thái SUSPEND, thì hàm *tk_frsm_tsk* sẽ được dùng. Trạng thái của Task đích khi hàm *tk_rel_wai* được gọi và kết quả của nó trong mỗi trạng thái được mô tả trong bảng 4.3

Trạng thái của Task đích	tk_rel_tsk ercd Parameter	Xử lý
Trạng thái thực thi (RUN, READY) (không cho Task gọi hàm)	E_OBJ	Không xử lý
Trạng thái RUN (cho Task gọi hàm)	E_OBJ	Không xử lý
Trạng thái WAIT	E_OK	Giải phóng trạng thái đợi
Trạng thái DORMANT	E_OBJ	Không xử lý
Trạng thái NON-EXISTENT	E_OBJ	Không xử lý

*Mã lỗi E_RLWAI được trả về cho Task đích mà không cần sự cấp phát gì về tài nguyên.

Bảng 4.3: Trạng thái của Task và kết quả của sự thực thi hàm *tk_rel_wai*

5. tk_sus_tsk: Suspend Task

[C Language Interface]

```
ER ercd = tk_sus_tsk ( ID tskid ) ;
```

[Parameters]

ID tskid Task ID

[Return Parameters]

ER	ercd	Mã lỗi (Error codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*tskid* không hợp lệ hay không được sử dụng
E_NOEXS	Task được chỉ định trong *tskid* không tồn tại
E_OBJ	Trạng thái của đối tượng không hợp lệ (gọi cho chính Task gọi hàm hay cho một Task đang ở trạng thái DORMANT)
E_CTX	Lỗi ngữ cảnh (hàm được gọi từ một *Task độc lập với ngữ cảnh* hay *dispatch* bị vô hiệu)
E_QOVR	Hàng đợi các yêu cầu suspend bị tràn (quá nhiều yêu cầu suspend được xếp hàng trong *suscnt*) hay hệ thống không cho phép các yêu cầu suspend lồng lên nhau

[Description]

- Đặt trạng thái của một Task được chỉ định trong *tskid* vào trạng thái SUSPEND và ngắt sự thực thi của Task đó.
- Trạng thái SUSPEND được giải phóng bằnh cách gọi các hàm system call *tk_rsm_tsk* hay *tk_frsm_tsk*.
- Nếu *tk_sus_tsk* được gọi cho một Task đang ở trạng thái WAIT, thì Task đó sẽ đi vào trạng thái WAIT-SUSPEND. Và nếu sau đó khi các điều kiện giải phóng Task được thỏa mãn thì Task đó sẽ đi vào trạng thái SUSPEND. Nếu *tk_rsm_tsk* được gọi cho một Task đang ở trạng thái WAIT- SUSPEND,Task đó sẽ trở lại trạng thái WAIT.
- Khi *tk_sus_tsk* được gọi nhiều lần cho cùng một Task, Task đó sẽ được đặt trong trạng thái SUSPEND nhiều lần (xếp lồng lên nhau). Điều này có nghĩa là để Task đó trở về trạng thái ban đầu của nó chỉ khi nào hàm *tk_rsm_tsk* được gọi cho nó bằng với số lần mà *tk_sus_tsk* đã gọi (*susnt*). Do đó , bộ xếp lồng của cặp *tk_sus_tsk* --- *tk_rsm_tsk* là có thể.
- Các yêu cầu suspend được xếp lồng và giới hạn của nó là phụ thuộc vào sự hiện thực.
- Các mã lỗi trả về được mô tả trong phần Error Codes.

6. tk_rsm_tsk: Resume Task

tk_frsm_tsk: Force Resume Task

[C Language Interface]

```
ER ercd = tk_rsm_tsk ( ID tskid ) ;
ER ercd = tk_frsm_tsk ( ID tskid ) ;
```

[Parameters]

ID	tskid	Task ID

[Return Parameters]

ER	ercd	Mã lỗi (Error codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*tskid* không hợp lệ hay không được sử dụng
E_NOEXS	Task được chỉ định trong *tskid* không tồn tại
E_OBJ	Trạng thái của đối tượng không hợp lệ (gọi cho một Task không ở trong trạng thái SUSPEND hay gọi cho chính Task gọi hàm hay cho một Task đang ở trạng thái DORMANT)

[Description]

- Hàm system call *tk_rel_wai* giải phóng trạng thái SUSPEND của một Task được chỉ định trong *tskid*.
- Nếu Task đích sớm được đặt trong trạng thái SUSPEND bởi hàm systemcall *tk_sus_tsk*, thì hàm system call này sẽ giải phóng trạng thái SUSPEND và khôi phục sự thực thi của Task đó.
- Nếu nó được gọi cho Task đanh ở trạng thái WAIT-SUSPEND, Task đó sẽ vào trạng thái WAIT.
- Hàm system call này không thể gọi cho chính Task gọi nó, nếu không mã lỗi E_OBJ sẽ được trả về.
- Khi hàm *tk_rsm_tsk* thực thi một lần thì chỉ giải phóng một yêu cầu suspend được lồng lên nhau (suspend nhiều lần). Nếu hàm *tk_sus_tsk* được gọi nhiều lần để suspend Task đích (*suscnt* ≥ 2), thì Task đích vẫn ở trong trạng thái SUSPEND sau khi hàm *tk_rsm_tsk* được thực thi. Ngược lại khi ta gọi hàm *tk_frsm_tsk* thì tất cả các yêu cầu suspend lồng nhau đều bị xóa bỏ ngay cả khi *suscnt* = 1.
- Sau khi một Task ở trạng thái RUN hay READY bị đặt trong trạng thái SUSPEND bởi hàm *tk_sus_tsk* và được khôi phục lại bởi hàm *tk_rsm_tsk* hay *tk_frsm_tsk*, thì Task đó sẽ có thứ tự (precedence) thực thi thấp nhất giữa các Task có cùng độ ưu tiên. Ví dụ các hàm system call sau được thực thi cho Task A và Task B có cùng độ ưu tiên và kết quả của nó như sau:

```
tk_sta_tsk (tskid=task_A, stacd_A);
tk_sta_tsk (tskid=task_B, stacd_B);
        /* Theo luật của FCFS, thứ tự thực thi là task_A --> task_B. */

tk_sus_tsk (tskid=task_A);
tk_rsm_tsk (tskid=task_A);
        /* Trong trường hợp này, thứ tự thực thi là task_A --> task_B. */
```

7. tk_dly_tsk: Delay Task

[C Language Interface]

```
ER ercd = tk_dly_tsk ( RELTIM dlytim ) ;
```

[Parameters]

RELTIM	dlytim	Thời gian làm trễ

[Return Parameters]

ER	ercd	Mã lỗi (Error codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_NOMEM	Thiếu bộ nhớ
E_PAR	Lỗi tham số (*dlytim* không hợp lệ)
E_RLWAI	Trạng thái WAIT của Task được giải phóng (do Task nhận được tk_rel_wai khi đang ở trạng thái WAIT)
E_DISWAI	Trạng thái WAIT của Task được giải phóng bởi trạng thái WAIT của Task bị vô hiệu
E_CTX	Lỗi ngữ cảnh (hàm được gọi từ một *Task độc lập với ngữ cảnh* hay *dispatch* bị vô hiệu)

[Description]

- Tạm thời làm ngưng sự thực thi của Task gọi hàm system call này và đợi cho tới khi thời gian *dlytim* kết thúc. Trạng thái của Task trong khi đợi thời gian trễ là trạng thái WAIT và có thể được giải phóng bởi hàm *tk_rel_wai*.
- Nếu trạng thái của Task trong khi đợi thời gian trễ là trạng thái WAIT-SUSPEND hay SUSPEND thì thời gian vẫn tiếp tục được đếm trong trạng thái SUSPEND.
- Đơn vị của *dlytim* cũng tương tự như thời gian của hệ thống (= 1 ms).
- Hàm system call này khác với hàm *tk_slp_tsk* về sự kết thúc bình thường của nó, không có mã lỗi trả về khi thời gian trễ hay hàm *tk_dly_tsk* kết thúc. Hơn thế, trạng thái đợi sẽ không được giải phóng bởi hàm *tk_wup_tsk* trong suốt thời gian làm trễ. Chỉ có một cách duy nhất để kết thúc *tk_dly_tsk* trước khi thời gian làm trễ kết thúc bằng cách gọi hàm *tk_ter_tsk* hay *tk_rel_wai*.

8. tk_sig_tev : Signal Task Event

[C Language Interface]

ER ercd = tk_sig_tev (ID tskid, INT tskevt) ;

[Parameters]

ID	tskid	Task ID
INT	tskevt	Số của task event

[Return Parameters]

ER	ercd	Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường

E_ID	*tskid* không hợp lệ hay không được sử dụng
E_PAR	Lỗi tham số (*tskevt* không hợp lệ)
E_NOEXS	Task được chỉ định trong *tskid* không tồn tại
E_OBJ	Trạng thái của đối tượng không hợp lệ (gọi cho một Task đang ở trạng thái DORMANT)

[Description]

- Gửi một task event *tskevt* đến một Task được chỉ định trong *tskid*. Có tất cả 8 loại task event cho mỗi task và được đánh số từ 1 đến 8.
- Số lượng task event đã gửi không được lưu trữ, chỉ có hay không có task event xuất hiện hay không.
- Khi TSK_SELF (=0) được gán cho *tskid* thì Task đích cũng chính là Task gọi hàm.Tuy nhiên, khi *tskid* = TSK_SELF được chỉ định trong hàm system call được gọi từ một Task-independent portion, thì mã lỗi E_ID sẽ được trả về.
- Các hàm task event được sử dụng cho sự đồng bộ hóa cũng giống như hàm *tk_slp_tsk* và *tk_wup_tsk*, nhưng khán nhau về cách sử dụng như sau:
    - Số lượng yêu cầu đánh thức (task event) không được lưu trữ.
    - Các yêu cầu đánh thức có thể được phân loại thành tám loại event.
- Thông thường các hàm task event được sử dung trong middleware, và không nên dùng trong các ứng dụng thông thường. Sử dụng tk_slp_tsk và tk_wup_tsk được đề nghị cho các ứng dung thông thường.

9. tk_wai_tev : Wait Task Event

[C Language Interface]

```
INT tevptn = tk_wai_tev ( INT waiptn, TMO tmout ) ;
```

[Parameters]

INT waiptn Mẫu task event

TMO tmout Thời gian timeout

[Return Parameters]

INT tevptn	Trạng thái của task event khi đợi được giải phóng
	hoặc Mã lỗi (Error codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_PAR	Lỗi tham số (*dlytim* không hợp lệ)
E_TMOUT	Thăm dò bị lỗi hay timeout
E_RLWAI	Trạng thái WAIT của Task được giải phóng (do Task nhận được tk_rel_wai khi đang ở trạng thái WAIT)

E_DISWAI	Trạng thái WAIT của Task được giải phóng bởi trạng thái WAIT của Task bị vô hiệu
E_CTX	Lỗi ngữ cảnh (hàm được gọi từ một *Task độc lập với ngữ cảnh* hay *dispatch* bị vô hiệu)

[Description]

- Đợi cho tới khi xuất hiện một trong những task event được chỉ định trong *waiptn.*
- Khi trạng thái đợi được giải phóng bởi task event, những task event được chỉ định trong *waiptn* thì được xóa (task event &= ~*waipnt*).
- Trạng thái của task event xuất hiện khi trạng thái đợi được giải phóng (trước khi bị xóa) được trả về thông qua tham số *tevptn*.
- Tham số *waipnt* và *tevpnt* chứa giá trị của phép OR luận lý của các bit cho mỗi task event trong dạng 1<< (số của task event – 1).
- Một thời gian đợi tối đa (timeout) có thể được thiết lập trong *tmout*. Nếu bị timeout trước khi điều kiện giải phóng đợi thỏa mãn (*tk_sig_tev* không được thực thi), thì hàm system call này sẽ kết thúc, và trả về mã lỗi E_TMOUT. Chỉ có giá trị dương mới được gán cho *tmout*. Đơn vị của timeout là ms. Khi TMO_POL = 0 được gán cho *tmout*, thì mã lỗi E_TMOUT được trả về mà không vào trạng thái WAIT ngay cả khi không có tsk event xuất hiện. Khi TMO_FEVR = -1 được gán cho *tmout*, thì task sẽ đợi cho tới khi có task event mà không bị timeout.

10. tk_dis_wai : Disable Task Wait

[C Language Interface]

INT tskwai = tk_dis_wai (ID tskid, UINT waitmask) ;

[Parameters]

ID	tskid	Task ID
UINT	waitmask	Thiết lập trạng thái đợi bị vô hiệu

[Return Parameters]

INT tskwai	trạng thái của Task sau khi trạng thái đợi bị vô hiệu
	hoặc Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*tskid* không hợp lệ hay không được sử dụng
E_PAR	Lỗi tham số (*waitmask* không hợp lệ)
E_NOEXS	Task được chỉ định trong *tskid* không tồn tại

[Description]

- Làm vô hiệu các trạng thái đợi được thiết lập trong *waitmask* cho một Task được chỉ định trong *tskid*. Nếu Task đó đang đợi bởi một nguyên nhân nào đó có trong *waitmask*, thì trạng thái đợi được giải phóng.

- *watmask* được chỉ định như là phép OR luận lý của các sự kết hợp các nguyên nhân đợi

```
#define  TTW_SLP     0x00000001    /* Wait caused by sleep */
#define  TTW_DLY     0x00000002    /* Wait for task delay */
#define  TTW_SEM     0x00000004   /* Wait for semaphore */
#define  TTW_FLG     0x00000008    /* Wait for event flag */
#define  TTW_MBX     0x00000040    /* Wait for mailbox */
#define  TTW_MTX     0x00000080    /* Wait for mutex */
#define  TTW_SMBF    0x00000100    /* Wait for message buffer sending */
#define  TTW_RMBF    0x00000200    /* Wait for message buffer receipt */
#define  TTW_CAL     0x00000400    /* Wait on rendezvous call */
#define  TTW_ACP     0x00000800    /* Wait for rendezvous acceptance */
#define  TTW_RDV     0x00001000    /* Wait for rendezvous completion */
#define  TTW_MPF     0x00002000    /* Wait for fixed-size memory pool */
#define  TTW_MPL     0x00004000  /* Wait for variable-size memory pool */
#define  TTW_EV1     0x00010000    /* Wait for task event #1 */
#define  TTW_EV2     0x00020000    /* Wait for task event #2 */
#define  TTW_EV3     0x00040000    /* Wait for task event #3 */
#define  TTW_EV4     0x00080000    /* Wait for task event #4 */
#define  TTW_EV5     0x00100000    /* Wait for task event #5 */
#define  TTW_EV6     0x00200000    /* Wait for task event #6 */
#define  TTW_EV7     0x00400000   /* Wait for task event #7 */
#define  TTW_EV8     0x00800000    /* Wait for task event #8 */
#define  TTX_SVC     0x80000000    /* Extended SVC disabled */
```

- TTX_SVC là một trường hợp chỉ định đặc biệt, nó không làm vô hiệu trạng thái đợi của task mà làm vô hiệu việc gọi một hàm SVC mở rộng. Nếu TTX_SVC được chỉ định, khi một task cố gắng gọi một hàm SVC mở rộng, thì mã lỗi E_DISWAI được trả về mà không gọi hàm đó. Việc chỉ định TTX_SVC sẽ không làm kết thúc một hàm SVC mở rộng đã được gọi trước khi chỉ định.
- Khi giá trị trả về của tham số *tskwait* là 0 thì có nghĩa là Task chưa đi vào trạng thái WAIT (hay đã được giải phóng), nếu khác 0 thì Task đang ở trong trạng thái WAIT vì một lý do nào khác ngoài những lý do trong *waitmask*.
- Khi một task được giải phóng khỏi các trạng thái đợi bởi hàm *tk_dis_wai* hay nó bị ngăn cản đi vào trạng thái WAIT trong khi hàm system call này có hiệu lực, thì mã lỗi E_DISWAI sẽ trả về cho task đó. Nghĩa là khi một hàm system call nào đó gọi cho task đó và làm nó đi vào trạng thái WAIT, trong khi nó đang ở trạng thái bị vô hiệu đợi thì mã lỗi E_DISWAI sẽ trả về cho hàm system call đó.
- Khi TSK_SELF (=0) được gán cho *tskid* thì Task đích cũng chính là Task gọi hàm.Tuy nhiên, khi *tskid* = TSK_SELF được chỉ định trong hàm system call được gọi từ một Task-independent portion, thì mã lỗi E_ID sẽ được trả về.

11. tk_ena_wai : Cho phép Task Wait

[C Language Interface]

```
ER ercd = tk_ena_wai ( ID tskid ) ;
```

[Parameters]

ID	tskid	Task ID

[Return Parameters]

ER	ercd	Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*tskid* không hợp lệ hay không được sử dụng
E_NOEXS	Task được chỉ định trong *tskid* không tồn tại

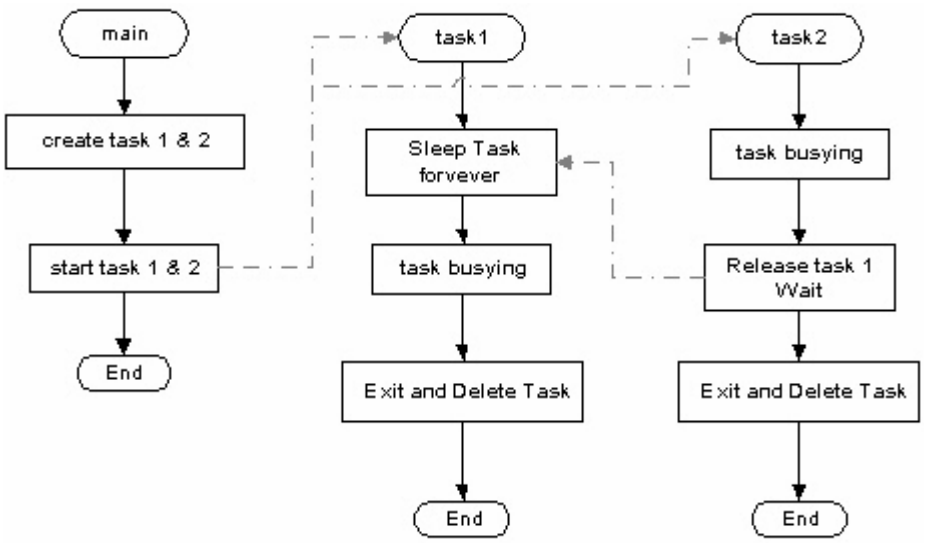
[Description]

- Giải phóng tất cả các điều kiện làm trạng thái đợi bị vô hiệu được thiết lập bởi hàm *tk_dis_wai* cho một task được chỉ định trong *tskid*.
- Khi TSK_SELF (=0) được gán cho *tskid* thì Task đích cũng chính là Task gọi hàm.Tuy nhiên, khi *tskid* = TSK_SELF được chỉ định trong hàm system call được gọi từ một Task-independent portion, thì mã lỗi E_ID sẽ được trả về.

3. THỰC HÀNH

3.1 Bài tập

Chương trình có sự đồng bộ giữa các task theo mô hình sau:



Đọc,hiểu và chạy chương trình đơn giản sau. Nắm rõ nguyên tắc để viết các ứng dụng sau này:

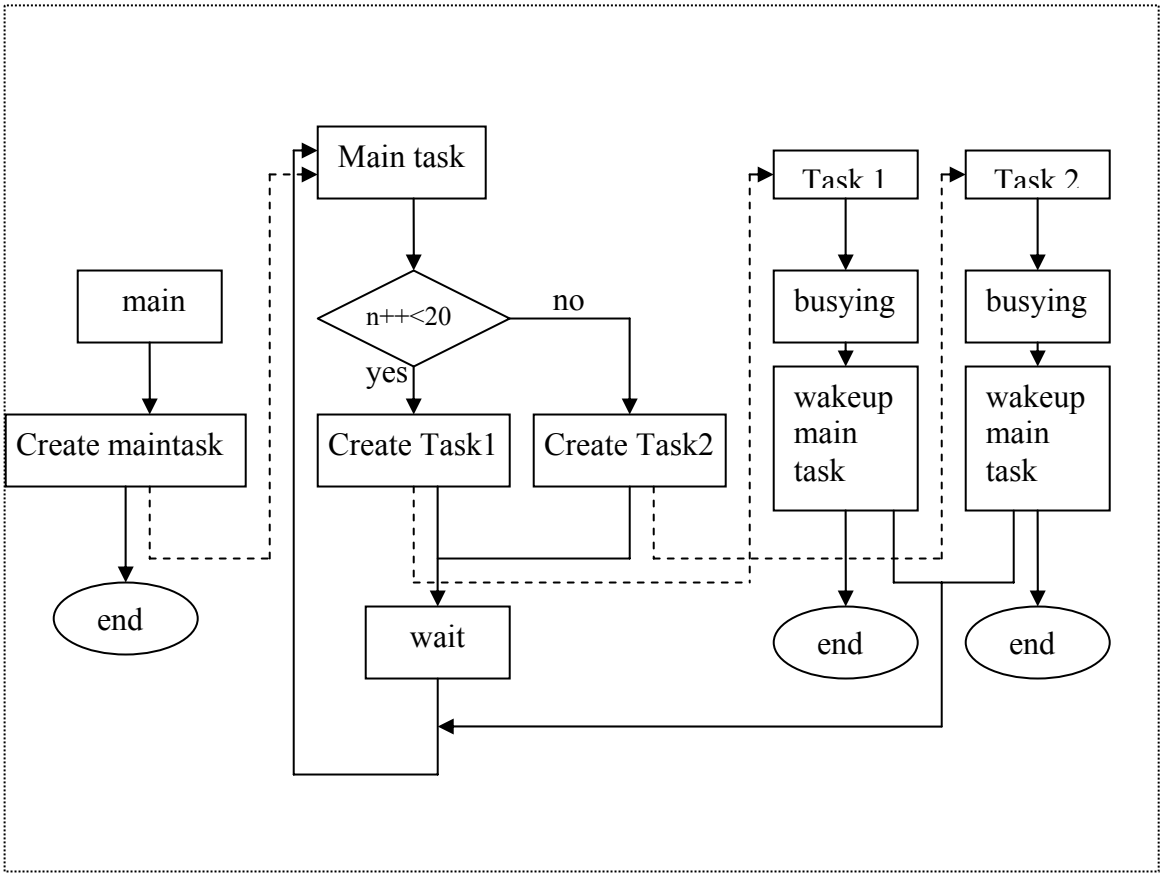
```
#include <basic.h>
#include <tk/tkernel.h>
#include <stdio.h>

IMPORT void task1(INT, VP);
ID taskid1 = -1;
ID taskid2 = -1;
```

```
/*****************************************************************
task1 wakeup every 10 seconds and display print
 *****************************************************************/
IMPORT void task2(INT stacd, VP exinf)
{
   ER ercd = 0;
   INT cnt = 0;

   while (cnt++ < 20)
   {
     printf("task %d\n", taskid2);
   }
   ercd = tk_rel_wai(taskid1);
   if (ercd < 0)
   {
   printf("tk_rel_tsk error = %x\n", ercd);
   }
   taskid2 = -1;
   tk_exd_tsk();
}


/*****************************************************************************
main
*****************************************************************************/
EXPORT ER main( INT ac, UB *av[] )
{
    T_CTSK ctsk;
 ER ercd = 0;


 printf("main: (ac=%d)\n", ac);

 if (ac < 0)
 {
   if (taskid1 >= 0)
   {
     ercd = tk_ter_tsk(taskid1);
     if (ercd < 0)
       printf("tk_ter_tsk err = %x\n", ercd);
     ercd = tk_del_tsk(taskid1);
     if (ercd < 0)
       printf("tk_del_tsk err = %x\n", ercd);
   }
   if (taskid2 >= 0)
   {
     ercd = tk_ter_tsk(taskid2);
     if (ercd < 0)
       printf("tk_ter_tsk err = %x\n", ercd);
     ercd = tk_del_tsk(taskid2);
     if (ercd < 0)
       printf("tk_del_tsk err = %x\n", ercd);
   }
   taskid1 = -1;
   taskid2 = -1;
   goto ext;
 }

 ctsk.exinf = (VP)0x74736574;
```

```
ctsk.tskatr = TA_HLNG | TA_RNG0;
ctsk.task = task1;
ctsk.itskpri = 80;
ctsk.stksz = 1024 * 4;
taskid1 = tk_cre_tsk(&ctsk);
printf("tk_cre_tsk: (taskid = %x)\n", taskid1);
if (taskid1 < E_OK)
{
   goto ext;
}

ctsk.exinf = (VP)0x74736574;
ctsk.tskatr = TA_HLNG | TA_RNG0;
ctsk.task = task2;
ctsk.itskpri = 80;
ctsk.stksz = 1024 * 4;
taskid2 = tk_cre_tsk(&ctsk);
printf("tk_cre_tsk: (taskid = %x)\n", taskid2);
if (taskid2 < E_OK)
{
   ercd = tk_del_tsk(taskid1);
   if (ercd < 0)
     printf("tk_del_tsk err = %x\n", ercd);
   goto ext;
}

tk_sta_tsk(taskid1, 0);
tk_sta_tsk(taskid2, 0);

/*end*/
ext:
printf("main ended\n");
return 0;
}

/*****************************************************************
task1 wakeup every 10 seconds and display print
 *****************************************************************/
IMPORT void task1(INT stacd, VP exinf)
{
   ER ercd = 0;
   INT cnt = 0;

   ercd = tk_slp_tsk(TMO_FEVR);
   if (ercd < 0 && ercd != E_RLWAI)
   printf("\ttk_slp_tsk err = %x\n", ercd);
   else if (ercd == E_RLWAI)
   printf("\tE_RLWAI = %x\n", ercd);
   while (cnt++ < 20)
   {
     printf("\ttask %d: \n", taskid1);
   }
   taskid1 = -1;
   tk_exd_tsk();
}
```

3.2 Bài tập đề nghị

Bạn thử viết một chương trình nhỏ có sơ đồ giao tiếp giữa các task như sau:


Main task
Task 1
Task 2
main
n++<20
no
yes
busying
busying
Create maintask
Create Task1
Create Task2
wakeup main task
wakeup main task
end
wait
end
end

BÀI THỰC HÀNH SỐ 6

SEMAPHORE – MESSAGE BUFFER

1. MỤC TIÊU

- Giúp sinh viên hiểu rõ và biết cách sử dụng các đối tượng độc lập với task để quản lý đồng bộ và giao tiếp giữa các task.
- Tìm hiểu và sử dụng Semaphore và Message buffer để đồng bộ và giao tiếp giữa các task.

2. LÝ THUYẾT

Các hàm đồng bộ và giao tiếp sử dụng những đối tượng độc lập với task để đạt đến sự đồng bộ và giao tiếp giữa các task. Những đối tượng được OS cung cấp cho mục tiêu này bao gồm semaphore, event flag, và mailbox. Ngoài ra để quản lý sự giao tiếp và đồng bộ có tính phức tạp hơn T-Kernel còn cung cấp các đối tượng khác như Mutex, Message buffer.

Trong bài thí nghiệm này sẽ đề cập đến các hàm system call của hai đối tượng cơ bản và đại diện là Semaphore và Message buffer.

2.1 Semaphore

Mỗi semaphore là một đối tượng cho biết tính sẵn sàng của một tài nguyên và số lượng của nó.

Mỗi semaphore được dùng để kiểm soát *vùng tranh chấp* (mutual exclusion) và đồng bộ hoá khi sử dụng một tài nguyên. OS cung cấp các hàm để tạo và xoá một semaphore, đạt được hay trả tài nguyên tương ứng với semaphore, và tham khảo trạng thái của semaphore. Mỗi semaphore được định danh bởi một số ID gọi là semaphore ID.

Mỗi semaphore chứa một số đếm tài nguyên cho biết tài nguyên tương ứng có tồn tại hay không, số lượng của nó và một hàng đợi các task đang chờ để chiếm giữ tài nguyên đó. Khi một task trả về *m* tài nguyên, nó sẽ làm tăng số đếm tài nguyên của semaphore tương ứng lên *m*. Khi một task chiếm giữ *n* tài nguyên, nó sẽ làm giảm số đếm tài nguyên của semaphore tương ứng xuống *n*. Nếu số lượng tài nguyên tương ứng của một semaphore không đủ và có một task cố gắng chiếm giữ tài nguyên, thì task đó sẽ đi vào trạng thái WAIT cho tới khi tài nguyên được trả về. Một task đang đợi các tài nguyên của một semaphore thì được đặt trong hàn đợi của semaphore tương ứng.

Để ngăn cản việc quá nhiều tài nguyên được trả về cho một semaphore, một số lượng tài nguyên tối đa được thiết lập cho mỗi semaphore. Lỗi sẽ được thông báo nếu số lượng tài nguyên trả về cho một semaphore vượt quá số lượng tối đa.

❖ **Các hàm system call của Semaphore :**

1. tk_cre_sem : Tạo một Semaphore

[C Language Interface]

```
ID semid = tk_cre_sem ( T_CSEM * pk_csem ) ;
```

[Parameters]

T_CSEM * pk_csem	Thông tin về semaphore được tạo

Chi tiết của pk_csem :

VP	exinf	Tthông tin phụ được thêm vào
ATR	sematr	Các thuộc tính của semaphore
INT	isemcnt	Số tài nguyên ban đầu của semaphore
INT	maxsem	Số tài nguyên tối đa của semaphore

[Return Parameters]

ID	semid	Semaphore ID
hoặc	Mã lỗi (Error codes)	

[Error Codes]

E_OK	Kết thúc bình thường
E_NOMEM	Thiếu bộ nhớ (vùng nhớ cho khối điều khiển không được cấp phát)
E_LIMIT	Số đếm semaphore vượt quá giới hạn của hệ thống
E_RSATR	Lỗi thuộc tính (*sematr* không hợp lệ hay không sử dụng được)
E_PAR	Lỗi tham số (*pk_csem* không hợp lệ; *isemcnt* hay *maxsem* có giá trị âm hay không hợp lệ)

[Description]

- Tạo một semaphore và gán cho nó một semaphore ID.
- Hàm system call này định vị một khối điều khiển cho semaphore được tạo, khởi tạo các tham số trong *pk_csem*. Nó phải có thể thiết lập *maxsem* tối thiểu là 65535. Việc giá trị lớn hơn 65535 là phụ thuộc vào hiện thực.
- *exinf* có thể được sử dụng tự do bởi người dùng để chèn thêm tthông tin về semaphore, và được tham khảo bằng hàm *tk_ref_sem*. Nếu tthông tin của semaphore là một vùng lớn, ứng dụng phải xin cấp phát vùng nhớ riêng biệc và đặt địa chỉ vào *exinf*.
- *sematr* cho biết các thuộc tính hệ thống trong các bit thấp của nó và tthông tin hiện thực trong các bit cao.

```
sematr := (TA_TFIFO | | TA_TPRI) | (TA_FIRST | | TA_CNT) |
                                  [TA_NODISWAI]
```

TA_TFIFO	Các task đợi trong hàng đợi theo thứ tự FIFO
TA_TPRI	Các task đợi trong hàng đợi theo độ ưu tiên
TA_FIRST	Task đầu tiên trong hàng đợi có quyền chiếm giữ semaphore
TA _CNT	Những task có ít yêu cầu hơn sẽ có quyền chiếm giữ semaphore
TA _NODISWAI	Cấm làm vô hiệu trạng thái đợi bởi hàm *tk_dis_wai*

```
#define    TA_TFIFO        0x00000000
#define    TA_TPRI         0x00000001
#define    TA_FIRST        0x00000000
#define    TA_CNT          0x00000002
  #define  TA_NODISWAI     0x00000080
```

2. tk_del_sem : Xóa bỏ một Semaphore

[C Language Interface]

ER ercd = tk_del_sem (ID semid) ;

[Parameters]

ID	semid	Semaphore ID

[Return Parameters]

ER	ercd	Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*semid* không hợp lệ hay không được sử dụng
E_NOEXS	Semaphore được chỉ định trong *semid* không tồn tại

[Description]

- Xóa bỏ semaphore có ID là *semid* .
- Hàm system call này giải phóng semaphore ID và khối điều khiển nó.
- Hàm này kết thúc bình thường ngay cả khi có một task đang đợi semaphore, nhưng mã lỗi E_DLT được trả về cho task đó.

3. tk_sig_sem : Trả tài nguyên về cho semaphore

[C Language Interface]

ER ercd = tk_sig_sem (ID semid, INT cnt) ;

[Parameters]

ID	semid	Semaphore ID
INT	cnt	Số tài nguyên trả về

[Return Parameters]

ER	ercd	Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*semid* không hợp lệ hay không được sử dụng
E_PAR	Lỗi tham số ($cnt \leq 0$)
E_NOEXS	Semaphore được chỉ định trong *semid* không tồn tại
E_QOVR	Hàng đợi bị tràn (*semcnt* vượt quá giới hạn)

[Description]

- Trả về cho semaphore (*semid*) một số lượng tài nguyên được chỉ định trong *cnt*. Nếu có một task đang đợi semaphore, thì số lượng tài nguyên mà nó yêu cầu được kiểm tra và

được cấp phát nếu có thể. Một task mà được cấp phát tài nguyên sẽ vào trạng thái READY. Trong một số trường hợp có thể có nhiều hơn một task được như vậy.

- Nếu số lượng tài nguyên trả về làm vượt số tài nguyên tối đa có thể có của semaphore, thì mã lỗi E_QOVR được trả về. Trong trường hợp này tài nguyên sẽ không được trả về và biến đếm (*semcnt*) của semaphore cũng không thay đổi.
- Lỗi sẽ không được trả về nếu *semcnt* vượt quá giá trị ban đầu (*isemcnt*).

4. tk_wai_sem : Đợi tài nguyên từ semaphore

[C Language Interface]

ER ercd = tk_wai_sem (ID semid, INT cnt, TMO tmout) ;

[Parameters]

ID	semid	Semaphore ID
INT	cnt	Số tài nguyên yêu cầu
TMO	tmout	Thời gian timeout

[Return Parameters]

ER	ercd	Mã lỗi (Error codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*semid* không hợp lệ hay không được sử dụng
E_PAR	Lỗi tham số (*tmout* ≤ (2), *cnt* ≤ 0)
E_NOEXS	Semaphore được chỉ định trong *semid* không tồn tại
E_DLT	Đối tượng được đợi bị xoá (semaphore được chỉ định bị xoá khi task đang đợi)
E_TMOUT	Thăm dò bị lỗi hay timeout
E_RLWAI	Trạng thái WAIT của Task được giải phóng (do Task nhận được *tk_rel_wai* khi đang ở trạng thái WAIT)
E_DISWAI	Trạng thái WAIT của Task được giải phóng bởi trạng thái WAIT của Task bị vô hiệu
E_CTX	Lỗi ngữ cảnh (hàm được gọi từ một *Task độc lập với ngữ cảnh* hay *dispatch* bị vô hiệu)

[Description]

- Thu từ semaphore (*semid*) một số lượng tài nguyên được chỉ định trong *cnt*. Nếu số lượng tài nguyên yêu cầu được cấp phát, thì task mà gọi hàm system call này sẽ không đi vào trạng thái WAIT và tiếp tục thực thi. Trong trường hợp này, biến đếm semaphore (*semcnt*) sẽ giảm đi một lượng là *cnt.* Nếu tài nguyên không có sẵn, thì hàm system call này sẽ vào trạng thái WAIT và task được đặt trong hàng đợi. Và *semcnt* sẽ không thay đổi trong trường hợp này.

- Một thời gian đợi tối đa (timeout) có thể được thiết lập trong *tmout*. Nếu bị timeout trước khi điều kiện giải phóng đợi thỏa mãn (*tk_sig_sem* không được thực thi), thì hàm system call này sẽ kết thúc, và trả về mã lỗi E_TMOUT. Chỉ có giá trị dương mới được gán cho *tmout*. Đơn vị của timeout là *ms*. Khi TMO_POL = 0 được gán cho *tmout*, thì mã lỗi E_TMOUT được trả về mà không vào trạng thái WAIT ngay cả khi không có tài nguyên nào bị chiếm giữ. Khi TMO_FEVR = -1 được gán cho *tmout*, thì task sẽ đợi cho tới khi chiếm giữ được tài nguyên mà không bị timeout.

5. tk_ref_sem : Tham khảo một semaphore

[C Language Interface]

ER ercd = tk_ref_sem (ID semid, T_RSEM * pk_rsem) ;

[Parameters]

ID	semid	Semaphore
T_RSEM*	pk_rsem	Địa chỉ của gói tthông tin về trạng thái semaphore

Chi tiết pk_rsem :

VP	exinf	Thông tin mở rộng
ID	wtsk	Thông tin task đang đợi
INT	semcnt	Biến đếm semaphore

(Những tham số khác tùy theo hiện thực có thêm vào sau)

[Return Parameters]

ER	ercd	Mã lỗi

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*semid* không hợp lệ hay không được sử dụng
E_NOEXS	Semaphore được chỉ định trong *semid* không tồn tại
E_PAR	Lỗi tham số (địa chỉ của gói tthông tin trả về không dùng được)

[Description]

- Tham khảo trạng thái của semaphore có semaphore ID là *semid*, truyền trong các thông số trả về biến đếm semaphore hiện tại (*semcnt*), tthông tin task đang đợi semaphore (*wtsk*), và tthông tin mở rộng (*exinf*).
- Tham số *wtsk* cho biết ID của một task đang đợi semaphore. Nếu có nhiều hơn một task đang đợi thì ID của task đứng đầu hàng đợi sẽ được trả về. Nếu không có task nào thì *wtsk* = 0 được trả về.
- Nếu semaphore tương ứng không tồn tại, mã lỗi E_NOEXS được trả về.

2.2 Message Buffer :

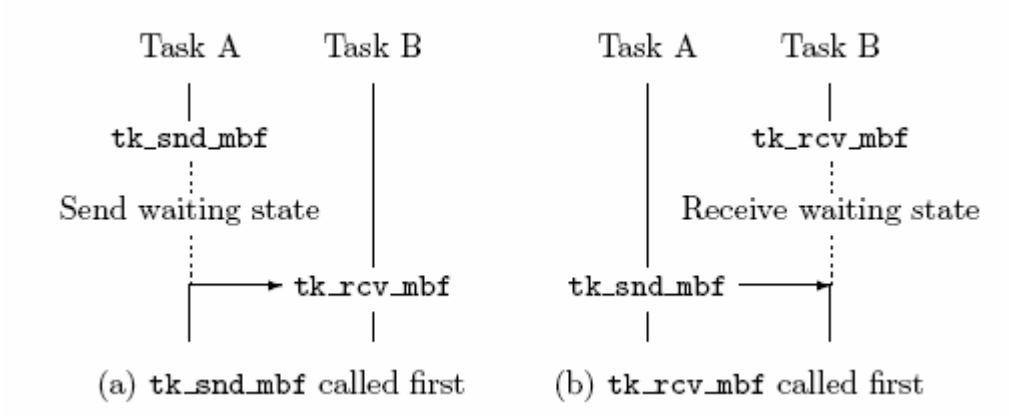
Message buffer là một đối tượng được dùng để đồng bộ và giao tiếp giữa các task bằng cách gởi những message có kích thước không cố định. T-Kernel cung cấp các hàm cho việc tạo, xoá một message buffer và các hàm gởi nhận message cũng như hàm tham khảo đến trạng thái của message

buffer. Mỗi một message buffer được định ra bởi một số ID duy nhất và được gọi là message buffer ID.

Mỗi một message buffer sẽ có một queue để lưu giữ các messagecần gởi và một queue dùng để chứa các task đang chờ để nhận các message. Mỗi message buffer sẽ có một vùng không gian dùng để lưu trữ các message cần gởi. Khi một task cần gởi message thì nó sẽ copy message cần gởi và gởi cho message buffer ,và trong trường hợp message buffer không còn đủ bộ nhớ thì message đó phải được đưa vào hàng chờ cho đến khi message buffer có đủ bộ nhớ và task gởi message sẽ được đưa vào hàng chờ . Bên phía nhận thì chỉ cần copy messsage có trong message buffer về là được , trong trường hợp message buffer không có message nào thì task sẽ đi vào trạng thái WAIT cho đến khi một message tiếp theo sẽ được gởi. Các task chờ để nhận message sẽ được đưa vào một queue của message buffer.

Một chức năng dùng để đồng bộ message giữa các task đó là thiết lập kích thước của message buffer là 0. Trong trường hợp này thì cả task gởi và nhận đều đi vào trạng thái WAIT cho đến khi cả 2 task cùng gọi các hàm system call tương ứng cho từng task và lúc này message sẽ được gởi đi.

Chức năng của message buffer khi thiết lập kích thước của nó là 0 sẽ được giải thích rõ hơn ở đây bằng cách sử dụng ví dụ bên dưới, khi mà cả 2 task A và B đang thực thi ở chế độ bất đồng bộ với nhau.



Hình 4.2 : Đồng bộ việc giao tiếp bằng cách sử dụng Message buffer

- Nếu task A gọi hàm tk_snd_mbf trước thì nó sẽ đi vào trạng thái WAIT cho đến khi task B gọi hàm tk_rcv_mbf. Trong trường hợp này task A sẽ được đưa vào queue của những task đang chờ để gởi message (hình a).
- Nếu task B gọi hàm tk_rcv_mbf trước thì task B sẽ đi vào trạng thái WAIT cho đến khi task A gọi hàm tk_snd_mbf. Trong trường hợp này task B sẽ được đưa vào queue của những task đang chờ nhận message của message buffer (hình b).
- Tại thời điểm mà cả task A và task B đều gọi hàm system call tương ưng của nó thì message sẽ được gởi đi và giải phóng các task này khỏi trạng thái WAIT.
- Task đang ở trạng thái chờ gởi message thì sẽ được gởi message theo thứ tự mà nó có ở trong queue chờ của message buffer. Ví dụ sau sẽ giúp làm rõ vấn đề này. Giả sử trong queue chờ để gởi message của message buffer đang có 2 task và có thứ tự là A , B .Task A muốn gởi message 40-byte , còn task B thì gởi message 10-byte. Nếu một task nào đó nhận một message có 20-byte từ message buffer , điều này làm cho không gian trống của message tăng lên 20 byte. Nhưng theo thứ tự trong queue thì task A phải được gởi trước rồi mới tới task B do đó mà dù task B có thể gởi được nhưng phải chờ task A gởi trước.

Điểm khác biệt cơ bản giữa message buffer và mail box đó là message buffer muốn gởi một message thì phải copy chúng rồi mới gởi được.Và ở đây chúng ta xem message buffer sẽ được hiện thực như một buffer vòng.

❖ **Các hàm system call của Message Buffer :**

1. tk_cre_mbf : Tạo một Message buffer

[C Language Interface]

ID mbfid = tk_cre_mbf (T_CMBF *pk_cmbf) ;

[Parameters]

T_CMBF* pk_cmbf Thông tin về Message buffer được tạo

Chi tiết của pk_cmtx :

VP	exinf	Thông tin phụ được thêm vào
ATR	mbfatr	Các thuộc tính của Message buffer
INT	bufsz	Kích thước của message buffer được tạo
INT	maxmsz	Kích thước tối đa của message cần gởi

[Return Parameters]

ID mbfid Message buffer ID

hoặc Mã lỗi (Error codes)

[Error Codes]

E_OK	Kết thúc sự thực thi bình thường
E_NOMEM	Thiếu bộ nhớ (vùng nhớ cho khối điều khiển hay của message buffer không được cấp phát)
E_LIMIT	Số lượng Message buffer vượt quá giới hạn của hệ thống
E_RSATR	Lỗi thuộc tính (mbfatr không có giá trị hoặc không thể được sử dụng)
E_PAR	Lỗi tham số (bk_cmbf là không có ý nghĩa hoặc bufsz và maxmsz là các số âm hay không có ý nghĩa)

[Description]

- Tạo một message buffer và gán cho nó một giá trị ID.
- Hàm system call này sẽ cấp phát một khối điều khiên cho việc tạo message buffer và dựa trên thông tin trong *bufsz* nó sẽ cấp phát cho một vùng nhớ để tạo buffer vòng.

- *exinf* là những thông tin được thiết lập một cách tự do bởi người dùng khi tạo ra message buffer và có thể được tham khảo bởi hàm *tk_ref_mbf* . Nếu như một vùng nhớ lớn hơn cần để lưu trữ những thông tin này hay những thông tin này có thể bị thay đổi khi message buffer được tạo ra thì nó có thể được cấp phát một vùng nhớ tách rời và địa chỉ bắt đầu khối nhớ đó sẽ được đưa vào trong tham số *exinf*.

- *mbfatr* dùng để lưu giữ thuộc tính của message buffer, trong đó những bit thấp dùng để lưu trữ những thuộc tính của hệ thống ,còn những bit cao để lưu giữ những thông tin mà phụ thuộc vào sự hiện thực. Thiết kế phần thuộc tính hệ thống của message buffer như sau :

mbfatr := (TA_TFIFO | | TA_TPRI) | [TA_NODISWAI]

TA_TFIFO	Task chờ để gởi message sẽ có thứ tự là FIFO
TA_TPRI	Task chờ để gởi message sẽ có thứ tự dựa trên độ ưu tiên của task
TA_NODISWAI	Hàm system call *tk_dis_wai* sẽ bị cấm

- Thứ tự của các task chờ để gởi message thì có thể là FIFO hoặc dựa trên độ ưu tiên của từng task nhưng thứ tự của các task chờ để nhận message chỉ có một thứ tự duy nhất đó là FIFO mà thôi.

```
#define   TA_TFIFO      0x00000000
#define   TA_TPRI       0x00000001
#define   TA_NODISWAI   0x00000080
```

2. tk_del_mbf : Xóa bỏ Message Buffer

[C Language Interface]

ER ercd = tk_del_mbf (ID mbfid) ;

[Parameters]

ID	mbfid	Message buffer ID

[Return Parameters]

ER	ercd	Mã lỗi

[Error Codes]

E_OK	Kết thúc sự thực thi bình thường
E_ID	ID không đúng (mbfid không có giá trị hoặc không được sử dụng)
E_NOEXS	Đối tượng này không tồn tại (message buffer mà được gán giá trị mbfid là không tồn tại)

[Description]

- Xóa message buffer có ID là *mbfid*.
- Khi hàm system call này được gọi thì nó sẽ giải phóng vùng nhớ dành cho khối điều khiển và không gian của messsage buffer tương ứng . Hàm này cũng sẽ kết thúc một cách bình thường nếu như ở đây vẫn còn các task đang chờ để gởi và nhận các message nhưng mã lỗi là E_DLT sẽ được trả về cho những task này và khi đó thì những message chưa được gởi cũng sẽ bị xoá.

3. tk_snd_mbf : Gửi một message đến Message Buffer

[C Language Interface]

ER ercd = tk_snd_mbf (ID mbfid, VP msg , INT msgsz, TMO tmout) ;

[Parameters]

ID	mbfid	Message buffer ID
INT	msgsz	Kích thước của message cần gởi
VP	msg	Địa chỉ bắt đầu của gói message cần gởi
TMO	tmout	Thời gian timeout

[Return Parameters]

ER	Mã lỗi (Error codes)

[Error Codes]

E_OK	Kết thúc sự thực thi bình thường
E_ID	ID không có giá trị (*mbfid* không đúng hoặc không được sử dụng)
E_NOEXS	Đối tượng không tồn tại (message buffer có *mbfid* không tồn tại)
E_PAR	Lỗi tham số (*msbsz* ≤ 0, *msgsz* < *maxmsz* , những giá trị mà không được sử dụng trong *msg*, hoặc *tmout* ≤ (-2))
E_DLT	Message buffer bị xoá trong lúc task đang chờ để gởi message.
E_RLWAI	Giải phóng khỏi trạng thái WAIT (hàm system call *tk_rel_wai* được đưa ra trong lúc task đang ở trạng thái WAIT)
E_DISWAI	Trạng thái WAIT sẽ được giải phóng vì task đang ở trạng thái mà nó không cho phép WAIT
E_TMOUT	Timeout
E_CTX	Lỗi ngữ cảnh (hàm này được triệu gọi từ những task nằm trong task-independence portion hay ở trạng thái không cho phép dispatching)

[Description]

- Gởi một messsage có địa chỉ bắt đầu là *msg* cho message buffer có ID là *mbfid*.
- Kích thước của message cần gởi đực chỉ ra trong thông số *msgsz*. Hàm system call này sẽ copy *msgsz* byte từ vị trí *msg* tới message buffer *mbfid*. Nếu *msgsz* lớn hơn *maxmsz* được thiết lập trong lúc tạo message buffer thì mã lỗi E_PAR sẽ được trả về. Nếu message buffer hiện tại không còn đủ để chứa message gởi tới thì task gởi message phải đi vào trạng thái WAIT cho đến khi có thể gởi được message cho message buffer. Các task đi vào trạng thái WAIT này sẽ được đưa vào hàng đợi và thứ tự của các task ở trong hàng đợi này là FIFO hay dựa trên độ ưu tiên là phụ thuộc vào thuộc tính được thiết lập cho message buffer lúc khởi tạo.
- Thời gian tối đa ở trong trạng thái WAIT có thể được thiết lập và nếu như thời gian tối đa này trôi qua mà task vẫn không gởi được message thì hàm system call này sẽ kết thúc , khi đó mã lỗi được trả về là E_TMOUT.
- Chỉ những giá trị dương được thiết lập cho *tmout*, đơn vị thời gian cơ bản cho timeout cũng giống như của hệ thống đó là 1 ms. Khi *tmout* được thiết lập là 0 (TMO_POL) thì có nghĩa là thời gian timeout sẽ là 0 và mã lỗi E_TMOUT sẽ được trả về mà không đi vào trạng thái WAIT nếu message buffe không còn đủ bộ nhớ. Khi TMO_FEVR = (-1) được thiết lập cho *tmout*, điều này có nghĩa là task có thời gian chờ là vô hạn và sẽ chờ

cho đến khi message buffer có đủ không gian để gởi message thì thôi. Một message có kích thước là 0 thì không được gởi. Nếu *msgsz* ≤ 0 thì mã lỗi là E_PAR sẽ được trả về.

- Khi hàm system call này được triệu gọi từ task-independence portion hoặc từ trạng thái mà không cho phép quá trình dispatch thì mã lỗi E_CTX sẽ được trả về , nhưng trong trường hợp *tmout* = TMO_POL thì điều này có thể được hiện thực.

4. tk_rcv_mbf : Nhận một message từ Message Buffer

[C Language Interface]

INT msgsz = tk_rcv_mbf (ID mbfid, VP msg , TMO tmout) ;

[Parameters]

ID	mbfid	Message buffer ID
VP	msg	Địa chỉ bắt đầu của gói message nhận được
TMO	tmout	Thời gian timeout

[Return Parameters]

ER	Mã lỗi (Error codes)

[Error Codes]

E_OK	Kết thúc sự thực thi bình thường
E_ID	ID không có giá trị (*mbfid* không đúng hoặc không được sử dụng)
E_NOEXS	Đối tượng không tồn tại (message buffer có *mbfid* không tồn tại)
E_PAR	Lỗi tham số (những giá trị mà không được sử dụng trong *msg*, hoặc *tmout* ≤ (-2))
E_DLT	Message buffer bị xoá trong lúc task đang chờ để gởi message.
E_RLWAI	Giải phóng khỏi trạng thái WAIT (hàm system call *tk_rel_wai* được đưa ra trong lúc task đang ở trạng thái WAIT)
E_DISWAI	Trạng thái WAIT sẽ được giải phóng vì task đang ở trạng thái mà nó không cho phép WAIT
E_TMOUT	Timeout
E_CTX	Lỗi ngữ cảnh (hàm này được triệu gọi từ những task nằm trong task-independence portion hay ở trạng thái không cho phép dispatching)

[Description]

- Nhận một message từ message buffer *mbfid* và lưu nó tại vị trí *msg*.
- Hàm system call này sẽ copy nội dung của message ở đầu queue được chứa trong message buffer *mbfid* vào bộ nhớ có kích thước là *msgsz* bắt đầu từ địa chỉ được chứa trong *msg* .
- Nếu không có message nào trong message buffer thì task gọi hàm system call này sẽ đi vào trạng thái WAIT và chờ cho đến khi có một message mới được gởi tới message buffer. Các task mà nhận message buffer được đưa vào hàng chờ chỉ theo một thứ tự duy nhất đó là FIFO.

- Thời gian tối đa ở trong trạng thái WAIT có thể được thiết lập và khi mà thời gian này đã trôi qua mà task vẫn chưa thoã mãn được những điều kiện để có thể đượcgiải phóng khỏi trạng thái WAIT thì hàm systemcall này sẽ kết thúc và mã lỗi trả về là E_TMOUT. Chỉ có những giá trị dương mới được thiết lập cho *tmout*.
- Khi mà TMO_POL=0 được thiết lập thì có nghĩa là E_TMO sẽ được trả về mà task không đi vào trạng thái WAIT ngay cả khi trong message buffer hiện tại không có message nào cả. Còn khi *tmout* được thiết lập với TMO_FEVER = (-1) thì task sẽ có thời gian chờ là vô hạn đến khi nào có message mới được gởi đến message buffer .

5. tk_ref_mbf : Tham khảo trạng thái của một Message Buffer

[C Language Interface]

ER ercd= tk_ref_mbf (ID mbfid, T_RMBF * pk_rmbf) ;

[Parameters]

ID	mbfid	Message buffer ID
T_RMBF *	pk_rmbf	Địa chỉ bắt đầu của gói thông tin trạng thái trả về

[Return Parameters]

ER	Mã lỗi (Error codes)

Chi tiết về *pk_rmbf*:

VP	exinf	Vùng thông tin mở rộng
ID	wtsk	Thông tin về task đang chờ đợi
ID	stsk	Thông tin về task gởi message
INT	msgsz	Kích thước của message sẽ được nhận tiếp theo
INT	frbufsz	Kích thước còn trống của buffer
INT	maxmsz	Kích thước tối đa của message

[Error Codes]

E_OK	Kết thúc sự thực thi bình thường
E_ID	ID không có giá trị (*mbfid* không đúng hoặc không được sử dụng)
E_NOEXS	Đối tượng không tồn tại (message buffer có *mbfid* không tồn tại)
E_PAR	Lỗi tham số (địa chỉ trả về của gói thông tin t trạng thái không thể sử dụng)

[Description]

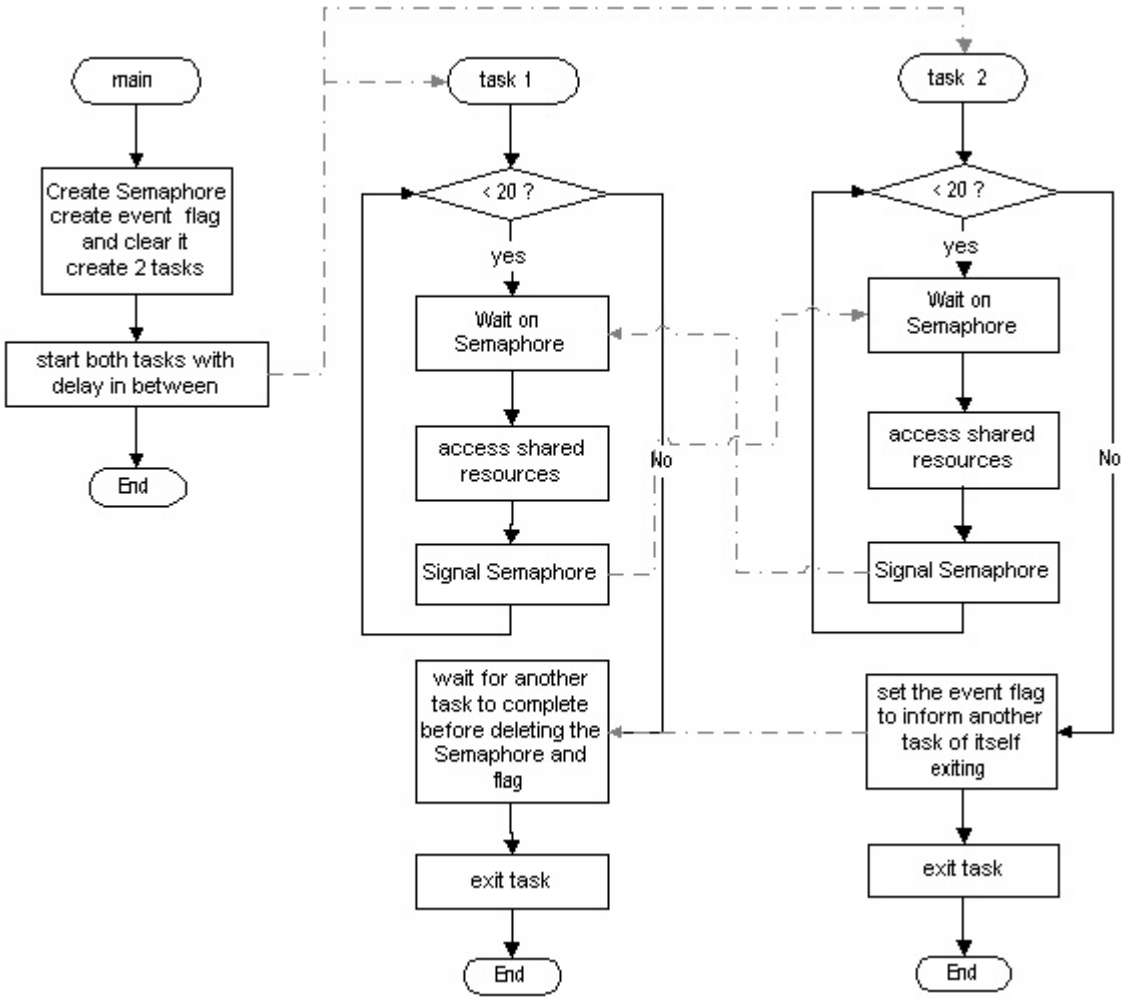
- Tham khảo đến trạng thái của message buffer có ID là *mbfid* . Thông số trả về bao gồm thông tin của task đang gởi message (stsk) , kích thước của message được nhận kế tiếp (msgsz) , kích thước còn trống của buffer (frbufsz) , kích thước tối đa của một message khi gởi (maxmsz) , thông tin về task đang chờ để nhận message(wtsk) và những thông tin mở rộng (exinf).
- ID của task đang chờ để gởi message tới messsage buffer được chỉ rõ trong thông số *stsk*.Nếu có nhiều task đang chờ để gởi message thì ID của task ở đầu hàng chờ sẽ được trả về, còn nếu như không có task nào đang chờ thì 0 sẽ được trả về. Nếu như *mbfid* là không tồn tại thì mã lỗi là E_NOEXS sẽ được trả về.

- Kích thước của message tiếp theo sẽ được nhận là *msgsz*. Nếu như không có message tiếp theo thì 0 sẽ được trả về. Một message mà có kích thước là 0 thì sẽ không thể được gởi.
- Thông số *frbufsz* chỉ ra kích thước còn trống của message buffer, giá trị của thông số này chỉ là giá trị xấp xỉ của message có thể gởi được mà thôi. m*axmsz* là giá trị tối đa của message có thể được gởi và giá trị của thông số này được thiết lập trong hàm *tk_cre_mbf*.

3. THỰC HÀNH

3.1 Semaphore :

Tạo một chương trình nhỏ sử dụng semaphore để đồng bộ giữa các task theo sơ đồ sau :



Đọc,hiểu và chạy chương trình đơn giản sau. Nắm rõ nguyên tắc để viết các ứng dụng sau này:

```
#include <basic.h>
#include <tk/tkernel.h>
#include <stdio.h>

IMPORT void task1(INT, VP);
ID taskid1 = -1;
ID taskid2 = -1;
```

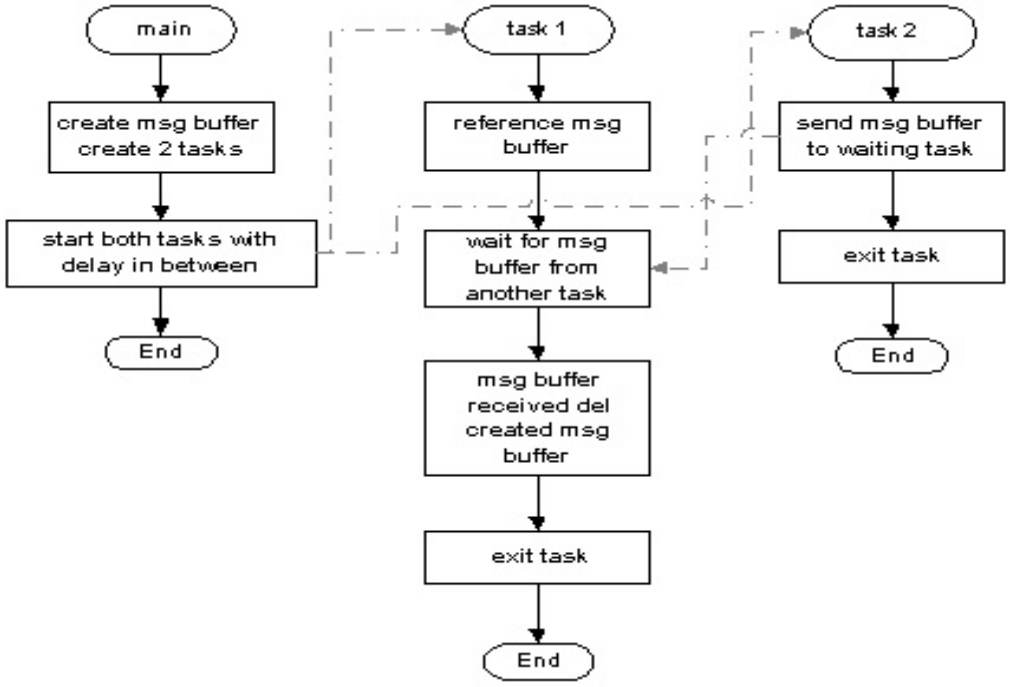
```
ID sem_id = -1;
ID flg_id = -1;

/*****************************************************************
task1 and task 2 trying to acquire resource
task 1 wait for task 2 before rel sem and event flg.
 *****************************************************************/
IMPORT void task1(INT stacd, VP exinf)
{
   int cnt = 0;
   ER ercd = 0;
   UINT p_flg;

   printf("This is task %d, going for 20 loop\n", taskid1);
   while (cnt < 20)
   {
     printf("task %d try get semaphore\n", taskid1);
     tk_wai_sem(sem_id, 1, TMO_FEVR);
printf("enter critical section task %d: cnt = %d\n", taskid1, cnt++);
     tk_dly_tsk(350);
     printf("delay task 1 over leave critical section\n");
     tk_sig_sem(sem_id, 1);
   }
   printf("wait for flag\n\n");
   ercd = tk_wai_flg(flg_id, 0x0001, TWF_ANDW | TWF_BITCLR, &p_flg,
TMO_FEVR);
   if (ercd < 0)
     printf("wait flag error %x", ercd);
   tk_del_sem(sem_id);
   tk_del_flg(flg_id);
   printf("task 1 exit and del task, flg, semaphore now\n");
   taskid1 = -1;
   sem_id = -1;
   flg_id = -1;
   tk_exd_tsk();
}



/*****************************************************************
task1 and task 2 trying to acquire resource
 *****************************************************************/
IMPORT void task2(INT stacd, VP exinf)
{
   int cnt = 0;
   ER ercd = 0;

   printf("\tThis is task %d, going for 20 loop\n", taskid2);
   while (cnt < 20)
   {
     printf("\ttask %d try get semaphore\n", taskid2);
     tk_wai_sem(sem_id, 1, TMO_FEVR);
     printf("\tenter critical section task %d: cnt = %d\n", taskid2,
cnt++);
     tk_dly_tsk(600);
     printf("\tdelay task 2 over leave critical section\n");
     tk_sig_sem(sem_id, 1);
   }
   ercd = tk_set_flg(flg_id, 0x0001);
```

```
   if (ercd < 0)
   {
     printf("\tset flag error %x", ercd);
   }
   printf("\ttask 2 exit and del task now\n");
   taskid2 = -1;
   tk_exd_tsk();
}


/*************************************************************************
*******
main
*************************************************************************
******/
EXPORT ER main( INT ac, UB *av[] )
{
 T_CTSK ctsk;
 T_CSEM sem;
 T_CFLG flg;
 ER ercd = 0;

 printf("main: (ac=%d)\n", ac);

 if (ac < 0)
 {
   if (taskid1 >= 0)
   {
     tk_ter_tsk(taskid1);
     tk_del_tsk(taskid1);
   }
   if (taskid2 >= 0)
   {
     tk_ter_tsk(taskid2);
     tk_del_tsk(taskid2);
   }
   goto ext;
 }
 sem.exinf = (VP)0x00000000;
 sem.sematr = TA_TFIFO | TA_FIRST;
 sem.isemcnt = 1;
 sem.maxsem = 1;
 sem_id = tk_cre_sem(&sem);
 printf("tk_cre_sem: (sem id = %d)\n", sem_id);
 if (sem_id < E_OK)
   goto ext;

 flg.exinf = (VP)0x00000000;
 flg.flgatr = TA_TFIFO | TA_FIRST;
 flg.iflgptn = 0x0000;
 flg_id = tk_cre_flg(&flg);
 printf("tk_cre_flg: (flg id = %d)\n", flg_id);
 if (flg_id < E_OK)
 {
   tk_del_sem(sem_id);
   goto ext;
 }
 ercd = tk_clr_flg(flg_id, 0x0000);
 if (ercd < 0)
```

```
{
   tk_del_sem(sem_id);
   goto ext;
}

ctsk.exinf =(VP)0x74736574;
ctsk.tskatr = TA_HLNG | TA_RNG0;
ctsk.task = task1;
ctsk.itskpri = 80;
ctsk.stksz = 1024 * 4;
taskid1 = tk_cre_tsk(&ctsk);
printf("tk_cre_tsk: (task1id = %d)\n", taskid1);
if (taskid1 < E_OK)
{
   tk_del_sem(sem_id);
   tk_del_flg(flg_id);
   goto ext;
}

ctsk.exinf = (VP)0x74736574;
ctsk.tskatr = TA_HLNG | TA_RNG0;
ctsk.task = task2;
ctsk.itskpri = 80;
ctsk.stksz = 1024 * 4;
taskid2 = tk_cre_tsk(&ctsk);
printf("tk_cre_tsk: (task2id = %d)\n", taskid2);
if (taskid2 < E_OK)
{
   tk_del_sem(sem_id);
   tk_del_flg(flg_id);
   tk_del_tsk(taskid1);
   goto ext;
}

printf("start tasks now\n");
tk_sta_tsk(taskid1, 0);
tk_dly_tsk(500);
tk_sta_tsk(taskid2, 0);

/*end*/
 ext:
printf("main ended\n\n");
return 0;
}
```

3.2 Message Buffer

Tạo một chương trình nhỏ sử dụng message buffer để đồng bộ giữa các task theo sơ đồ sau :



Thực hiện những bước tương tự như bài tập 1, với tên chương trình là *mbf_create*. Sau đây là nội dung file main.c

```
#include <basic.h>
#include <tk/tkernel.h>
#include <stdio.h>


IMPORT void task1(INT, VP);
ID taskid1 = -1;
ID taskid2 = -1;
ID mbf_id = -1;

/*************************************************************************
*******
task1
*************************************************************************
******/
IMPORT void task1(INT stacd, VP exinf)
{
 ER ercd = 0;
 T_RMBF mbf;
 UINT Msg;

 printf("This is task %d waiting for mbf\n", taskid1);

 ercd = tk_ref_mbf(mbf_id, &mbf);
 if (ercd < 0)
 {
   printf("ref mbf error %x\n", ercd);
   tk_del_mbf(mbf_id);
   goto exit;
 }
 printf("msg sz = %d free size = %d\n", mbf.msgsz, mbf.frbufsz);
 printf("wait for msg buf %d\n\n", mbf_id);
```

```
 ercd = tk_rcv_mbf(mbf_id, (VP)&Msg, TMO_FEVR);
 if (ercd < 0)
 {
   printf("rcv mbf error %x\n", ercd);
   tk_del_mbf(mbf_id);
   goto exit;
 }
 printf("data rcv = %x\n", Msg);
 ercd = tk_del_mbf(mbf_id);
 if (ercd < 0)
     printf("del mbf error %x\n", ercd);
exit:
 printf("task 1 exit and del task, msg buf now\n");
 taskid1 = -1;
 mbf_id = -1;
 tk_exd_tsk();
}


/************************************************************************
*******
task1 wakeup every 10 seconds and display print

************************************************************************
*****/
IMPORT void task2(INT stacd, VP exinf)
{
 UW buf = 0;
 ER ercd = 0;

 buf = 0xAA55AA55;
 printf("\ttask %d snd msg buf %d, data = %x\n", taskid2, mbf_id, buf);
 ercd = tk_snd_mbf(mbf_id, (VP)&buf, 4, TMO_FEVR);
 if (ercd < 0)
 {
   printf("\tsnd mbf fails error %x\n", ercd);
   tk_del_mbf(mbf_id);
 }
 printf("\ttask 2 exit and del task now\n");
 taskid2 = -1;
 tk_exd_tsk();
}


/************************************************************************
*******
main
************************************************************************
******/
EXPORT ER main( INT ac, UB *av[] )
{
 T_CTSK ctsk;
 T_CMBF _mbf;

 printf("main: (ac = %d)\n", ac);

 if (ac < 0)
 {
   if (taskid1 >= 0)
```

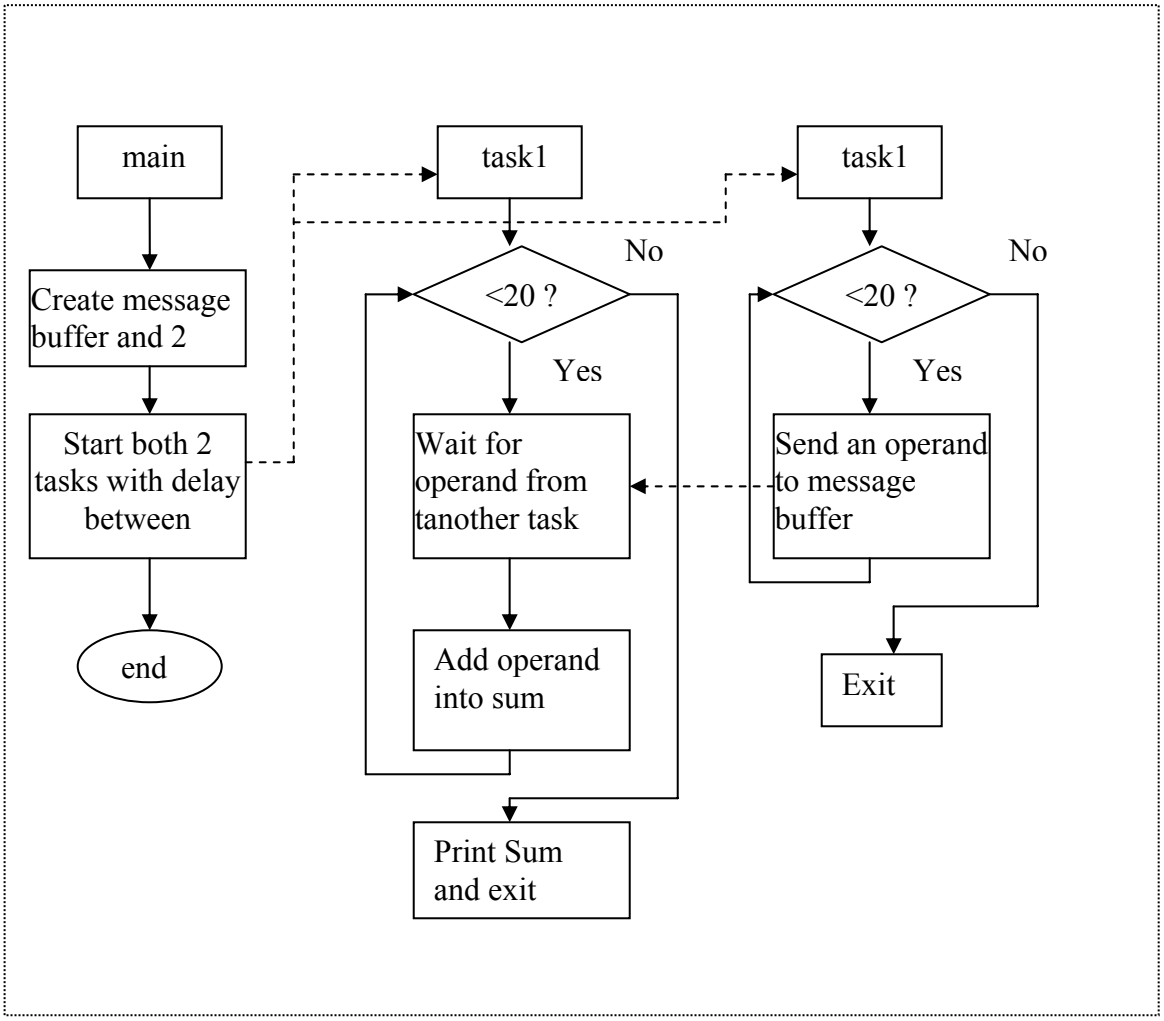
```
    {
      tk_ter_tsk(taskid1);
      tk_del_tsk(taskid1);
    }
    if (taskid2 >= 0)
    {
      tk_ter_tsk(taskid2);
      tk_del_tsk(taskid2);
    }
    goto ext;
  }
  _mbf.exinf = (VP)0x00000000;
  _mbf.mbfatr = TA_TFIFO;
  _mbf.bufsz = 0;
  _mbf.maxmsz = sizeof(UW);
  mbf_id = tk_cre_mbf(&_mbf);
  printf("tk_cre_mbf: (mbf id = %d)\n", mbf_id);
  if (mbf_id < E_OK)
  {
      printf("cre mbx fails = %x\n", mbf_id);
    goto ext;
  }

  ctsk.exinf =(VP)0x74736574;
  ctsk.tskatr = TA_HLNG | TA_RNG0;
  ctsk.task = task1;
  ctsk.itskpri = 80;
  ctsk.stksz = 1024 * 4;
  taskid1 = tk_cre_tsk(&ctsk);
  printf("tk_cre_tsk: (task1id = %d)\n", taskid1);
  if (taskid1 < E_OK)
  {
    tk_del_mbf(mbf_id);
          printf("cre tsk 1 fails = %x\n", taskid1);
    goto ext;
  }

  ctsk.exinf = (VP)0x74736574;
  ctsk.tskatr = TA_HLNG | TA_RNG0;
  ctsk.task = task2;
  ctsk.itskpri = 80;
  ctsk.stksz = 1024 * 4;
  taskid2 = tk_cre_tsk(&ctsk);
  printf("tk_cre_tsk: (task2id = %d)\n", taskid2);
  if (taskid2 < E_OK)
  {
    tk_del_mbf(mbf_id);
    tk_del_tsk(taskid1);
          printf("cre tsk 2 fails = %x\n", taskid2);
    goto ext;
  }

  printf("start tasks now\n");
  tk_sta_tsk(taskid1, 0);
  tk_dly_tsk(500);
  tk_sta_tsk(taskid2, 0);

  /*end*/
ext:
```

```
    printf("main ended\n\n");
    return 0;
}
```

3.3 Bài tập đề nghị :

Tạo một chương trình thực hiện các công việc sau : trong main tạo task 1, task2 và 1 message buffer, task 1 thực hiện tính tổng của 20 số nguyên do task 2 gửi thông qua message buffer.

Chương trình có sơ đồ sau :



BÀI THỰC HÀNH SỐ 7

MEMORY POOL

1. MỤC TIÊU

- Giúp sinh viên hiểu rõ và biết cách tạo và sử dụng các đối tượng được dùng làm vùng nhớ chia sẽ giữa các task, tên là memory pool.
- Sử dụng Message buffer để truyền địa chỉ của vùng nhớ chia sẽ giữa các task.

2. LÝ THUYẾT

Những hàm quản lý memory pool cung cấp cơ chế quản lý ở cấp phần mềm các memory pool và sự cấp phát bộ nhớ.

Có hai loại memory pool là memory pool có kích thước cố định (*fixed-size memory pool*) và memory pool có kích thước không cố định (tức là có thể thay đổi được kích thước *variable-size memory pool*), chúng được xem xét độc lập và có các hàm system call riêng biệt nhau cho hoạt động của chúng. Tất cả các khối bộ nhớ được cấp phát từ fixed-size memory pool thì đều cùng một kích thước, ngược lại nếu chúng được cấp phát từ variable-size memory pool thì có thể có những kích thước khác nhau.

Tất cả bộ nhớ được quản lý bởi các hàm quản lý memory pool thì đều nằm trong không gian hệ thống, T-Kernel không cung cấp các hàm quản lý không gian bộ nhớ của task.

2.1 Fixed-size Memory pool

Mỗi Fixed-size memory pool là một đối tượng dùng để quản lý động các khối bộ nhớ có kích thước cố định.). OS cung cấp các hàm system call gồm tạo hay hủy bỏ một fixed-size memory pool, lấy và trả các khối bộ nhớ từ một fixed-size memory pool, và tham khảo trạng thái của fixed-size memory pool. Mỗi fixed-size memory pool được định danh bằng một ID gọi là fixed-size memory pool ID.

Mỗi fixed-size memory pool có một không gian bộ nhớ được gọi đơn giản là vùng memory pool, và một hàng đợi các task đang chờ xin cấp phát bộ nhớ. Một task đợi lấy một khối bộ nhớ từ một fixed-size memory pool, nếu fixed-size memory pool đó không đủ vùng nhớ để cấp phát thì task đó sẽ đi vào trạng thái WAIT và được đặt trong hàng đợi các task của fixed-size memory pool đó.

❖ **Các hàm system call của Fixed-size Memory pool :**

1. tk_cre_mpf : Tạo một fixed-size memory pool

[C Language Interface]

```
ID mpfid = tk_cre_mpf ( T_CMPF * pk_cmpf ) ;
```

[Parameters]

T_CMPF * pk_cmpf Thông tin về fixed-size memory pool được tạo

Chi tiết của pk_cmpf :

VP exinf Thông tin phụ được thêm vào
ATR mpfatr Các thuộc tính của memory pool

INT mpfcnt Số lượng khối bộ nhớ
INT blfsz Kích thước của một khối bộ nhớ (tính bằng byte)

[Return Parameters]

ID	mpfid	Fixed-size memory pool ID
hoặc	Mã lỗi (Error codes)	

[Error Codes]

E_OK	Kết thúc bình thường
E_NOMEM	Thiếu bộ nhớ (vùng nhớ cho khối điều khiển hay vùng memory pool không được cấp phát)
E_LIMIT	Số lượng fixed-size memory pool vượt quá giới hạn của hệ thống
E_RSATR	Lỗi thuộc tính (*mpfatr* không hợp lệ hay không sử dụng được)
E_PAR	Lỗi tham số (*pk_cmpf* không hợp lệ)

[Description]

- Tạo một fixed-size memory pool và gán cho nó một fixed-size memory pool ID.
- Hàm system call này định vị một không gian bộ nhớ để sử dụng như một memory pool dựa trên thông tin được chỉ định trong các tham số *mpfcnt* và *blfsz*, và gán một khối điều khiển cho memory pool đó. Một khối bộ nhớ có kích thước *blfsz* có thể được cấp phát từ một memory pool được tạo bằng cách gọi hàm system call *tk_get_mpf*.
- *exinf* có thể được sử dụng tự do bởi người dùng để chèn thêm thông tin về fixed-size memory pool, và được tham khảo bằng hàm *tk_ref_mpf*. Nếu thông tin của fixed-size memory pool là một vùng lớn, ứng dụng phải xin cấp phát vùng nhớ riêng biệc và đặt địa chỉ vào *exinf*.
- *mbfatr* cho biết các thuộc tính hệ thống trong các bit thấp của nó và thông tin hiện thực trong các bit cao.

```
mbxatr := (TA_TFIFO | | TA_TPRI) | [TA_NODISWAI
          | (TA_RNG0 | | TA_RNG1 | | TA_RNG2 | | TA_RNG3)
```

TA_TFIFO	Các task đợi trong hàng đợi theo thứ tự FIFO
TA_TPRI	Các task đợi trong hàng đợi theo độ ưu tiên
TA_RNG*n*	Đặc quyền truy cập bộ nhớ được thiết lập ở cấp bảo vệ *n*
TA _NODISWAI	Cấm làm vô hiệu trạng thái đợi bởi hàm *tk_dis_wai*

```
#define  TA_TFIFO          0x00000000
#define  TA_TPRI           0x00000001
#define  TA_NODISWAI       0x00000080
#define  TA_RNG0           0x00000000 /* protection level 0 */
#define  TA_RNG1           0x00000100 /* protection level 1 */
#define  TA_RNG2           0x00000200 /* protection level 2 */
```

```
#define   TA_RNG3          0x00000300 /* protection level 3 */
```

- TA_RNG*n* được chỉ định để giới hạn mức độ truy cập bộ nhớ. Chỉ có những task thực thi ở cấp bảo vệ bằng hoặc cao hơn cấp bảo vệ của fixed-size memory pool thì có thể truy cập bộ nhớ được cấp phát. Ví dụ bộ nhớ được cấp phát từ một memory pool có cấp bảo vệ TA_RNG1 thì chỉ có những task chạy ở cấp TA_RNG0 hay TA_RNG1 mới truy cập được vùng nhớ trên. Memory pool được tạo thường trú trong vùng nhớ của không gian hệ thống.T-Kernel không cung cấp những hàm cho việc tạo memory pool trong không gian của task.

2. tk_del_mpf : Xóa bỏ một fixed-size memorypool

[C Language Interface]

```
ER ercd  =  tk_del_mpf ( ID  mpfid ) ;
```

[Parameters]

ID	mpfid	Memory pool ID

[Return Parameters]

ER	ercd	Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*mpfid* không hợp lệ hay không được sử dụng
E_NOEXS	Variable-size memory pool được chỉ định trong *mpfid* không tồn tại

[Description]

- Loại bỏ một fixed-size memory pool có ID là *mpfid* .
- Không có sự kiểm tra hay thông báo nào được thực hiện cho dù có hay không có các task đang sử dụng các khối bộ nhớ của memory pool. Hàm system call này vẫn kết thúc bình thường ngay cả khi tất cả các khối nhớ được trả về hoàn toàn cho memory pool.
- Hàm system call này giải phóng memory pool ID tương ứng, không gian bộ nhớ khối điều khiển nó, và cả chính nó.
- Hàm này kết thúc bình thường ngay cả khi có một task đang đợi được cấp phát bộ nhớ, nhưng mã lỗi E_DLT được trả về cho task đó.

3. tk_get_mpf : Xin cấp một block bộ nhớ từ memory pool

[C Language Interface]

```
ER ercd = tk_get_mpf  (ID  mpfid,  VP *  p_blf,  TMO  tmout ) ;
```

[Parameters]

ID	mpfid	Fixed-size memory pool ID
TMO	tmout	Thời gian timeout

[Return Parameters]

ER	ercd	Mã lỗi (Error codes)
VP	blf	Địa chỉ bắt đầu của khối bộ nhớ

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*mpfid* không hợp lệ hay không được sử dụng
E_PAR	Lỗi tham số (*tmout* ≤ (2))
E_NOEXS	Fixed-size memory pool được chỉ định trong *mpfid* không tồn tại
E_DLT	Đối tượng được đợi bị xoá (fixed-size memory poolchỉ định bị xoá khi task đang đợi)
E_TMOUT	Thăm dò bị lỗi hay timeout
E_RLWAI	Trạng thái WAIT của Task được giải phóng (do Task nhận được *tk_rel_wai* khi đang ở trạng thái WAIT)
E_DISWAI	Trạng thái WAIT của Task được giải phóng bởi trạng thái WAIT của Task bị vô hiệu
E_CTX	Lỗi ngữ cảnh (hàm được gọi từ một *Task độc lập với ngữ cảnh* hay *dispatch* bị vô hiệu)

[Description]

- Nhận một khối bộ nhớ từ một fixed-size memory pool có ID là *mpfid*.Địa chỉ của khối bộ nhớ được cấp phát trả về trong *blf*. Kích thước khối bội nhớ được chỉ định trong tham số *blfsz* khi fixed-size memory pool được tạo. Bộ nhớ được cấp phát sẽ không bị xóa về 0, do vậy nội dung của nó là không xác định.
- Nếu một khối không được cấp phát từ fixed-size memory pool chỉ định thì task sẽ đi vào trạng thái WAIT và được xếp hàng để đợi được cấp phát.
- Một thời gian đợi tối đa (timeout) có thể được thiết lập trong *tmout*. Nếu bị timeout trước khi điều kiện giải phóng đợi thỏa mãn (không có sẵn bộ nhớ), thì hàm system call này sẽ kết thúc, và trả về mã lỗi E_TMOUT. Chỉ có giá trị dương mới được gán cho *tmout*. Đơn vị của timeout là *ms*. Khi TMO_POL =0 được gán cho *tmout*, thì mã lỗi E_TMOUT được trả về mà không vào trạng thái WAIT ngay cả khi không có sự cấp phát bộ nhớ. Khi TMO_FEVR = -1 được gán cho *tmout*, thì task sẽ đợi cho tới khi lấy được cấp phát bộ nhớ mà không bị timeout.

4. tk_rel_mpf : Trả một block nhớ cho memory pool

[C Language Interface]

```
ER ercd = tk_rel_mpf ( ID   mpfid, VP  blf  ) ;
```

[Parameters]

ID	mpfid	Fixed-size memory pool ID
VP	blf	Địa chỉ bắt đầu của khối bộ nhớ

[Return Parameters]

ER	ercd	Mã lỗi

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*mpfid* không hợp lệ hay không được sử dụng
E_NOEXS	Fixed-size memory pool được chỉ định trong *mpfid* không tồn tại
E_PAR	Lỗi tham số (*blf* không hợp lệ, hoặc khối bộ nhớ trả về sai memory pool)

[Description]

- Trả một khối bộ nhớ *blf* về cho một fixed-size memory pool được chỉ định trong *mpfid*.
- Khi một khối bộ nhớ được trả về cho một fixed-size memory pool thì nó phải là fixed-size memory pool đã cấp phát khối bộ nhớ đó, nếu không lỗi E_PAR sẽ được trả về.

5. tk_ref_mpf : Tham khao thông tin của memory pool

[C Language Interface]

ER ercd = tk_ref_mpf (ID mpfid, T_RMPF * pk_rmpf) ;

[Parameters]

ID	mpfid	Fixed-size memory pool ID
T_RMPF*	pk_rmpf	Địa chỉ của gói thông tin về trạng thái fixed-size memory pool

Chi tiết pk_rmpf :

VP	exinf	Thông tin mở rộng
ID	wtsk	Thông tin task đang đợi
INT	frbcnt	Số lượng khối bộ nhớ trống

(Những tham số khác tùy theo hiện thực có thêm vào sau)

[Return Parameters]

ER	ercd	Mã lỗi

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*mpfid* không hợp lệ hay không được sử dụng
E_NOEXS	Fixed-size memory pool được chỉ định trong *mpfid* không tồn tại
E_PAR	Lỗi tham số (địa chỉ của gói thông tin trả về không dùng được)

[Description]

- Tham khảo trạng thái của fixed-size memory pool có ID là *mpfid*, truyền trong các thông số trả về số lượng khối bộ nhớ trống, thông tin task đang đợi event flag (*wtsk*), và thông tin mở rộng (*exinf*).

- Tham số *wtsk* cho biết ID của một task đang đợi được cấp phát. Nếu có nhiều hơn một task đang đợi thì ID của task đứng đầu hàng đợi sẽ được trả về. Nếu không có task nào thì *wtsk* = 0 được trả về.
- Nếu fixed-size memory pool tương ứng không tồn tại, mã lỗi E_NOEXS được trả về.Nếu không còn khối nhớ nào trống trong memory pool thì *frbcnt* = 0.

2.2 Variable-size Memory pool

Mỗi variable-size memory pool là một đối tượng dùng để quản lý động các khối bộ nhớ có kích thước thước bất kỳ. OS cung cấp các hàm system call gồm tạo hay hủy bỏ một variable-size memory pool, lấy và trả các khối bộ nhớ từ một variable-size memory pool, và tham khảo trạng thái của variable-size memory pool. Mỗi variable-size memory pool được định danh bằng một ID gọi là variable-size memory pool ID.

Mỗi variable-size memory pool có một không gian bộ nhớ được gọi đơn giản là vùng memory pool, và một hàng đợi các task đang chờ xin cấp phát bộ nhớ. Một task đợi lấy một khối bộ nhớ từ một variable-size memory pool, nếu variable-size memory pool đó không đủ vùng nhớ để cấp phát thì task đó sẽ đi vào trạng thái WAIT và được đặt trong hàng đợi các task của variable-size memory pool đó.

❖ **Các hàm system call của Variable-size Memory pool :**

1. tk_cre_mpl : Tạo một variable-size memory pool

[C Language Interface]

```
ID mplid = tk_cre_mpl ( T_CMPL * pk_cmpl ) ;
```

[Parameters]

T_CMPL * pk_cmpl Thông tin về variable-size memory pool được tạo

Chi tiết của pk_cmpl :

VP	exinf	Thông tin phụ được thêm vào
ATR	mplatr	Các thuộc tính của memory pool
INT	mplsz	Kích thước của variable-size memory pool

[Return Parameters]

ID	mplid	Variable-size memory pool ID
hoặc	Mã lỗi (Error codes)	

[Error Codes]

E_OK	Kết thúc bình thường
E_NOMEM	Thiếu bộ nhớ (vùng nhớ cho khối điều khiển hay vùng memory pool không được cấp phát)
E_LIMIT	Số lượng variable-size memory pool vượt quá giới hạn của hệ thống
E_RSATR	Lỗi thuộc tính (*mplatr* không hợp lệ hay không sử dụng được)
E_PAR	Lỗi tham số (*pk_cmpl* không hợp lệ)

[Description]

- Tạo một variable-size memory pool và gán cho nó một variable-size memory pool ID.
- Hàm system call này định vị một không gian bộ nhớ để sử dụng như một memory pool dựa trên thông tin được chỉ định trong tham số *mplsz*, và gán một khối điều khiển cho memory pool đó.
- *exinf* có thể được sử dụng tự do bởi người dùng để chèn thêm thông tin về variable-size memory pool, và được tham khảo bằng hàm *tk_ref_mpl*. Nếu thông tin của variable-size memory pool là một vùng lớn, ứng dụng phải xin cấp phát vùng nhớ riêng biệc và đặt địa chỉ vào *exinf*.
- *mblatr* cho biết các thuộc tính hệ thống trong các bit thấp của nó và thông tin hiện thực trong các bit cao.

```
mblatr := (TA_TFIFO | | TA_TPRI) | [TA_NODISWAI
          | (TA_RNG0 | | TA_RNG1 | | TA_RNG2 | | TA_RNG3)
```

TA_TFIFO	Các task đợi trong hàng đợi theo thứ tự FIFO
TA_TPRI	Các task đợi trong hàng đợi theo độ ưu tiên
TA_RNG*n*	Đặc quyền truy cập bộ nhớ được thiết lập ở cấp bảo vệ *n*
TA _NODISWAI	Cấm làm vô hiệu trạng thái đợi bởi hàm *tk_dis_wai*

```
#define   TA_TFIFO          0x00000000
#define   TA_TPRI           0x00000001
#define   TA_NODISWAI       0x00000080
#define   TA_RNG0           0x00000000 /* protection level 0 */
#define   TA_RNG1           0x00000100 /* protection level 1 */
#define   TA_RNG2           0x00000200 /* protection level 2 */
#define   TA_RNG3           0x00000300 /* protection level 3 */
```

- Bộ nhớ được cấp phát cho các task theo thứ tự của hàng đợi ngay cả khi những task khác có yêu cầu ít hơn task đứng đầu hàng đợi.
- TA_RNG*n* được chỉ định để giới hạn mức độ truy cập bộ nhớ. Chỉ có những task thực thi ở cấp bảo vệ bằng hoặc cao hơn cấp bảo vệ của fixed-size memory pool thì có thể truy cập bộ nhớ được cấp phát. Ví dụ bộ nhớ được cấp phát từ một memory pool có cấp bảo vệ TA_RNG1 thì chỉ có những task chạy ở cấp TA_RNG0 hay TA_RNG1 mới truy cập được vùng nhớ trên. Memory pool được tạo thường trú trong vùng nhớ của không gian hệ thống.T-Kernel không cung cấp những hàm cho việc tạo memory pool trong không gian của task.

2. tk_del_mpl : Xóa bỏ một variable-size memorypool

[C Language Interface]

```
ER ercd = tk_del_mpl ( ID mplid ) ;
```

[Parameters]

ID	mplid	Memory pool ID

[Return Parameters]

ER	ercd	Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*mplid* không hợp lệ hay không được sử dụng
E_NOEXS	Variable-size memory pool được chỉ định trong *mplid* không tồn tại

[Description]

- Loại bỏ một variable-size memory pool có ID là *mplid* .
- Không có sự kiểm tra hay thông báo nào được thực hiện cho dù có hay không có các task đang sử dụng các khối bộ nhớ của memory pool. Hàm system call này vẫn kết thúc bình thường ngay cả khi tất cả các khối nhớ được trả về hoàn toàn cho memory pool.
- Hàm system call này giải phóng memory pool ID tương ứng, không gian bộ nhớ khối điều khiển nó, và cả chính nó.
- Hàm này kết thúc bình thường ngay cả khi có một task đang đợi được cấp phát bộ nhớ, nhưng mã lỗi E_DLT được trả về cho task đó.

3. tk_get_mpl : Xin cấp bộ nhớ từ memory pool

[C Language Interface]

ER ercd = tk_get_mpl (ID mplid, INT blksz, VP * p_blk, TMO tmout) ;

[Parameters]

ID	mpfid	Fixed-size memory pool ID
INT	blksz	Kích thước khối bộ nhớ xin cấp phát (tính bằng byte)
TMO	tmout	Thời gian timeout

[Return Parameters]

ER	ercd	Mã lỗi (Error codes)
VP	blk	Địa chỉ bắt đầu của khối bộ nhớ

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*mplid* không hợp lệ hay không được sử dụng
E_PAR	Lỗi tham số (*tmout* ≤ (2))
E_NOEXS	Variable-size memory pool được chỉ định trong *mplid* không tồn tại
E_DLT	Đối tượng được đợi bị xoá (variable-size memory poolchỉ định bị xoá khi task đang đợi)
E_TMOUT	Thăm dò bị lỗi hay timeout

E_RLWAI	Trạng thái WAIT của Task được giải phóng (do Task nhận được *tk_rel_wai* khi đang ở trạng thái WAIT)
E_DISWAI	Trạng thái WAIT của Task được giải phóng bởi trạng thái WAIT của Task bị vô hiệu
E_CTX	Lỗi ngữ cảnh (hàm được gọi từ một *Task độc lập với ngữ cảnh* hay *dispatch* bị vô hiệu)

[Description]

- Nhận một khối bộ nhớ có kích thước *blksz* từ một variable-size memory pool có ID là *mplid*. Địa chỉ của khối bộ nhớ được cấp phát trả về trong *blk*. Bộ nhớ được cấp phát sẽ không bị xóa về 0, do vậy nội dung của nó là không xác định.
- Nếu một khối không được cấp phát từ variable-size memory pool chỉ định thì task sẽ đi vào trạng thái WAIT và được xếp hàng để đợi được cấp phát.
- Một thời gian đợi tối đa (timeout) có thể được thiết lập trong *tmout*. Nếu bị timeout trước khi điều kiện giải phóng đợi thỏa mãn (không có sẵn bộ nhớ), thì hàm system call này sẽ kết thúc, và trả về mã lỗi E_TMOUT. Chỉ có giá trị dương mới được gán cho *tmout*. Đơn vị của timeout là *ms*. Khi TMO_POL =0 được gán cho *tmout*, thì mã lỗi E_TMOUT được trả về mà không vào trạng thái WAIT ngay cả khi không có sự cấp phát bộ nhớ. Khi TMO_FEVR = -1 được gán cho *tmout*, thì task sẽ đợi cho tới khi lấy được cấp phát bộ nhớ mà không bị timeout.

4. tk_rel_mpf : Trả vùng nhớ cho memory pool

[C Language Interface]

ER ercd = tk_rel_mpl (ID mplid, VP blk) ;

[Parameters]

ID	mplid	Variable-size memory pool ID
VP	blk	Địa chỉ bắt đầu của khối bộ nhớ

[Return Parameters]

ER	ercd	Mã lỗi

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*mplid* không hợp lệ hay không được sử dụng
E_NOEXS	Variable-size memory pool được chỉ định trong *mplid* không tồn tại
E_PAR	Lỗi tham số (*blk* không hợp lệ, hoặc khối bộ nhớ trả về sai memory pool)

[Description]

- Trả một khối bộ nhớ *blk* về cho một variable-size memory pool được chỉ định trong *mplid*.

- Khi một khối bộ nhớ được trả về cho một variable-size memory pool thì nó phải là variable-size memory pool đã cấp phát khối bộ nhớ đó, nếu không lỗi E_PAR sẽ được trả về.

5. tk_ref_mpl : Tham khao thông tin của memory pool

[C Language Interface]

ER ercd = tk_ref_mpl (ID mplid, T_RMPF * pk_rmpl) ;

[Parameters]

ID	mplid	Variable-size memory pool ID
T_RMPL*	pk_rmpl	Địa chỉ của gói thông tin về trạng thái variable-size memory pool

Chi tiết pk_rmpl :

VP	exinf	Thông tin mở rộng
ID	wtsk	Thông tin task đang đợi
INT	frsz	Kích thước bộ nhớ trống (bytes)
INT	maxsz	Kích thước tối đa của memory pool (bytes)

(Những tham số khác tùy theo hiện thực có thêm vào sau)

[Return Parameters]

ER	ercd	Mã lỗi

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*mplid* không hợp lệ hay không được sử dụng
E_NOEXS	Variable-size memory pool được chỉ định trong *mplid* không tồn tại
E_PAR	Lỗi tham số (địa chỉ của gói thông tin trả về không dùng được)

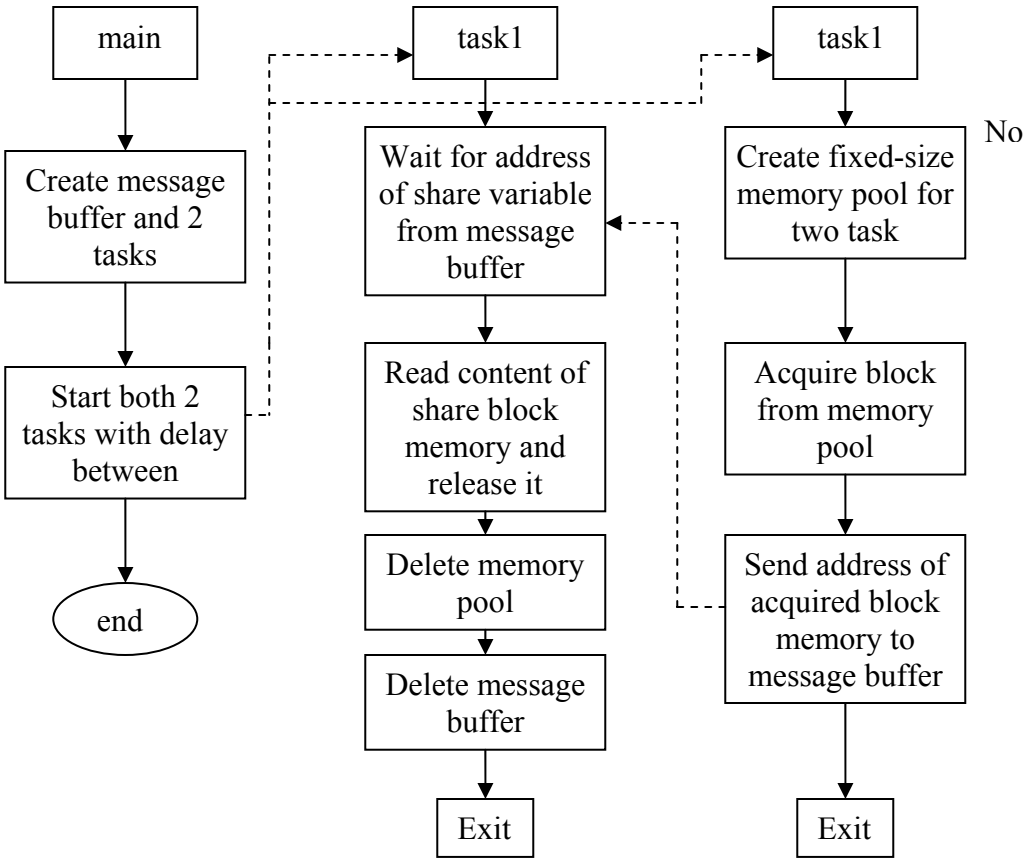
[Description]

- Tham khảo trạng thái của variable-size memory pool có ID là *mpfid*, truyền trong các thông số trả về kích thước bộ nhớ trống *frsz*, kích thước tối đa của memory pool *maxsz,* thông tin task đang đợi event flag (*wtsk*), và thông tin mở rộng (*exinf*).
- Tham số *wtsk* cho biết ID của một task đang đợi được cấp phát. Nếu có nhiều hơn một task đang đợi thì ID của task đứng đầu hàng đợi sẽ được trả về. Nếu không có task nào thì *wtsk* = 0 được trả về.
- Nếu variable-size memory pool tương ứng không tồn tại, mã lỗi E_NOEXS được trả về.

3. THỰC HÀNH

3.1 Fixed-size Memory Pool

Nội dung của bài này là tìm hiểu cách tạo và cơ chế xin các block nhớ từ fixed-size memory pool, sự truyền nhận địa chỉ thông qua message buffer, và cách lấy nội dung từ vùng nhớ chia sẽ trên memory pool từ các task khác nhau. Chương trình có sơ đồ như sau :



Cách tạo chương trinh tương tự như những bài trước, với tên chương trình này là *mpf_create*, viết nội dung sau vào main.c. Sau đó biên dịch, thực thi và giải thích các thông số trong chương trình.

```
#include <basic.h>
#include <tk/tkernel.h>
#include <stdio.h>

IMPORT void task1(INT, VP);
ID taskid1 = -1;
ID taskid2 = -1;
ID mbf_id = -1;
ID mem_id = -1;
VP buf = 0;

/*************************************************************************
**********
```

```
task1
************************************************************************
*********/
IMPORT void task1(INT stacd, VP exinf)
{
  ER ercd = 0;
  UW p;

  printf("This is task %d waiting for mbf\n", taskid1);

  printf("wait for message buffer %d\n\n", mbf_id);
  ercd = tk_rcv_mbf(mbf_id, (VP)&p, TMO_FEVR);
  if (ercd < 0)
  {
    printf("rcv mbf error %x\n", ercd);
    goto exit;
  }
  printf("task %d rcv mbf data = %x\n\n", taskid1, *((UW*)p));
  ercd = tk_rel_mpf(mem_id, (VP)p);
  if (ercd < 0)
  {
       printf("rel mpf error %x\n", ercd);
    tk_del_mbf(mbf_id);
    tk_del_mpf(mem_id);
    goto exit;
  }
  ercd = tk_del_mpf(mem_id);
  if (ercd < 0)
  {
      printf("del mpf error %x\n", ercd);
    tk_del_mbf(mbf_id);
    goto exit;
  }
  ercd = tk_del_mbf(mbf_id);
  if (ercd < 0)
      printf("del mbf error %x\n", ercd);
exit:
  printf("task 1 exit and del task, mbf, mpf now\n");
  mem_id = -1;
  taskid1 = -1;
  mbf_id = -1;
  tk_exd_tsk();
}



/************************************************************************
**********
task1 wakeup every 10 seconds and display print

************************************************************************
********/
IMPORT void task2(INT stacd, VP exinf)
{
  T_CMPF mempool;
  ER ercd = 0;

  mempool.exinf = (VP)0x00000000;
  mempool.mpfatr = TA_TFIFO | TA_RNG0;
```

```
  mempool.mpfcnt = (INT)32;
  mempool.blfsz = (INT)4;
  mem_id = tk_cre_mpf(&mempool);
  if (mem_id < 0)
  {
    printf("\tcre var mem pool fails error %x\n", mem_id);
    goto exit;
  }
  ercd = tk_get_mpf(mem_id, (VP *)&buf, TMO_FEVR);
  if (ercd < 0)
  {
    printf("\tget var mem pool fails error %x\n", ercd);
    tk_del_mpf(mem_id);
    goto exit;
  }
  printf("\taddr = %x\n", buf);
  *((UW *)buf) = 0xAA55AA55;
  printf("\ttask %d snd mbf, data = %x\n", taskid2, *((UW *)buf));
  ercd = tk_snd_mbf(mbf_id, (VP)&buf,sizeof(UW),TMO_FEVR);
  if (ercd < 0)
  {
    printf("\tsnd mbf fails error %x\n", ercd);
    tk_rel_mpf(mem_id, (VP)&buf);
    tk_del_mpf(mem_id);
    tk_del_mbf(mbf_id);
  }
exit:
  printf("\ttask 2 exit and del task now\n");
  taskid2 = -1;
  tk_exd_tsk();
}


/*********************************************************************
**********
main
*********************************************************************
*********/
EXPORT ER main( INT ac, UB *av[] )
{
  T_CTSK ctsk;
  T_CMBF mbf;

  printf("main: (ac = %d)\n", ac);

  if (ac < 0)
  {
    if (taskid1 >= 0)
    {
      tk_ter_tsk(taskid1);
      tk_del_tsk(taskid1);
    }
    if (taskid2 >= 0)
    {
      tk_ter_tsk(taskid2);
      tk_del_tsk(taskid2);
    }
    goto ext;
  }
```

```
  mbf.exinf = (VP)0x00000000;
  mbf.mbfatr = TA_TFIFO;
  mbf.bufsz = 0;
  mbf.maxmsz = sizeof(UW);
  mbf_id = tk_cre_mbf(&mbf);
  printf("tk_cre_mbf: (mbf id = %d)\n", mbf_id);
  if (mbf_id < E_OK)
  {
          printf("cre mbf fails = %x\n", mbf_id);
    goto ext;
  }

  ctsk.exinf =(VP)0x74736574;
  ctsk.tskatr = TA_HLNG | TA_RNG0;
  ctsk.task = task1;
  ctsk.itskpri = 80;
  ctsk.stksz = 1024 * 4;
  taskid1 = tk_cre_tsk(&ctsk);
  printf("tk_cre_tsk: (task1id = %d)\n", taskid1);
  if (taskid1 < E_OK)
  {
    tk_del_mbf(mbf_id);
          printf("cre tsk 1 fails = %x\n", taskid1);
    goto ext;
  }

  ctsk.exinf = (VP)0x74736574;
  ctsk.tskatr = TA_HLNG | TA_RNG0;
  ctsk.task = task2;
  ctsk.itskpri = 80;
  ctsk.stksz = 1024 * 4;
  taskid2 = tk_cre_tsk(&ctsk);
  printf("tk_cre_tsk: (task2id = %d)\n", taskid2);
  if (taskid2 < E_OK)
  {
    tk_del_mbf(mbf_id);
    tk_del_tsk(taskid1);
          printf("cre tsk 2 fails = %x\n", taskid2);
    goto ext;
  }

  printf("start tasks now\n");
  tk_sta_tsk(taskid1, 0);
  tk_dly_tsk(500);
  tk_sta_tsk(taskid2, 0);

  /*end*/
ext:
  printf("main ended\n\n");
  return 0;
}
```

3.2 Variale-size Memory Pool

Trong phần này sinh viên sẽ tự viết một chương trình tạo và sử dụng variable size memory pool. Nội dung tương tự như bài trên, nhưng đối với variable size memory pool, ta có thể xin cấp phát khối nhớ cho một struct bất kì làm đối tượng chia sẽ. Cấu trúc chương trình tương tự, chỉ thay đổi một số lệnh để hù hợp với memory pool.

Ví dụ ta có thể định nghĩa một stuct bất kì :

```
typedef struct {
     UH   x;
     UH   y;
}_Point;
```

Cơ chế truyền địa chỉ tương tự, ví dụ biến *p* (UW) chứa địa chỉ của struct trong memry pool, ta có thể truy cập nội dung của struct thông qua các phep ép kiểu con trỏ :

`(*((_Point *)p)).x` //thành phần x của struct _Point có địa chỉ trong memory pool là *p*.

Dùng hàm sizeof(*struct_name*) để lấy kích thước của struct đó.

BÀI THỰC HÀNH SỐ 8

QUẢN LÝ THỜI GIAN

1. MỤC TIÊU

- Hiểu được các hàm quản lý thời gian hệ thống.
- Sử dụng được cyclic handler và alarm handler phục vụ cho các chương trình sau này.

2. LÝ THUYẾT

Những hàm quản lý thời gian dùng cho việc thực hiện các xử lý phụ thuộc thời gian. Chúng bao gồm các hàm cho việc quản lý thời gian hệ thống, trình xử lý *cyclic* (chu kì), và trình xử lý *alarm* (báo động).

Tên chung cho cả trình xử lý *cyclic* và trình xử lý *alarm* là trình xử lý sự kiện thời gian (time event handlers).

2.1 Quản lý thời gian hệ thống

Những hàm quản lý thời gian hệ thống dùng để thao tác với thời gian của hệ thống, bao gồm thiết lập và tham khảo đồng hồ (*clock*), hay tham khảo thời gian hoạt động của hệ thống (*system operating time*).

❖ **Các hàm system call :**

1. tk_set_tim : Thiết lập thời gian

[C Language Interface]

```
ER ercd = tk_set_tim ( SYSTIM *    pk_tim ) ;
```

[Parameters]

SYSTIM * pk_tim	Địa chỉ của gói thời gian hiện tại
Chi tiết pk_tim :	
SYSTIM systim	Thời gian hiện tại của hệ thống cần thiết lập

[Return Parameters]

ER	ercd	Mã lỗi

[Error Codes]

E_OK	Kết thúc bình thường
E_PAR	Lỗi tham số (*pk_tim* không hợp lệ hay thời gian thiết lập không hợp lệ)

[Description]

- Thiết lập giá trị của đồng hồ hệ thống theo một giá trị được chỉ định trong *systim*.
- Thời gian hệ thống được mô tả bằng tổng số *ms* tính từ 0:00:00 (GMT), ngày 1 tháng 1, năm 1985.

- Thời gian *relative* được chỉ định trong RELTIM hay TMO không thay đổi cho dù đồng hồ hệ thống thay đổi bởi hàm *tk_set_tim* trong suốt thời giam hệ thống hoạt động.

2. tk_get_tim : Lấy thời gian hệ thống

[C Language Interface]

ER ercd = tk_get_tim (SYSTIM * pk_tim) ;

[Parameters]

SYSTIM * pk_tim Địa chỉ của gói thời gian hiện tại

Chi tiết pk_tim :

SYSTIM systim Thời gian hiện tại của hệ thống cần lấy

[Return Parameters]

ER ercd Mã lỗi

[Error Codes]

E_OK Kết thúc bình thường

E_PAR Lỗi tham số (*pk_tim* không hợp lệ)

[Description]

- Lấy giá trị của đồng hồ hệ thống và trả nó về trong *systim*.
- Thời gian hệ thống được mô tả bằng tổng số *ms* tính từ 0:00:00 (GMT), ngày 1 tháng 1, năm 1985.
- Thời gian *relative* được chỉ định trong RELTIM hay TMO không thay đổi cho dù đồng hồ hệ thống thay đổi bởi hàm *tk_set_tim* trong suốt thời giam hệ thống hoạt động.

3. tk_get_otm : Lấy thời gian hoạt động

[C Language Interface]

ER ercd = tk_get_otm (SYSTIM * pk_tim) ;

[Parameters]

SYSTIM * pk_tim Địa chỉ trả về của gói thời gian hoạt động

Chi tiết pk_tim :

SYSTIM systim Thời gian hoạt động của hệ thống cần lấy

[Return Parameters]

ER ercd Mã lỗi

[Error Codes]

E_OK Kết thúc bình thường

E_PAR Lỗi tham số (*pk_tim* không hợp lệ)

[Description]

- Lấy thời gian hoạt động của hệ thống.

- Thời gian hoạt động của hệ thống, không giống như thời gian hệ thống, nó cho biết độ dài thời gian được tính từ khi hệ thống được khởi động. Nó không bị ảnh hưởng bởi hàm *tk_set_tim*.
- Thời gian hoạt động của hệ thống phải có cùng độ chính xác với thời gian hệ thống.

2.2 Trình xử lý cyclic

Một trình xử lý cyclic là một trình xử lý sự kiện thời gian xảy ra theo một khoảng thời gian định kì. Những hàm xử lý cyclic được cung cấp để tạo hoặc huỷ bỏ một trình xử lý cyclic, làm hoặc dừng hoạt động của trình xử lý cyclic, và tham khảo trạng thái của một cyclic. Mỗi trình xử lý cyclic được đinh danh bởi một ID.

Thời gian và pha chu kì (gọi tắt là *pha*)thì được chỉ định cho mỗi trình xử lý cyclic khi nó được tạo. Khi hoạt động của một trình xử lý cyclic được yêu cầu, T-Kernel xác định thời gian mà trình xử lý cyclic sẽ bắt đầu lần tiếp theo dựa trên thời gian và pha của trình xử lý cyclic tương ứng. Khi một trình xử lý cyclic được tạo, thì thời gian nó được thực thi tiếp theo là thời gian chu kì cộng với pha. Khi thời gian thực thi một trình xử lý cyclic đến, *exinf*, chứa tthông tin mở rộng về cyclic sẽ được truyền đến nó như là một tham số khởi đầu. Thời gian khi một trình xử lý cyclic thực thi cộng với thời gian chu kì của nó trở thành thời gian thực thi lần kế tiếp. Đôi khi thời gian thực thi kế tiếp sẽ được thiết lập lại trong khi trình xử lý cyclic đang hoạt động.

Theo nguyên tắc thì pha của một trình xử lý cyclic không dài hơn thời gian chu kì của nó.

Môt trình xử lý cyclic có hai trạng thái hoạt động là *active* và *inactive*. Khi trạng thái một trình xử lý cyclic là inactive, nó sẽ không được thực thi cho dù thời gian chu kì của nó đến. Để làm active một trình xử lý cyclic ta gọi hàm system call *tk_sta_cyc*, còn để làm inactive ta gọi hàm system call *tk_stp_cyc*. Còn trạng thái của một trình xử lý cyclic sau khi tạo nó được cho biết trong thuộc tính của nó.

❖ **Các hàm system call :**

1. tk_cre_cyc : Tạo một trình xử lý cyclic

[C Language Interface]

```
ID cycid = tk_cre_cyc ( T_CCYC * pk_ccyc ) ;
```

[Parameters]

T_CCYC * pk_ccyc Tthông tin về trình xử lý cyclic được tạo

Chi tiết của pk_ccyc :

VP	exinf	Tthông tin phụ được thêm vào
ATR	cycatr	Các thuộc tính của trình xử lý cyclic
FP	cychdr	Địa chỉ của trình xử lý cyclic
RETIM	cyctim	Thời gian chu kì
RETIM	cycphs	Pha chu kì (Pha)

[Return Parameters]

ID cycid ID của trình xử lý cyclic

hoặc Mã lỗi (Error codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_NOMEM	Thiếu bộ nhớ (vùng nhớ cho khối điều khiển không được cấp phát)
E_LIMIT	Số lượng trình xử lý cyclic vượt quá giới hạn của hệ thống
E_RSATR	Lỗi thuộc tính (*cycatr* không hợp lệ hay không sử dụng được)
E_PAR	Lỗi tham số (*pk_ccyc* không hợp lệ)

[Description]

- Tạo một trình xử lý cyclic và gán cho nó một ID. Một trình xử lý cyclic là một trình xử lý thực thi tại những khoảng thời gian chỉ định như là một Task-independent portion.
- *exinf* có thể được sử dụng tự do bởi người dùng để chèn thêm tthông tin về trình xử lý cyclic, và được tham khảo bằng hàm *tk_ref_cyc*. Nếu tthông tin của trình xử lý cyclic là một vùng lớn, ứng dụng phải xin cấp phát vùng nhớ riêng biệc và đặt địa chỉ vào *exinf*.
- *cycatr* cho biết các thuộc tính hệ thống trong các bit thấp của nó và tthông tin hiện thực trong các bit cao.

  cycatr := (TA_ASM | | TA_HLNG)| [TA_SSTA] | [TA_PHS]

TA_ASM	Cho biết trình xử lý được viết bằng hợp ngữ
TA_HLNG	Cho biết trình xử lý viết bằng ngôn ngữ cấp cao
TA_STA	Trình xử lý cyclic sẽ hoạt động (active) ngay sau khi được tạo
TA_PHS	Lưu pha chu kì

```
#define TA_ASM      0x00000000 /* Assembly program */
#define TA_HLNG     0x00000001 /* High-level language program */
#define TA_STA      0x00000002 /* activate cyclic handler */
#define TA_PHS      0x00000004 /* save cyclic handler cycle phase */
```

- Khi TA_HLNG được chỉ định, khi trình xử lý cyclic bắt đầu thực thi, nó sẽ nhảy đến địa chỉ của nó không trực tiếp mà thông qua sự hỗ trợ của môi trường ngôn ngữ cấp cao. Hàm xử lý cycle có dạng như sau :

```
void cychdr(VP exinf )
{
    /*
    (processing)
    */
    return ; //Thoát khỏi trình xử lý cyclic
}
```

  Định dạng của trình xử lý cyclic khi TA_ASM được chỉ định phụ thuộc vào sự hiện thực nhưng tham số *exinf* được truyền trong tham số khởi động.

- *cycphs* cho biết độ dài thời gian từ lúc được tạo cho tới khi nó bắt đầu (tức là đến thời gian xử lý, tuy nhiên được xử lý hay không còn tuỳ thuộc vào trang thái của cycle). Sau đó nó được bắt đầu tuần hoàn theo khoảng thời gian chu kì được chỉ định trong *cyctim*. Nếu *cycphs* = 0 thì trình xử lý cyclic bắt đầu ngay sau khi nó được tạo. *cyctim* không thể

được chỉ định là 0. Khoảng thời gian bắt đầu của trình xử lý cyclic lần thứ *n* sau khi được tạo ít nhất là *cycphs* + *cyctim** (*n* – 1).

- Khi TA_STA được chỉ định, trình xử lý cyclic sẽ ở trạng thái active ngay sau khi được tạo, và thực thi theo thời gian đã trình bày ở trên. Nếu TA_STA không được chỉ định, thì thời gian chu kì vẫn được tính nhưng trình xử lý cyclic sẽ không được thực thi một các thật sự.
- Khi TA_PHS được chỉ định, thì thời gian chu kì sẽ không được khởi động lại mỗi khi hàm *tk_sta_cyc* để làm active trình xử lý cyclic, ngược lại nếu TA_PHS được chỉ định thì hàm *tk_sta_cyc* sẽ khởi động lại thời gian chu kì và trình xử lý cyclic sẽ thực thi sau khoảng thời gian *cyctim* từ lúc gọi hàm *tk_sta_cyc.* Hàm *tk_sta_cyc* không ảnh hưởng gì đến *cycphs*.
- Trong khi trình xử lý cyclic đang thực thi, thì bất cứ sự *dispatching* nào cũng bị làm trễ cho tới khi trình xử lý kết thúc thực thi
- Một trình xử lý cyclic thực thi như là một Task-independent portion, do vậy không thể gọi một hàm system call trong trình xử lý cyclic.

2. tk_del_cyc : Hủy bỏ trình xử lý cyclic

[C Language Interface]

ER ercd = tk_del_cyc (ID cycid) ;

[Parameters]

ID	cycid	ID của trình xử lý cyclic cần xoá

[Return Parameters]

ER	ercd	Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*cycid* không hợp lệ hay không được sử dụng
E_NOEXS	Trình xử lý cyclic được chỉ định trong *cycid* không tồn tại

[Description]

- Loại bỏ một trình xử lý cyclic.

3. tk_sta_cyc : Khởi động trình xử lý cyclic

[C Language Interface]

ER ercd = tk_sta_cyc (ID cycid) ;

[Parameters]

ID	cycid	ID của trình xử lý cyclic

[Return Parameters]

ER	ercd	Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*cycid* không hợp lệ hay không được sử dụng
E_NOEXS	Trình xử lý cyclic được chỉ định trong *cycid* không tồn tại

[Description]

- Làm active một trình xử lý cyclic, đặt nó trong trạng thái active.
- Khi thuộc tính TA_PHS được chỉ định, thì thời gian chu kì sẽ không được khởi động lại mỗi khi trình xử lý cyclic đi vào trạng thái active. Nếu nó đang ở trạng thái active khi hàm system call này được thực thi, thì nó sẽ tiếp tục mà không có sự thay đổi gì trong trạng thái active. Ngược lại nếu TA_PHS được chỉ định thì hàm *tk_sta_cyc* sẽ khởi động lại thời gian chu kì và trình xử lý cyclic sẽ thực thi sau khoảng thời gian *cyctim* từ lúc gọi hàm *tk_sta_cyc*.

4. tk_stp_cyc : Dừng trình xử lý cyclic

[C Language Interface]

ER ercd = tk_stp_cyc (ID cycid) ;

[Parameters]

ID	cycid	ID của trình xử lý cyclic

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*cycid* không hợp lệ hay không được sử dụng
E_NOEXS	Trình xử lý cyclic được chỉ định trong *cycid* không tồn tại

[Return Parameters]

ER	ercd	Mã lỗi (Error Codes)

[Description]

- Làm inactive một trình xử lý cyclic, đặt nó trong trạng thái inactive. Nếu nó đang ở trong trạng thái inactive thì hàm này sẽ không có ảnh hưởng gì.

5. tk_ref_cyc : tham khảo trạng thái trình xử lý cyclic

[C Language Interface]

ER ercd = tk_ref_cyc (ID cycid, T_RCYC * pk_rcyc) ;

[Parameters]

ID	cycid	Cyclic handler ID
T_RCYC*	pk_rcyc	Địa chỉ của gói tthông tin về trạng thái trình xử lý cyclic

Chi tiết pk_rcyc :

VP	exinf	Thông tin mở rộng
RELTIM	lfttim	Thời gian còn lại cho tới thời gian bắt đầu kế tiếp
UINT	cycstat	Trạng thái của cyclic

(Những tham số khác tùy theo hiện thực có thêm vào sau)

[Return Parameters]

ER	ercd	Mã lỗi

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*cycid* không hợp lệ hay không được sử dụng
E_NOEXS	Trình xử lý cyclic được chỉ định trong *cycid* không tồn tại
E_PAR	Lỗi tham số (địa chỉ của gói tthông tin trả về không dùng được)

[Description]

- Tham khảo trạng thái của trình xử lý cyclic có ID là cycid, truyền trong các thông số trả về state của cyclic cycstat, khoảng thời gian còn lại cho tới khi lần bắt đầu tiếp theo đến , và tthông tin mở rộng (*exinf*).
- Những tthông tin sau được trả về trong *cycstat*

cycstat := (TCYC_STP | TCYC_STA)

TCYC_STP	Cyclic ở trạng thái inactive
TCYC_STA	Cyclic ở trạng thái active

```
#define  TCYC_STP   0x00     //Cyclic ở trạng thái inactive
#define  TCYC_STA   0x01     //    Cyclic ở trạng thái active
```

- Nếu trình xử lý cyclic tương ứng không tồn tại, mã lỗi E_NOEXS được trả về.

2.3 Trình xử lý alarm

Một trình xử lý alarm là một trình xử lý sự kiện thời gian xảy ra tại một thời điểm được chỉ định. Những hàm xử lý alarm được cung cấp để tạo hoặc huỷ bỏ một trình xử lý alarm, làm active hoặc inactive trình xử lý alarm, và tham khảo trạng thái của một alarm. Mỗi trình xử lý alarm được định danh bởi một ID.

Thời gian mà tại đó trình xử lý alrm bắt đầu (còn gọi là thời gian alarm) có thể được thiết lập độc lập cho mỗi trình xử lý alarm.

Sau khi một trình xử lý alarm được tạo, thời gian alarm chưa được thiết lập và nó ở trong trạng thái inactive. Thời gian alarm được thiết lập khi trình xử lý alarm đó đi vào trạng thái active bởi hàm system call *tk_sta_alm*, và sau đó thời gian sẽ được tính. Khi hàm *tk_stp_alm* được gọi để làm inactive một trình xử lý alarm thì thời gian thiết lập cho alarm đó sẽ bị huỷ bỏ.

❖ **Các hàm system call :**

1. tk_cre_alm : Tạo một trình xử lý alarm

[C Language Interface]

```
ID cycid = tk_cre_alm ( T_CALM * pk_calm ) ;
```

[Parameters]

T_CALM * pk_calm Thông tin về trình xử lý alarm được tạo

Chi tiết của pk_calm :

VP	exinf	Tthông tin phụ được thêm vào
ATR	almatr	Các thuộc tính của trình xử lý alarm
FP	almhdr	Địa chỉ của trình xử lý alarm

[Return Parameters]

ID almid ID của trình xử lý alarm

hoặc Mã lỗi (Error codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_NOMEM	Thiếu bộ nhớ (vùng nhớ cho khối điều khiển không được cấp phát)
E_LIMIT	Số lượng trình xử lý alarm vượt quá giới hạn của hệ thống
E_RSATR	Lỗi thuộc tính (*almatr* không hợp lệ hay không sử dụng được)
E_PAR	Lỗi tham số (*pk_calm* không hợp lệ)

[Description]

- Tạo một trình xử lý a;arm và gán cho nó một ID. Một trình xử lý a;arm là một trình xử lý thực thi tại thời gian chỉ định như là một Task-independent portion.
- *exinf* có thể được sử dụng tự do bởi người dùng để chèn thêm tthông tin về trình xử lý alarm, và được tham khảo bằng hàm *tk_ref_alm*. Nếu tthông tin của trình xử lý alarm là một vùng lớn, ứng dụng phải xin cấp phát vùng nhớ riêng biệc và đặt địa chỉ vào *exinf*.
- *almatr* cho biết các thuộc tính hệ thống trong các bit thấp của nó và tthông tin hiện thực trong các bit cao.

  almatr := (TA_ASM || TA_HLNG)

  TA_ASM Cho biết trình xử lý được viết bằng hợp ngữ

  TA_HLNG Cho biết trình xử lý viết bằng ngôn ngữ cấp cao

```
#define TA_ASM    0x00000000 /* Assembly program */
#define TA_HLNG   0x00000001 /* High-level language program */
```

- Khi TA_HLNG được chỉ định, khi trình xử lý alarm bắt đầu thực thi, nó sẽ nhảy đến địa chỉ của nó không trực tiếp mà thông qua sự hỗ trợ của môi trường ngôn ngữ cấp cao. Hàm xử lý alarm có dạng như sau :

```
void almhdr (VP exinf )
{
    /*
    (processing)
    */
    return ; //Thoát khỏi trình xử lý alarm
}
```

Định dạng của trình xử lý alarm khi TA_ASM được chỉ định phụ thuộc vào sự hiện thực nhưng tham số *exinf* được truyền trong tham số khởi động.

- Trong khi trình xử lý alarm đang thực thi, thì bất cứ sự *dispatching* nào cũng bị làm trễ cho tới khi trình xử lý kết thúc thực thi
- Một trình xử lý alarm thực thi như là một Task-independent portion, do vậy không thể gọi một hàm system call trong trình xử lý alarm.

2. tk_del_alm : Hủy bỏ trình xử lý alarm

[C Language Interface]

ER ercd = tk_del_alm (ID almid) ;

[Parameters]

ID	almid	ID của trình xử lý alarm cần xoá

[Return Parameters]

ER	ercd	Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*almid* không hợp lệ hay không được sử dụng
E_NOEXS	Trình xử lý alarm được chỉ định trong *almid* không tồn tại

[Description]

- Loại bỏ một trình xử lý alarm.

3. tk_sta_alm : Khởi động trình xử lý alarm

[C Language Interface]

ER ercd = tk_sta_alm (ID almid , RELTIM almtim) ;

[Parameters]

ID	almid	ID của trình xử lý alarm
RELTIM	almtim	Thời gian trình xử lý alarm bắt đầu

[Return Parameters]

ER	ercd	Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*almid* không hợp lệ hay không được sử dụng
E_NOEXS	Trình xử lý alarm được chỉ định trong *almid* không tồn tại

[Description]

- Thiết lập thời gian alarm của trình xử lý alarm có ID là almid bằng thời gian được chỉ định trong tham số almtim và đặt nó trong trạng thái active.

- Sau một khoảng thời gian bằng *almtim* từ lúc gọi hàm *tk_sta_alm* thì trình xử lý alarm sẽ bắt đầu thực thi. Nếu trình xử lý alarm đó đang ở trong trạng thái active khi gọi hàm này thì sự thiết lập *almtim* cũ sẽ bị bỏ qua và thời gian alarm sẽ được thiết lập lại mới theo chỉ định.
- Nếu *almtim* = 0 thì trình xử lý alarm sẽ bắt đầu ngay khi nó được active.

4. tk_stp_alm : Dừng trình xử lý alarm

[C Language Interface]

ER ercd = tk_stp_alm (ID almid) ;

[Parameters]

ID	almid	ID của trình xử lý alarm cần xoá

[Return Parameters]

ER	ercd	Mã lỗi (Error Codes)

[Error Codes]

E_OK	Kết thúc bình thường
E_ID	*almid* không hợp lệ hay không được sử dụng
E_NOEXS	Trình xử lý alarm được chỉ định trong *almid* không tồn tại

[Description]

- Huỷ bỏ thời gian alarm của một trình xử lý alarm, đặt nó trong trạng thái inactive. Nếu nó đang ở trong trạng thái inactive thì hàm này sẽ không có ảnh hưởng gì.

5. tk_ref_alm : Tham khảo trạng thái trình xử lý alarm

[C Language Interface]

ER ercd = tk_ref_alm (ID almid, T_RALM * pk_ralm) ;

[Parameters]

ID	almid	Alarm handler ID
T_RALM*	pk_ralm	Địa chỉ của gói tthông tin về trạng thái trình xử lý cyclic

Chi tiết pk_ralm :

VP	exinf	Thông tin mở rộng
RELTIM	lfttim	Thời gian còn lại cho tới thời gian mà trình xử lý alarm bắt đầu
UINT	almstat	Trạng thái của alarm

(Những tham số khác tùy theo hiện thực có thêm vào sau)

[Return Parameters]

ER	ercd	Mã lỗi

[Error Codes]

E_OK	Kết thúc bình thường

E_ID	*almid* không hợp lệ hay không được sử dụng
E_NOEXS	Trình xử lý alarm được chỉ định trong *almid* không tồn tại
E_PAR	Lỗi tham số (địa chỉ của gói tthông tin trả về không dùng được)

[Description]

- Tham khảo trạng thái của trình xử lý alarm có ID là *almid*, truyền trong các thông số trả về state của alarm alm*stat*, khoảng thời gian còn lại cho tới khi trình xử lý bắt đầu, và tthông tin mở rộng (*exinf*).
- Những tthông tin sau được trả về trong *almstat*

  almstat := (TCYC_STP | TCYC_STA)

TALM_STP	Alarm ở trạng thái inactive
TALM_STA	Alarm ở trạng thái active

```
#define  TALM_STP   0x00     //   Alarm ở trạng thái inactive
#define  TALM_STA   0x01     //   Alarm ở trạng thái active
```

Nếu trình xử lý alarm tương ứng không tồn tại, mã lỗi E_NOEXS được trả về.

3. THỰC HÀNH

3.1 Cyclic handler

Bạn hãy chạy và cho biết chương trình sau làm gì vẽ lại sơ đồ

```
#include <basic.h>
#include <tk/tkernel.h>
#include <stdio.h>

IMPORT void cychdr(VP);
ID cyctskid = -1;
ID taskid1 = -1;
INT cnt;

/*************************************************************
task1 wakeup every 10 seconds and display print
 *************************************************************/
IMPORT void task1(INT stacd, VP exinf)
{
  ER ercd = 0;
  SYSTIM systime;
  INT count = *((INT *)exinf);

  while (1)
  {
  printf("sleep task now\n");
  ercd = tk_slp_tsk(TMO_FEVR);
  if (ercd < 0)
  {
     printf("tk_slp_tsk error = %x\n", ercd);
     break;
  }
  ercd = tk_get_tim(&systime);
  printf("time hi = %d lo = %ul\n", systime.hi, systime.lo);
```

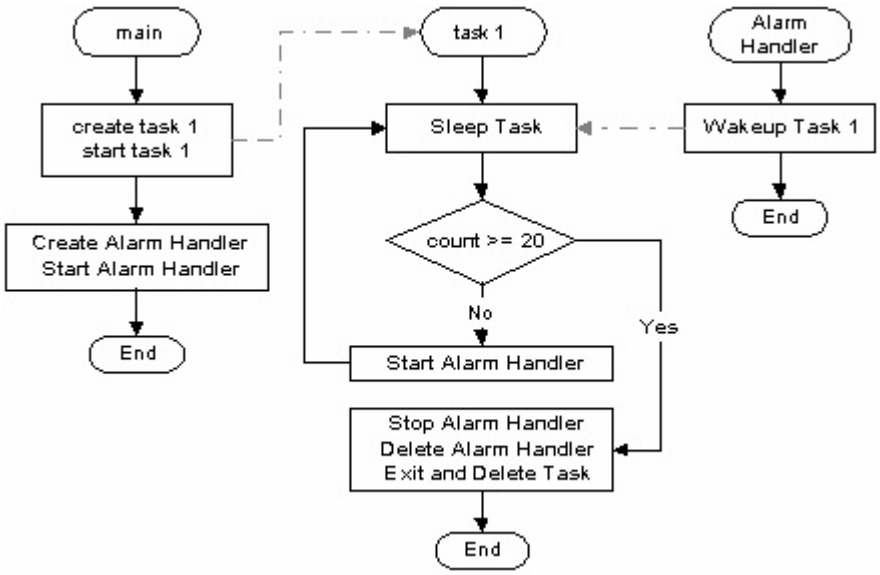
```
  printf("This is task 1, going for 20 loop\n");
  if (count++ >= 20)
  {
    printf("20 loops over\n");
    break;
  }
  printf("count = %d\n", count);
  }
  printf("exit and del task now\n");
  ercd = tk_stp_cyc(cyctskid);
  if (ercd < 0)
  {
  printf("tk_stp_cyc error = %x\n", ercd);
  }
  ercd = tk_del_cyc(cyctskid);
  if (ercd < 0)
  {
  printf("tk_del_cyc error = %x\n", ercd);
  }
  cyctskid = -1;
  taskid1 = -1;
  ercd = tk_get_otm(&systime);
  printf("OS time hi = %d lo = %ul\n", systime.hi, systime.lo);
  tk_exd_tsk();
}


/************************************************************************
**********
main
************************************************************************
*********/
EXPORT ER main( INT ac, UB *av[] )
{
  T_CCYC cyctsk;
  T_CTSK ctsk;
  ER ercd = 0;
  printf("main: (ac=%d)\n", ac);

  if (ac < 0)
  {
    if (cyctskid >= 0)
    {
    ercd = tk_stp_cyc(cyctskid);
    ercd = tk_del_cyc(cyctskid);
    cyctskid = -1;
    }
    if (taskid1 >= 0)
    {
    tk_ter_tsk(taskid1);
    tk_del_tsk(taskid1);
    taskid1 = -1;
    }
    goto ext;
  }

  cnt = 0;
  ctsk.exinf = (VP)&cnt;
  ctsk.tskatr = TA_HLNG | TA_RNG0;
```

```
  ctsk.task = task1;
  ctsk.itskpri = 80;
  ctsk.stksz = 1024 * 4;
  taskid1 = tk_cre_tsk(&ctsk);
  printf("tk_cre_tsk: (taskid = %d)\n", taskid1);
  if (taskid1 < E_OK)
  {
    printf("tk_cre_tsk error = %x\n", taskid1);
    goto ext;
  }

  ercd = tk_sta_tsk(taskid1, 0);
  if (ercd < 0)
  {
    printf("tk_sta_tsk error %x\n", ercd);
    tk_del_tsk(taskid1);
    taskid1 = -1;
    goto ext;
  }

  cyctsk.exinf = (VP)0x00000000;
  cyctsk.cycatr = TA_HLNG;
  cyctsk.cychdr = cychdr;
  cyctsk.cyctim = 1000;
  cyctsk.cycphs = 0;
  cyctskid = tk_cre_cyc(&cyctsk);
  printf("tk_cre_cyc: (cyctskid = %d)\n", cyctskid);
  if (cyctskid < E_OK)
  {
    tk_del_tsk(taskid1);
    taskid1 = -1;
    goto ext;
  }

  printf("This is cychandler, going for 20 time\n");
  ercd = tk_sta_cyc(cyctskid);
  if (ercd < 0)
  {
    printf("tk_sta_cyc error = %x\n", ercd);
    ercd = tk_del_cyc(cyctskid);
    tk_del_tsk(taskid1);
    taskid1 = -1;
    cyctskid = -1;
  }

  /*end*/
 ext:
  printf("main ended\n");
  return 0;
}

/*****************************************************************
task1 wakeup every 10 seconds and display print
 *****************************************************************/
IMPORT void cychdr(VP exinf)
{
  ER ercd = 0;

  ercd = tk_wup_tsk(taskid1);
```

```
  return;
}
```

3.2 Alarm handler

Đọc hiểu và chạy chương trình sau đây




```
#include <basic.h>
#include <tk/tkernel.h>
#include <stdio.h>
#ifdef DEBUG
#include <util/tmonitor.h>
#endif

IMPORT void almhdr(VP);
ID almtskid = -1;
ID taskid1 = -1;
INT cnt;

/*****************************************************************
task1 wakeup every 10 seconds and display print
 *****************************************************************/
IMPORT void task1(INT stacd, VP exinf)
{
  ER ercd = 0;
  SYSTIM systime;
  INT count = *((INT *)exinf);

  while (1)
  {
   printf("sleep task now\n");
   ercd = tk_slp_tsk(TMO_FEVR);
   if (ercd < 0)
   {
     printf("tk_slp_tsk error = %x\n", ercd);
     break;
   }
   ercd = tk_get_tim(&systime);
```

```
  printf("time hi = %d lo = %ul\n", systime.hi, systime.lo);
  printf("This is task 1, going for 20 loop\n");
  if (count++ >= 20)
  {
    printf("20 loops over\n");
    break;
  }
  printf("count = %d\n", count);
  ercd = tk_sta_alm(almtskid, 1500);
  if (ercd < 0)
  {
    printf("tk_sta_alm error = %x\n", ercd);
    break;
  }
  }
  printf("exit and del task now\n");
  ercd = tk_stp_alm(almtskid);
  if (ercd < 0)
  {
  printf("tk_stp_cyc error = %x\n", ercd);
  }
  ercd = tk_del_alm(almtskid);
  if (ercd < 0)
  {
  printf("tk_del_cyc error = %x\n", ercd);
  }
  almtskid = -1;
  taskid1 = -1;
  ercd = tk_get_otm(&systime);
  printf("OS time hi = %d lo = %ul\n", systime.hi, systime.lo);
  tk_exd_tsk();
}

/*************************************************************************
**main
*************************************************************************
*/
EXPORT ER main( INT ac, UB *av[] )
{
  T_CALM almtsk;
  T_CTSK ctsk;
  ER ercd = 0;

#ifdef DEBUG
  tm_monitor();
#endif
  printf("main: (ac=%d)\n", ac);

  if (ac < 0)
  {
    if (almtskid >= 0)
    {
    ercd = tk_stp_alm(almtskid);
    ercd = tk_del_alm(almtskid);
    almtskid = -1;
    }
    if (taskid1 >= 0)
    {
    tk_ter_tsk(taskid1);
```

```
    tk_del_tsk(taskid1);
    taskid1 = -1;
    }
    goto ext;
  }

  cnt = 5;
  ctsk.exinf = (VP)&cnt;
  ctsk.tskatr = TA_HLNG | TA_RNG0;
  ctsk.task = task1;
  ctsk.itskpri = 80;
  ctsk.stksz = 1024 * 4;
  taskid1 = tk_cre_tsk(&ctsk);
  printf("tk_cre_tsk: (taskid = %d)\n", taskid1);
  if (taskid1 < E_OK)
  {
    printf("tk_cre_tsk error = %x\n", taskid1);
    goto ext;
  }
  ercd = tk_sta_tsk(taskid1, 0);
  if (ercd < 0)
  {
    printf("tk_sta_tsk error %x\n", ercd);
    tk_del_tsk(taskid1);
    taskid1 = -1;
    goto ext;
  }

  almtsk.exinf = (VP)0x00000000;
  almtsk.almatr = TA_HLNG;
  almtsk.almhdr = almhdr;
  almtskid = tk_cre_alm(&almtsk);
  printf("tk_cre_almc: (almtskid = %d)\n", almtskid);
  if (almtskid < E_OK)
  {
    tk_del_tsk(taskid1);
    taskid1 = -1;
    goto ext;
  }

  printf("This is alarm handler, going for 20 time\n");
  ercd = tk_sta_alm(almtskid, 2000);
  if (ercd < 0)
  {
    printf("tk_sta_cyc error = %x\n", ercd);
    ercd = tk_del_alm(almtskid);
    ercd = tk_del_tsk(taskid1);
    taskid1 = -1;
    almtskid = -1;
  }

  /*end*/
 ext:
  printf("main ended\n");
  return 0;
}

/*****************************************************************
task1 wakeup every 10 seconds and display print
```

```
 ************************************************************/
IMPORT void almhdr(VP exinf)
{
  ER ercd = 0;

  // printf("tk_wup_tsk\n");
  ercd = tk_wup_tsk(taskid1);
  if (ercd <0)
  {
   //printf("tk_wup_tsk error = %x\n", ercd);
  }
  return;
}
```

BÀI THỰC HÀNH SỐ 9

DEVICE MANAGERMENT và SCREEN DRIVER

1. MỤC TIÊU

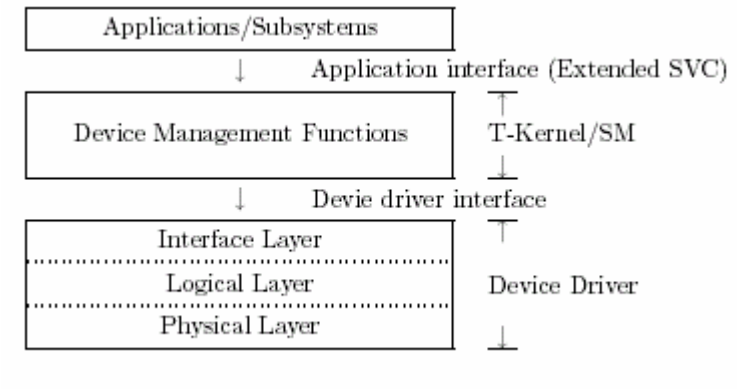
- Hiểu được cách mà hệ thống quản lý các thiết bị và cách giao tiếp với các thiết bị này thông qua các device driver.
- Viết được một chương trình đơn giản để giao tiếp thông qua screen.

2. LÝ THUYẾT

Sau đây là những gì cần lưu ý khi sử dụng các hàm quản lý hệ thống :

- Theo nguyên tắc, những hàm có tên *tk*_--- là hàm SVC mở rộng, những hàm còn lại là những hàm thư viện (bao gồm cả những hàm inline) hay những macro của ngôn ngữ C.
- Một vài thư viện và macro gọi hàm SVC mở rộng hay những hàm system call một cách gián tiếp.
- Những mã lỗi như E_PAR, E_MACV, và E_NOMEM sẽ không được mô tả mặc dù luôn có khả năng xuất hiện, ngoại trừ một vài lí do đặt biệt nào đó.
- Ngoại trừ những nơi có chú ý, hàm SVC mở rộng và những hàm thư viện của T-Kernel/SM không thể được gọi từ một Task-independent portion và trong khi dispatching và interupt bị vô hiệu. Tuy nhiên, có một vài giới hạn phụ thuộc vào sự hiện thực (E_CTX) .
- Hàm SVC mở rộng và những hàm thư viện của T-Kernel/SM không thể được gọi từ một mức bảo vệ thấp hơn mà tại đó những hàm system call của T-Kernel/OS có thể được gọi (thấp hơn TSVCLimit)(E_OACV).

2.1 Những khái niệm cơ bản



Hình 2.1 Những hàm quản lý thiết bị

Device Name (UB* type)

Mỗi device name là một chuỗi kí tự có chiều dài lên tới 8 kí tự chứa các phần tử sau.

```
#define  L_DEVNM  8   /*chiều dài tên thết bị*/
```

Type Tên cho biết loại thiết bị.

Những kí tự từ ‘a’ đến ‘z’ và ‘A’ đến ‘Z’ có thể được sử dụng.

Unit Một chữ cái chỉ ra một thiết bị vật lý.

Mỗi unit được gán một kí thự từ ‘a’ đến ‘z’ theo thứ tự từ ‘a’.

Subunit Gồm 1 đến 3 kí số cho biết một thiết bị luận lý.

Mỗi subunit được gán một số từ 0 tới 254 theo thứ tự bắt đầu từ 0.

Device name có định dạng gồm type + unit + subunit. Một vài thiết bị có thể không có unit hay subunit.

Một tên chứa type + unit được gọi là tên một thiết bị vật lý. Một tên chứa type + unit + subunit được gọi là tên một thiết bị luận lý để phân biệt nó với tên một thiết bị vật lý. Nếu không có subunit thì tên thiết bị vật lý và tên thiết bị luận lý là giống nhau. Thuật ngữ “device name” chính nó có nghĩa là tện một thiết bị luận lý.

Một subunit nói chung thì tham khảo đến một phân vùng của một đĩa cứng, nhưng cũng có thể được sử dụng như là tên một thiết bị luận lý khác.

Ví dụ

had	Đĩa cứng
hda0	Đĩa cứng (phân vùng 1)
fda	Đĩa mềm
rsa	Cổng nối tiếp
kpbd	Keyboard/poiting device

Device ID (ID type)

Bằng cách đăng kí một thiết bị (device driver) với T-Kernel/SM, một device ID (>0) được gán cho thiết bị đó (tên thiết bị vật lý). ID của thiết bị luận lý chứa ID thiết bị vật lý thêm vào số subunit (1 đến 255).

devid :	ID của thiết bị tại phần đăng kí
devid	Thiết bị vật lý
devid + n + 1	Subunit thứ n (thiết bị luận lý)

Ví dụ

had	devid	Entire hard disk
hda0	devid + 1	1st partition of hard disk
hda1	devid + 2	2nd partition of hard disk

Device Attribute (ATR type)

Nhữgn thuộc tính của thiết bị được định nghĩa như sau để phân loại thiết bị bằng các thuộc tính của chúng.

IIII IIII IIII IIII PRxx xxxx KKKK KKKK

16 bit cao là những thuộc tính phụ thuộc thiết bị được định nghĩa cho mỗi thiết bị.

16 bit thấp là những thuộc tính chuẩn được định nghĩa như sau.

```
#define TD_PROTECT      0x8000   /* P: write protection */
#define TD_REMOVABLE    0x4000   /* R: removable media */

#define TD_DEVKIND      0x00ff      /* K: device/media kind */
#define TD_DEVTYPE      0x00f0   /* device type */

                                                /* device type */
#define TDK_UNDEF     0x0000  /* undefined/unknown */
#define TDK_DISK      0x0010  /* disk device */

                                                /* disk kind */
#define TDK_DISK_UNDEF    0x0010  /* miscellaneous disk */
#define TDK_DISK_RAM      0x0011     /* RAM disk (used as main memory) */
#define TDK_DISK_ROM      0x0012     /* ROM disk (used as main memory) */
#define TDK_DISK_FLA      0x0013  /* Flash ROM or other silicon disk */
#define TDK_DISK_FD       0x0014     /* floppy disk */
#define TDK_DISK_HD       0x0015  /* hard disk */
#define TDK_DISK_CDROM    0x0016  /* CD-ROM */
```

Hiện tại thì không có thiết bị nào ngoài đĩa được định nghĩa. Những thiết bị khác được gán là loại không xác định (TDK_UNDEF).

Device Descriptor (ID type)

Một device descriptor (>0) là một định danh được sử dụng cho sự truy cập một thiết bị, được gán bởi T-Kernel/SM khi một thiết bị được mở để sử dụng.

Một device descriptor phụ thuộc vào một nhóm resou. những hành vi sử dụng một device description có thể được thể hiện chỉ bởi những task thuộc cùng nhóm resource như device desriptor đó. Mã lỗi E_OACV được trả về nếu điều trên bị vi phạm.

Request ID (ID type)

Khi có một yêu cầu IO đến một thiết bị, một ID yêu cầu (>0) được gán để định danh yêu cầu đó. ID này có thể được sử dụng để đợi IO hoàn thành.

Data Number (INT type)

Data của thiết bị được chỉ định bằng một data. Data được phân loại thành data chi tiết thiết bị (device-specific data) và data thuộc tính (attribute data).

Device-specific data : Data number ≥ 0

Attribute data : Datanumber <0

Attribute data chỉ định driver hoặc trạng thái thiết bị đạt được hoặc chế dộ thiết đặt, và những chức năng đặt biệt, .v.v…

2.2 Interface ứng dụng

Những hàm sau đây được cung cấp như là những hàm interface ứng dụng, được gọi như SVC mở rộng. Những hàm này không thể được gọi từ một Task-independent portion và trong khi dispatching và interupt bị vô hiệu (E_CTX).

ID tk_opn_dev (UB *devnm, UINT omode)

ER tk_cls_dev (ID dd, UINT option)

ID tk_rea_dev (ID dd, INT start, VP buf, INT size, TMO tmout)

ER tk_srea_dev (ID dd, INT start, VP buf, INT size, INT *asize)

ID tk_wri_dev (ID dd, INT start, VP buf, INT size, TMO tmout)

ER tk_swri_dev (ID dd, INT start, VP buf, INT size, INT *asize)

ID tk_wai_dev (ID dd, ID reqid, INT *asize, ER *ioerr, TMO tmout)

INT tk_sus_dev (UINT mode)

INT tk_get_dev (ID devid, UB *devnm)

ID tk_ref_dev (UB *devnm, T_RDEV *rdev)

ID tk_oref_dev (ID dd, T_RDEV *rdev)

INT tk_lst_dev (T_LDEV *ldev, INT start, INT ndev)

INT tk_evt_dev (ID devid, INT evttyp, VP evtinf)

- **ID tk_opn_dev (UB *devnm, UINT omode)**

  devnm Tên thiết bị

  omode Chế độ mở

  mã trả về Device descriptor hay lỗi

Mở một thiết bị có tên là *devnm* với chế độ là *omode*, và chuẩn bị cho việc truy cập thiết bị

omode := (TD_READ | | TD_WRITE | | TD_UPDATE) | [TD_EXCL | |
TD_WEXCL] | [TD_NOLOCK]

```
#define   TD_READ     0x0001  /* read only */
#define   TD_WRITE    0x0002  /* write only */
#define   TD_UPDATE   0x0003  /* read/write */
#define   TD_EXCL     0x0100  /* exclusive */
#define   TD_WEXCL    0x0200  /* exclusive write */
#define   TD_NOLOCK   0x1000   /* lock (making resident) not necessary */
```

Khi TD_READ được thiết lập thì hàm *tk_wri_dev ( )* không thể được sử dụng.

Khi TD_WRITE được thiết lập thì hàm *tk_rea_dev ( )* không thể được sử dụng.

Khi TD_EXCL được thiết lập thì tất cả các hành vi mở đồng thời bị cấm.

Khi TD_WEXCL được thiết lập thì tất cả các hành vi mở đồng thời (concurrent opening) trong chế độ ghi (TD_WRITE hay TD_UPDATE) bị cấm.

Present Open Mode		Concurrent Open Mode					
		No exclusive mode		TD_WEXCL		TD_EXCL	
		R	W	R	W	R	W
No exclusive mode	R	Yes	Yes	Yes	Yes	No	No
	W	Yes	Yes	No	No	No	No
TD_WEXCL	R	Yes	No	Yes	No	No	No
	W	Yes	No	No	No	No	No
TD_EXCL	R	No	No	No	No	No	No
	W	No	No	No	No	No	No

R = TD_READ

W = TD_WRITE hay TD_UPDATE

Yes = có thể được mở

No = không thể được mở (E_BUSY)

E_BUSY	Thiết bị đang bận (mở độc quyền)
E_NOEXS	Thiết bị không tồn tại
E_LIMIT	Số lượng mở vượt quá giới hạn

Những lỗi khác có thể có do device driver trả về

- **ER tk_cls_dev (ID dd, UINT option)**

dd	Device descriptor
option	Tuỳ chọn đóng

Đóng một thiết bị có descriptor là *dd*. Nếu có một yêu cầu về thiết bị thì yêu cầu đó sẽ bị bỏ qua và thiết bị sẽ bị đóng.

option := [TD_EJECT]

#define TD_EJECT 0x0001 /* eject media */

Nếu thiết bị như nhau không được mở bởi một task khác thì media sẽ được tháo ra. Trong trường hợp thiết bị không thể tháo madia thì yêu cầu sẽ bị bỏ qua.

E_ID	*dd* không hợp lệ

Những lỗi khác có thể có do device driver trả về

- **ID tk_rea_dev (ID dd, INT start, VP buf, INT size, TMO tmout)**

dd	Device descriptor
start	Vùng bắt đầu đọc (≥ 0: Device-specific data, < 0: Attribute data)
buf	Vùng đệm để chứa dữ liệu đọc
size	Kích thước cần đọc
tmout	Thời gian timeout chấp nhận yêu cầu (ms)
return code	Request ID hoặc lỗi

Bắt đầu đọc data chi tiết thiết bị hoặc data thuộc tính từ thiết bị được chỉ định.Hàm này chỉ đọc và trả về mà không cần phải đợi cho việc đọc hoàn thành. Không gian được chỉ định trong *buf* không được mất cho tới khi việc đọc hoàn thành. Việc đợi cho hành vi đọc hoàn thành được thực hiện bằng cách gọi hàm *tk_wai_dev*.

Thời gian cho hành vi đọc là khác nhau cho mỗi thiết bị.

Trong trường hợp đọc data chi tiết thiết bị, mỗi loại thiết bị sẽ quyết định *start* và *size* cho riêng data đó. Với data thuộc tính, *start* là một chỉ số data thuộc tính và *size* tính bằng byte. Data thuộc tính được chỉ định sẽ được đọc. Thông thường thì kích thước đọc ít nhất phải bằng với kích thước của data thuộc tính đó. Việc đọc nhiều thuộc tính thì không thể thực hiện được trong một hành vi đọc. Khi *size* = 0, thì việc đọc thực sự sẽ không thực hiện nhưng kích thước hiện tại của data sẽ được kiểm tra.

Việc có hay không một yêu cầu mới được chấp nhận trong khi quá trình đọc hay ghi chưa kết thúc là phụ thuộc vào device driver. Nếu yêu cầu mới không được chấp nhận thì nó sẽ được xếp trong hàng đợi, thời gian timeout cho việc đợi yêu cầu được chấp nhận được thiết lập trong *tmout* (có thể là TMO_POL hay TMO_FEVR).

E_ID	*dd* không hợp lệ hoặc chưa được mở
E_OACV	Chế độ mở không hợp lệ (không cho phép đọc)
E_LIMIT	Số lượng yêu cầu vượt quá giới hạn
E_ABORT	Quá trình bị bỏ qua

Những lỗi khác có thể có do device driver trả về

- **ID tk_srea_dev (ID dd, INT start, VP buf, INT size, INT *asize)**

Đọc đồng bộ. Hàm này tương đương đoạn code sau.

```
ER tk_srea_dev( ID dd, INT start, VP buf, INT size, INT *asize )
{
        ER er, ioer;
        er = tk_rea_dev(dd, start, buf, size, TMO_FEVR);
        if ( er > 0 ) {
                er = tk_wai_dev(dd, er, asize, &ioer, TMO_FEVR);
                if ( er > 0 ) er = ioerr;
        }
        return er;
}
```

- **ID tk_wri_dev (ID dd, INT start, VP buf, INT size, TMO tmout)**

dd	Device descriptor
start	Vùng bắt đầu ghi (≥ 0: Device-specific data, < 0: Attribute data)
buf	Vùng đệm để chứa dữ liệu cần ghi
size	Kích thước cần ghi
tmout	Thời gian timeout chấp nhận yêu cầu (ms)

return code Request ID hoặc lỗi

Bắt đầu ghi data chi tiết thiết bị hoặc data thuộc tính từ thiết bị được chỉ định.Hàm này chỉ ghi và trả về mà không cần phải đợi cho việc ghi hoàn thành. Không gian được chỉ định trong *buf* không được mất cho tới khi việc ghi hoàn thành. Việc đợi cho quá trình ghi hoàn thành được thực hiện bằng cách gọi hàm *tk_wai_dev*.

Thời gian cho quá trình ghi là khác nhau cho mỗi thiết bị.

Trong trường hợpghi data chi tiết thiết bị, mỗi loại thiết bị sẽ quyết định *start* và *size* cho riêng data đó. Với data thuộc tính, *start* là một chỉ số data thuộc tính và *size* tính bằng byte. Data thuộc tính được chỉ định sẽ được ghi. Thông thường thì kích thước ghi ít nhất phải bằng với kích thước của data thuộc tính đó. Việc ghi nhiều thuộc tính thì không thể thực hiện được trong một quá trình ghi . Khi *size* = 0, thì việc ghi thực sự sẽ không thực hiện nhưng kích thước hiện tại của data sẽ được kiểm tra.

Việc có hay không một yêu cầu mới được chấp nhận trong khi quá trình đọc hay ghi chưa kết thúc là phụ thuộc vào device driver. Nếu yêu cầu mới không được chấp nhận thì nó sẽ được xếp trong hàng đợi, thời gian timeout cho việc đợi yêu cầu được chấp nhận được thiết lập trong *tmout* (có thể là TMO_POL hay TMO_FEVR).

E_ID	*dd* không hợp lệ hoặc chưa được mở
E_OACV	Chế độ mở không hợp lệ (không cho phép ghi)
E_LIMIT	Số lượng yêu cầu vượt quá giới hạn
E_ABORT	Quá trình bị bỏ qua

Những lỗi khác có thể có do device driver trả về

- **ID tk_swri_dev (ID dd, INT start, VP buf, INT size, INT *asize)**

Ghi đồng bộ. Hàm này tương đương đoạn code sau.

```
ER tk_swri_dev( ID dd, INT start, VP buf, INT size, INT *asize )
{
        ER er, ioer;
        er = tk_wri_dev(dd, start, buf, size, TMO_FEVR);
        if ( er > 0 ) {
                er = tk_wai_dev(dd, er, asize, &ioer, TMO_FEVR);
                if ( er > 0 ) er = ioerr;
        }
        return er;
}
```

- **ID tk_wai_dev (ID dd, ID reqid, INT *asize, ER *ioer, TMO tmout)**

dd	Device descriptor
reqid	Request ID
asize	Kích thước đọc hay ghi được
ioer	Lỗi IO trả về
tmout	Timeout (ms)

return code Request ID được hoàn thành hoặc lỗi

Hàm này thực hiện việc chờ cho yêu cầu *reqid* của thiết bị *dd* được thực hiện xong. Hàm này chỉ có tác dụng cho những yêu sầu đang được xử lý khi hàm được gọi, những yêu cầu nảy sinh sau khi gọi hàm sẽ không được đợi.

Khi có nhiều yêu cầu được xử lý dồng thời, thứ tự hoàn thành các yêu cầu không nhất thiết phải đúng với thứ tự chúng nảy sinh nhưng điều này là phụ thuộc vào thiết bị. Tuy nhiên, quá trình xử lý được đảm bảo thực hiện trong một sự liện tục mà kết quả thì phù hợp với thứ tự các yêu cầu. Ví dụ khi thực hiện việc đọc từ một đĩa, thứ tự có thể thay đổi như sau.

Thứ tự các block được yêu cầu 1 4 3 2 5

Thứ tự quá trình xử lý 1 2 3 4 5

Thứ tự trên cũng dể hiểu vì như vậy việc truy cập đĩa sẽ hiệu quả hơn.

Thời gian timeout được thiết lập trong *tmout*. Nếu lỗi timeout (E_TMOUT) được trả về, *tk_wai_dev* phải được gọi để hoàn thành vì quá trình xử lý các yêu cầu vẫn tiếp tục.

Khi *reqid* > 0 và *tmout* = TMO_FEVR cùng được thiết lập thì việc đợi sẽ không có timeout.

Nếu kết quả xử lý có lỗi (lỗi IO,v.v…) thì mã lỗi sẽ được trả về trong *ioer* chứ không phải là kết quả của hàm này. Mã trả về của hàm được sử dụng cho những lỗi mà chính việc đợi này không được xử lý hợp lệ. Khi lỗi được truyền trong mã trả về thì *ioer* không có nghĩa. Nếu lỗi được trả về, *tk_wai_dev* phải được gọi để hoàn thành vì quá trình xử lý các yêu cầu vẫn tiếp tục.

Các yêu cầu trong *reqid* và quá trình xử lý sẽ kết thúc nếu như có ngoại lệ xảy ra trong khi xử lý, điều này là tuỳ thuộc vào device driver. Tuy nhiên, khi *reqid* = 0 các yêu cầu sẽ không được bỏ qua mà được đối xử như là timeout. Trong trường hợp này E_ABORT được trả về.

Nhiều task không thể cùng đợi sự hoàn thành của cùng một request ID tại một thời điểm.

E_ID	*dd* không hợp lệ hoặc chưa được mở
	reqid không hợp lệ hoặc yêu cầu cho *dd*
E_OBJ	Có một task khác đang đợi cho *reqid*
E_NOEXS	Không có yêu cầu nào được thực hiện
E_TMOUT	Timeout (quá trình vẫn tiếp tục)
E_ABORT	Quá trình bị bỏ qua

Những lỗi khác có thể có do device driver trả về

- **INT tk_sus_dev (UINT mode)**

  mode Chế độ treo

  return code Số lượng yêu cầu vô hiệu treo hoặc lỗi

Thực hiện quá trình được chỉ định trong *mode*, sau đó gửi số yêu cầu vô hiệu treo hay lỗi trong mã trả về.

mode := ((TD_SUSPEND | [TD_FORCE]) | | TD_DISSUS | | TD_ENASUS | | TD_CHECK)

```
#define    TD_SUSPEND      0x0001 /* suspend */
#define    TD_DISSUS       0x0002 /* disable suspension */
#define    TD_ENASUS       0x0003 /* enable suspension */
#define    TD_CHECK        0x0004 /* get suspend disable request count */
#define    TD_FORCE        0x8000 /* forcibly suspend */
```

TD_SUSPEND Làm treo

Nếu trạng thái treo bị vô hiệu thì mã lỗi E_BUSY được trả về.

TD_SUSPEND | TD_FORCE Bắt buộc treo

Làm treo ngay cả khi trạng thái này bị vô hiệu.

TD_DISSUS Làm vô hiện trạng thái treo

TD_ENASUS Cho phép làm treo

Nếu số lượng yêu cầu cho phép treo nhiều hơn số yêu cầu vô hiệu treo thì sẽ không có chuyện gì sảy ra.

TD_CHECK Lấy số yêu cầu làm vô hiệu trạng thái treo

Quá trình làm treo được thực hiện theo những bước sau.

1. Xử lý trước khi bắt đầu treo trong mỗi subsystem

   tk_evt_ssy(0, TSEVT_SUSPEND_BEGIN, 0, 0)

2. Quá trình treo trong các thiết bị không đĩa
3. Quá trình treo trong các thiết bị đĩa
4. Xử lý sau khi treo trong mỗi subsystem

   tk_evt_ssy(0, TSEVT_SUSPEND_DONE, 0, 0)

5. SUSPEND Đi vào trạng thái treo

   tk_set_pow(TPW_DOSUSPEND)

E_BUSY Trạng thái ttreo bị vô hiệu

E_QOVR Số yêu cầu vô hiệu treo vượt quá giới hạn

- **ID tk_get_dev (ID devid, UB *devnm)**

devid Device ID

devnm Địa chỉ chứa tên của thiết bị

return code Device ID của thiết bị vật lý hay mã lỗi

Lấy tên của thiết bị được chỉ định trong *devid* và đặt kết quả trả về trong idevnm.

devid có thể là device ID của thiết bị vật lý hoặc thiết bị luận lý. *devnm* cần một không gian gồm (L_DEVNM + 1) byte hoặc lớn hơn.

E_NOEXS Thiết bị được chỉ định trong *devid* không tồn tại

- **ID tk_ref_dev (UB *devnm, T_RDEV *rdev)**

- **ID tk_oref_dev (ID dd, T_RDEV *rdev)**

devnm	Device name
dd	Device descriptor
rdev	Device information
return code	Device ID or error

```
typedef struct t_rdev {
        ATR devatr; /* device attributes */
        INT blksz; /* block size of device-specific data (-1: unknown) */
        INT nsub; /* subunit count */
        INT subno; /* 0: physical device: 1 to nsub: subunit number+1 */
        /* Implementation-dependent information may be added beyond this point.*/
} T_RDEV;
```

Lấy tthông tin của thiết bị được chỉ định trong *dennm* hay *dd* và đặt kết quả trong *rdev*.

- **INT   tk_lst_dev (T_LDEV *ldev, INT start, INT ndev)**

ldev	Location of registered device information (array)
start	Starting number
ndev	Number to acquire
return code	Remaining device registration count or error

```
typedef struct t_ldev {
        ATR devatr;  /* device attributes */
        INT  blksz;    /* device-specific data block size (-1: unknown) */
        INT  nsub;    /* subunits */
        UB   devnm[L_DEVNM]; /* physical device name */
        /* Implementation-dependent information may be added beyond this point.*/
} T_LDEV;
```

Lấy thông tin về thiết bị được đăng ký.

- **INT   tk_evt_dev (ID devid, INT evttyp, VP evtinf)**

devid	ID của thiết bị nhận sự kiện
evttyp	Loại sự kiện
evtinf	Tthông tin cho mỗi loại sự kiện
return code	Mã trả về của device driver hay lỗi

Gửi một sự kiện đến thiết bị.

Những sự kiện yêu cầu driver :

```
#define TDV_CARDEVT 1   /* PC Card event (see Card Manager) */
#define TDV_USBEVT    2   /* USB event (see USB Manager) */
```

E_NOEXS	Thiết bị được chỉ định không tồn tại

E_PAR	Những sự kiện quản lý thiết bị nội (evttyp < 0) không thể được chỉ định.

2.2 Screen driver

Screen có cấu trúc dữ liệu được tổ chức như sau

```
/* SCREEN data number */
  typedef   enum {
      /* common attribute */
      DN_SCRSPEC        = TDN_DISPSPEC,   /* DEV_SPEC    (R) */
      /* specific attributes: -100 to -199 for general purpose */
      DN_SCRLIST        = -100,     /* TC[]      (R) */
      DN_SCRNO          = -101,     /* W     (RW)*/
      DN_SCRCOLOR       = -102,     /* COLOR[]  (RW)*/
      DN_SCRBMP         = -103,     /* BMP     (R) */
      DN_SCRBRIGHT      = -200,     /* W     (RW)*/
      DN_SCRUPDFN       = -300,     /* FP        (R) */
      DN_SCRVFREQ       = -301,     /* W     (RW)*/
      DN_SCRADJUST      = -302,     /* ScrAdjust  (RW)*/
      DN_SCRDEVINFO     = -303,     /* ScrDevInfo    (R) */
      DN_SCRMEMCLK      = -304,     /* W     (RW)*/
      DN_SCRUPDRECT     = -305,     /* RECT        (RW)*/
      DN_SCRWRITE       = -306,     /* -         (W) */
  } ScrDataNo;
```

Tuy nhiên, trên Sh7760 thì không hỗ trợ một số thuộc tính dữ liệu sau

- DN_SCRNO (display mode) setting
- DN_SCRBRIGHT (screen brightness)
- DN_SCRADJUST (monitor timing adjustment)
- DN_SCRVFREQ (monitor vertical synchronization frequency)
- DN_SCEMEMCLK(Video-RAM clock setting)
- DN_SCRWRITE (direct screen writing)

Do đó chúng ta sẽ chỉ tìm hiểu những thuộc tính dữ liệu mà T-engine có hỗ trợ mà thôi

DN_SCRSPEC : lấy thông tin đặc tả của thiết bị (R)

```
data : DEV_SPEC devspec;
  typedef   struct  {
```

```
        H   attr;       /* device attribute */
        H   planes;     /* number of planes */
        H   pixbits;    /* number of pixel bit (boarder/effective) */
        H   hpixels;    /* number of horizontal pixels */
        H   vpixels;    /* number of vertical pixels */
        H   hres;       /* horizontal resolution */
        H   vres;       /* vertical resolution */
        H   color[4];   /* color information */
        H   resv[6];
    } DEV_SPEC;
```

DN_SCRLIST : lấy thông tin về các chế độ hiển thị

data TC list[];

-------ghi chú thêm--------

typedef UH TC ; // Tron character code

typedef unsigned short UH; // số nguyên 16bit không dấu

Danh sách các chế độ hiển thị có định dạng sau

<demarcator><display mode><demarcator><display mode>……..<0>

<demarcator> dùng để tách rời các chế độ hiển thị

<display mode> là một chuỗi kí tự để chỉ ra độ phân giải , độ sâu của màu sắc,…ví dụ như chuỗi sau “1024*768 256C”.

DN_SCRCOLOR : lấy hoặc thiết lập giá trị của bảng ánh xạ màu

data COLOR map[*]

Khi DEV_SPEC.attr.P = 0 thì không có bảng ánh xạ màu nào được sử dụng.

DN_SCRBMP : lấy thông tin về khoảng không gian để vẽ hình của thiết bị.

data BMP devbmp

-------ghi chú thêm---------

Cấu trúc của một BMP được định nghĩa như sau

```
    #define PLANES    1
    typedef struct Bitmap {
        UW  planes;         /* number of planes     */
        UH  pixbits;        /* number of pixel bits     */
        UH  rowbytes;       /* row bytes of a plane     */
```

```
    RECT  bounds;                /* coordinate specification    */
    UB    *baseaddr[PLANES];     /* base addresses      */
} BMP;
```

---

devbmp.baseaddr[*] trỏ tới địa chỉ bộ nhớ của của vùng ảnh được vẽ trên thiết bị.

Vùng bộ nhớ này thì có thể được truy xuất trực tiếp bởi device primitive (nhưng không thể được truy xuất trực tiếp từ ứng dụng) và vùng nhớ này chỉ tồn tại khi DEV_SPEC.attr.M = 1.

DN_SCRUPDRECT : cập nhật lại màn hình

data RECT r

------ghi chú thêm---------

Cấu trúc của một RECT được định nghĩa như sau

```
typedef struct point {
    H       x ;      /* horizontal coordinate */
    H       y;       /* vertical coordinate */
} PNT;
typedef   union  rect {
     struct  _rect {
        H       left;       /* left coordinate value */
        H       top;        /* upper coordinate */
        H       right;      /* right coordinate value */
        H       bottom;     /* bottom coordinate value */
     } c;
     struct {
        PNT    lefttop;     /* left-top point */
        PNT    rightbot;    /* right-bottom point */
     } p;
} RECT;
```

Chức năng này dùng để cập nhật màn hình ở đúng vị trí có sự thay đổi mà thôi .

Cơ chế hiển thị một hình ảnh lên trên screen

Để vẽ bất kì một hình ảnh nào lên trên màn hình thì ta phải có một mảng dữ liệu để thiết lập màu cho từng pixel tương ứng.

Ví dụ để hiển thị một kí tự bất kì thì ta phải vẽ kí tự đó dưới dạng các pixel riêng lẻ rồi dựa vào đó mà có một dữ liệu tương ứng. Trong thư viện mà chúng ta được hỗ trợ có định nghĩa sẵn cách vẽ

một kí tự với kích thước là 8x16 (có nghĩa là 8 bit theo chiều ngang và 16 bit theo chiều rộng). Chúng ta sẽ làm rõ vấn đề trên bằng cách hiển thị môt kí tự A lên màn hình với kích thước 8x16.

Dữ liệu tương ứng cho bảng giá trị bên dưới là

{ 0x00 , 0x10 , 0x10 , 0x28 ,0x28 , 0x28 , 0x44 , 0x44 , 0x44 , 0x7c , 0x82 , 0x82 , 0x82, 0x82 , 0x82 , 0x00 } /* A */

Cột Hàng	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0
3	0	0	0	1	0	0	0	0
4	0	0	1	0	1	0	0	0
5	0	0	1	0	1	0	0	0
6	0	0	1	0	1	0	0	0
7	0	1	0	0	0	1	0	0
8	0	1	0	0	0	1	0	0
9	0	1	0	0	0	1	0	0
10	0	1	1	1	1	1	0	0
11	1	0	0	0	0	0	1	0
12	1	0	0	0	0	0	1	0
13	1	0	0	0	0	0	1	0
14	1	0	0	0	0	0	1	0
15	1	0	0	0	0	0	1	0
16	0	0	0	0	0	0	0	0

Những vị trí mà tại đó pixel có giá trị là 1 thì có nghĩa là tại điểm đó sẽ được vẽ bằng màu được chỉ định còn ngược lại thì nó sẽ có màu là màu của màu nền

Vậy chúng ta hiện tại đã có những gì ???

Chúng ta hiện tại chỉ có một thư viện đồ họa hỗ trợ sẵn với tên là **graphic.h**, tuy nhiên bên trong chỉ định nghĩa một số hàm cơ bản như vẽ màu cho một pixel, vẽ đường thẳng, vẽ button, vẽ kí tự. Do đó nếu muốn hiển thị một file.bmp hay có đuôi mở rộng nào đó thì chỉ có một cách đó là phải giải nén file đó ra rồi lấy giữ liệu để hiển thị lên trên màn hình mà thôi. Và vì vậy mà mọi hoạt động đều phải do người lập trình tự quản lý lấy.

3. THỰC HÀNH

Đầu tiên chúng ta sẽ làm quen với các hàm có sẵn trong **graphic.h** để thấy rõ cách hoạt động của screen.

Sau đây là một chương trình sẵn cần có và chỉ cần thêm một số đoạn code vào để thử các hàm theo ý muốn.

```
#include <tk/tkernel.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include"graphic.h"
EXPORT ER main( INT ac, UB *av[] )
{
```

```
  if ( ac < 0 ) { /* in unload */
       return E_OK;
    }
     T_CTSK    ctsk;               /* create task information */

            /* task ID */
  ID   tskid;
       /* create task */

    ctsk.exinf   = NULL;               /* extend information */
    ctsk.tskatr  = TA_HLNG | TA_RNG0;  /* task attribute */
    ctsk.task    = main_task;               /* task start address */
    ctsk.itskpri = 80;               /* task start priority */
    ctsk.stksz   = 1024 * 10;           /* stack size */

    tskid = tk_cre_tsk( &ctsk );       /* create task */

    if (tskid < E_OK) goto ext;        /* fail to create task */

    /* start task */

    tk_sta_tsk(tskid, 0);              /* start task */

    /* exit */
ext:
  return E_OK;
}
----------------------------------------main_task.c-----------------------
-----------------------------
LOCAL BMP devbmp;
LOCAL RECT r_upd ;
LOCAL void init( void )
{
    /* fill total screen in white */
    fill_rect( devbmp.bounds.c.left, devbmp.bounds.c.top,
               devbmp.bounds.c.right, devbmp.bounds.c.bottom,
               WHITE, &devbmp );
/*enter your code here to test some functions in the graphic.h */
      /* update area is to be all */
    r_upd = devbmp.bounds;
}
EXPORT void main_task( INT stacd, VP exinf )
{
    UB      devnm[8];         /* device name */
    ID        dds;         /* device descriptor */
    INT     asiz;             /* read/write size */
    ER      ercd;             /* error code */

    /* open SCREEN device */
    strcpy( devnm, "SCREEN" );
    dds = tk_opn_dev( devnm, TD_UPDATE ); /* open device in update mode */
    if (dds < E_OK) goto ext;           /* fail to open */

   /* read device specific image area */
    ercd = tk_srea_dev( dds, DN_SCRBMP, &devbmp, sizeof(BMP), &asiz );
    if (ercd < E_OK || asiz != sizeof(BMP)) goto ext;

    /* color map setting */
    switch (devbmp.pixbits >> 8)
```

```
  { /* bit count per 1 pixel */
     case 8 :
     ercd = tk_swri_dev( dds, DN_SCRCOLOR, (VP)colormap, sizeof(colormap),
&asiz );
         if (ercd < E_OK || asiz != sizeof(colormap)) goto ext;
         break;
     }

     /* init_mainialize screen */
     init();

     /*vẽ combo_box được dùng cho phần bài tập */--------------
     /*
   TCompo_box  test;
    test.count = 4;
    test.name = (char**)malloc (4 * sizeof(char*));
    test.name[0]="Element1";
    test.name[1]="Element2";
    test.name[2]="Element3";
    test.name[3]="Element4";
    test.width = 240;
    test.height = 105;
    init_compo(&test);
    combo_box(&test,0,200,&devbmp);// 1/3 kich thuoc man hinh
    */
    //-------------------------------------------------------------------------
-
     tk_swri_dev(dds, DN_SCRUPDRECT, &r_upd, sizeof(RECT), &asiz);

     /* close SCREEN device */
     ercd = tk_cls_dev( dds, 0 );
     if (ercd < E_OK) goto ext;          /* fail to close */

     /* terminate and delete task */
ext:
     tk_exd_tsk();
}
```

3.2 Bài tập đề nghị

Thực hiện quá trình vẽ một combo_box có các thuộc tính được định nghĩa như sau

```
typedef struct {
   char** name; //mang chua cac chuoi ten
   int  count;//so thanh phan
   int  width;//do rong của combo_box
   int  height;//chieu cao của combo_box
   int  flag;//ghi lai gia' tri dang duoc cho.n
   int  display;//so^' thu tu cua thanh phan dang duoc cho.n (1 or 2 or 3)
   int  color;//luu gia' tri mau hien thi
}TCombo_box;
```

Nếu số thành phần của combobox là lớn hơn 3 thì chỉ hiển thị 3 thành phần còn các thành phần còn lại sẽ bị che khuất đi và chỉ được hiển thị khi người dùng chọn đến (quá trình xử lý sự kiện sẽ được học ở bài sau)

Sau đây là các hàm có sẵn

```
-------------------------------------------------------------------
LOCAL void init_compo(TCompo_box * compobox)
```

```
{
   compobox->flag = 0;
   compobox->display = 0;
   compobox->color =0;
}
-----------------------------------------------------------------------
Hãy hiện thực hàm bên dưới với x1, y1 là toạ độ bắt đầu để vẽ combo_box
LOCAL void combo_box(TCompo_box *compobox,int x1, int y1,BMP *p_bmp )
{
/* enter your code here */
}
```

Ghi chú

Vì bài tập này sẽ được dùng cho bài thí nghiệm tiếp theo nên sinh viên phải hoàn thành trước khi bước sang bài thí nghiệm kế tiếp

BÀI THỰC HÀNH SỐ 10

LOAD BITMAP , THIẾT BỊ KBPD VÀ CÁC HÀM XỬ LÝ SỰ KIỆN

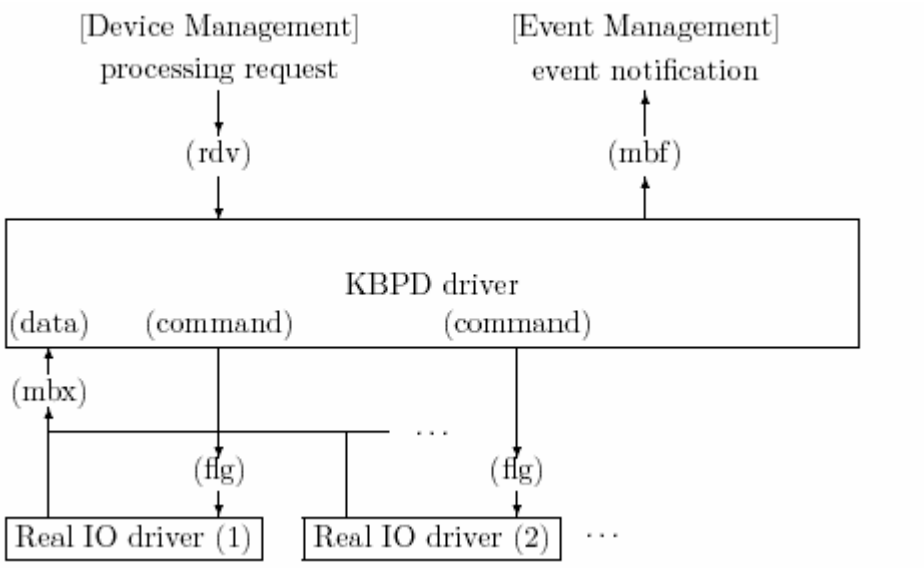
1. MỤC TIÊU

- Hiểu được cách xử lý các sự kiện từ các phím nhấn và của màn hình
- Load một hình ảnh đơn giản lên trên màn hình

2. LÍ THUYẾT

2.1 Thiết bị kbpd (key board / pointing device)

KBPD driver được hiện thực tách rời khỏi IO driver thật sự để mà có khả năng hỗ trợ cho nhiều loại keyboard và pointing device khác nhau. Do đó mà KBPD driver sẽ không phụ thuộc trực tiếp vào KB/PD device và không thực hiện quá trình truy xuất IO.



(rdv) Rendezvous cho việc chấp nhận các yêu cầu xử lý

(mbf) Message buffer cho việc báo hiệu những sự kiện xảy ra

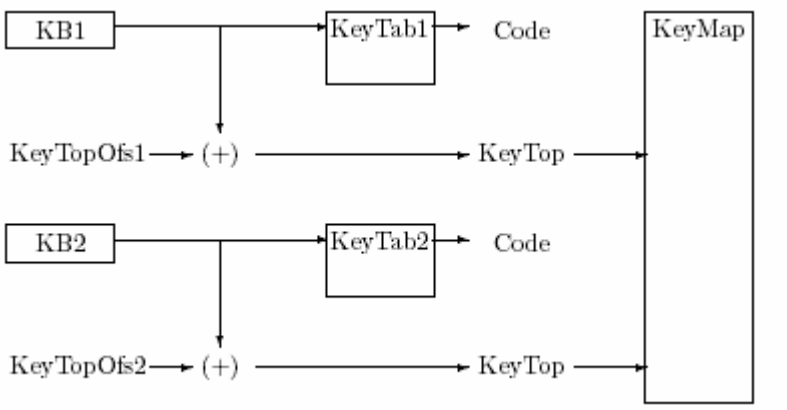
(mbx) Mail box cho việc lưu trữ dữ liệu

(flg) Cờ sự kiện được sử dụng với các yêu cầu

Một "real IO driver" gởi các tác vụ của thiết bị KB/PD tới mailbox cho KBPD ở một định dạng cố định . Các thiết bị khác nhau sẽ gởi các tác vụ này tới cùng một mailbox . KBPD driver có thể gởi các lệnh tới "real IO driver" bằng cách sử dụng cờ sự kiện và nếu như ở đây có nhiều IO driver thì các lệnh này sẽ được gởi cho tất cả các driver đó.

Đặc tả T-Engine cho phép việc kết nối 2 hay nhiều keyboard khác nhau và sử dụng chúng cùng một lúc.

- Mỗi loại keyboard sẽ có một KeyTab tương ứng với nó. Nếu có nhiều keyboard nhưng cùng một loại thì chúng sẽ được ánh xạ cùng một loại KeyTab.
- KeyTab được dùng để ánh xạ các phím nhấn trên keyboard và tạo ra các key-top code tương ứng.
- KeyMap chứa các mẫu bit mà tương ứng ở chế độ ON hay OFF tùy thuộc vào key-top code được sinh ra bởi keyboard cộng với KeyTopOfs. Những key-top code cộng với offset sẽ được ghi nhận như những sự kiện gởi tới phần mềm bên trên.



Sau đây là đặc tả cho kppd

```
/* KBPD attribute data number */
typedef   enum {
    DN_KPEVENT      = TDN_EVENT, /* message buffer fot event notification
                                    data: ID      RW */
    DN_KPINPUT      = -100,       /* mail box for input  data: IDR  */
    DN_KPSTAT       = -101,       /* KB/PD status  data: KPstat       RW */
    DN_KEYMAP       = -102,       /* key map data: KeyMap     R  */
    DN_KEYTAB       = -103,       /* key table   data: KeyTab     RW */
    DN_KEYMODE      = -104,      /* key behavior mode data: KeyModeRW */
    DN_PDMODE       = -105,       /* PD behavior mode     data: PdMode
                                    RW */
    DN_PDRANGE      = -106,       /* range data: PdRange RW */
    DN_PDSIM        = -107,       /* PD simulation data: W       RW */
    DN_PDSIMINH     = -108,       /* temporarily disabled PD simulation
                                    data: BOOL   RW */
    DN_KEYID        = -109,        /* key board ID data: UW          RW */
    DN_KPMETABUT    = -110,       /* mata key/button status
```

```
                                                    data: MetaBut[2]    W */
    DN_KEYDEF_S     = -200,         /* keyboard definition 1 (-200 to -327)   */
    DN_KEYDEF_E     = -327,         /* data: KeyDef     RW */
    DN_KEYDEF2_S    = -400,         /* keyboard definition 2 (-400 to -527)   */
    DN_KEYDEF2_E    = -527          /* data: KeyDef     RW */
} KPDataNo;
```

Tuy nhiên , chúng ta chỉ quan tâm tới thuộc tính DN_KEYMAP và DN_PDRANGE của KBPD vì chúng cần thiết cho ứng dụng của chúng ta. Còn các thuộc tính còn lại sẽ được tìm hiểu thêm tùy theo sở thích của người học.

DN_KEYMAP

Data KeyMap

```
#define  KEYMAX   256
typedef  UB KeyMap[KEYMAX/8]
```

KeyMap lưu giữ trạng thái hiện thời của các phím trên keyboard.

Khi một phím được nhấn thì key-top code bit tương ứng (từ 0 đến 255) sẽ được thiết lập là 1 còn khi một phím không được nhấn thì bit sẽ được xóa về 0. Nếu có 2 hay nhiều keyboard cùng loại được kết nối thì các phím nhấn sẽ có cùng một key-top code giống nhau (sau khi đã cộng với KeyTopOfs) nếu được nhấn cùng lúc và sẽ rơi vào trạng thái không xác định.

Do đó khi cần thiết để xử lý các sự kiện của bàn phím thì ta cần phải cập nhật trạng thái của các phím nhấn để xác định xem phím nào đã được nhấn rồi sẽ có hàm xử lý tương ứng.

Sau đây là một số scancode được định nghĩa sẵn cho các nút nhấn trên T-Engine

```
--------------------------
#define   SW1_1 0x6c      /*Center button – left*/
#define   SW1_2 0x6b      /*Center button – right*/
#define   SW1_3 0x69      /*Center button – up*/
#define   SW1_4 0x6a      /*Center button – down*/
#define   SW1_5 0x6d      /*Center button – enter*/
#define   SW2 0x6e        /*Left button */
#define   SW3 0x6f        /*Right button*/
--------------------------
```

DN_PDRANGE

```
typedef struct {
    H       xmax;       /* X coordinate maximum */
    H       ymax;       /* Y coordinate maximum */
} PdRange;
```

Xác định phạm vi xử lý của Pointing Device .

Để xử lý các thao tác của bàn phím thì ta dùng KeyMap để theo dõi các trạng thái của các phím , trong khi đó thì để xử lý Pointing Device thì chúng ta sẽ bắt những sự kiện được gởi về từ thiết bị này.

Định nghĩa cho kiểu dữ liệu của sự kiện như sau :

```
typedef struct {
    W       type;   /* event type    */
    UW      time;   /* event rise time      */
    PNT     pos;    /* PD position when event is sent  */
    EVDATA  data;   /* specific data of event     */
    UW      stat;   /* mata key/PD button status    */
} EVENT;
```

type : loại sự kiện được gởi về , có tất cả là 16 loại sự kiện

```
/*
 *    event type
 */
#define     EV_NULL    0     /* null event      */
#define     EV_BUTDWN  1     /* button down        */
#define     EV_BUTUP   2     /* button up     */
#define     EV_KEYDWN  3     /* key down     */
#define     EV_KEYUP   4     /* key up     */
#define     EV_AUTKEY      5     /* automatic key repeat*/
#define     EV_DEVICE  6     /* device event  */
#define     EV_RSV     7     /* reserved       */
#define     EV_APPL1       8     /* application event #1*/
#define     EV_APPL2       9     /* application event #2 */
#define     EV_APPL3       10    /* application event #3 */
#define     EV_APPL4       11    /* application event #4 */
#define     EV_APPL5       12    /* application event #5 */
#define     EV_APPL6       13    /* application event #6 */
#define     EV_APPL7       14    /* application event #7 */
#define     EV_APPL8       15    /* application event #8 */
```

Mỗi loại sự kiện thì có một "mask" tương ứng để cho phép loại sự kiện đó có thể được thực thi

Event	Type No.	Type Mask		
EV_NULL	0	—		
EV_BUTDWN	1	EM_BUTDWN	(0x0001)	
EV_BUTUP	2	EM_BUTUP	(0x0002)	
EV_KEYDWN	3	EM_KEYDWN	(0x0004)	
EV_KEYUP	4	EM_KEYUP	(0x0008)	
EV_AUTKEY	5	EM_AUTKEY	(0x0010)	
EV_DEVICE	6	EM_DEVICE	(0x0020)	
EV_RSV	7	EM_RSV	(0x0040)	(reserved)
EV_APPL1	8	EM_APPL1	(0x0080)	
:	:	:	:	
EV_APPL8	15	EM_APPL8	(0x4000)	

Tuy nhiên, ở đây cũng định nghĩa thêm 2 loại mask đặc biệt nữa đó là :

EM_ NULL 0x0000

EM_ ALL 0x7fff

time : dùng để chỉ ra khoảng thời gian mà sự kiện này xảy ra. Giá trị này được dùng để chỉ ra thứ tự giữa các sự kiện và khoảng cách giữa các sự kiện mà thôi mà không dùng để chỉ ra thời gian tuyệt đối. Đơn vị thời gian ở đây được tính bằng mili giây

pos : là vị trí mà tại đó sự kiện này được xảy ra

```
typedef  struct point {
      H        x;            /*  horizontal coordinate */
      H        y;            /*   vertical coordinate  */
} PTN;
```

data : dữ liệu của một sự kiện cụ thể và nó phụ thuộc vào kiểu sự kiện

```
typedef union {
    struct {        /* EV_KEYUP,EV_KEYDWN,EV_AUTKEY */
    UH   keytop;    /* key top code     */
    TC   code;    /* character code      */
  } key;
  struct {        /* EV_DEVICE      */
    H kind;    /* device event type(DE_xxx)*/
    H devno;      /* device number      */
  } dev;
  W info;         /* other data for event   */
} EVDATA;
```

stat : là một mẫu bit nó chỉ ra trạng thái của các button trên Pointing Device hay của một meta-key. Một mẫu bit có giá trị là 0 nghĩa là nó được giải phóng (OFF) còn nếu có giá trị la 1 thì có nghĩa là nó đang được nhấn (ON)

```
#define   ES_BUT      0x00000001 /* PD main button     */
#define   ES_BUT2     0x00000002 /* PD sub-button      */
#define   ES_ALPH     0x00000004 /* english lock key   */
#define   ES_KANA     0x00000008 /* Katakana lock key     */
#define   ES_LSHFT    0x00000010 /* left shift key     */
#define   ES_RSHFT    0x00000020 /* right shift key    */
#define   ES_EXT      0x00000040 /* extension key      */
#define   ES_CMD      0x00000080 /* command key        */
#define   ES_LLSHFT   0x00000100 /* left shift simple lock  */
#define   ES_LRSHFT   0x00000200 /* right shift simple lock    */
#define   ES_LEXT     0x00000400 /* extended simple lock*/
#define   ES_LCMD     0x00000800 /* command simple lock    */
#define   ES_TLSHFT   0x00001000 /* left shift temporary shift    */
#define   ES_TRSHFT   0x00002000 /* right shift temporary shift  */
#define   ES_TEXT     0x00004000 /* extended temporary shift   */
#define   ES_TCMD     0x00008000 /* command temporary shift */
#define   ES_HAN      0x00010000 /* half width key     */
#define   ES_KBSEL    0x00020000 /* select keyboard    */
#define   ES_NODSP    0x00200000 /* pointer unvisible      */
#define   ES_PDSIM    0x00C00000 /* PD simulation      */
```

Ở đây chúng ta chỉ cần biết được giá trị của ES_BUT là đủ rồi

2.2 Load một hình ảnh bitmap

Để thực hiện quá trình load bất cứ một file ảnh nào lên trên T-Engine thì chúng ta buộc phải hiểu rõ cấu trúc của file đó để có thể thực hiện quá trình đọc file và lấy dữ liệu cần hiển thị lên trên màn hình

- Giới thiệu : Định dạng file .bmp là một chuẩn file ảnh của Window 3.0 và cho một file DIB(device independent bipmap) về sau. Một ảnh BMP có thể có 2 màu trắng đen (1 bit trên 1 pixel) hoặc tới 24 bit trên 1 pixel (16.7 triệu màu). Nó có thể được nén nhưng rất ít khi được sử dụng trong thực tế và không được đề cập ở đây.
- Cấu trúc của file.bmp : Mỗi file BMP bao gồm 3 hoặc 4 phần như hình bên, phần đầu tiên là header của file, tiếp theo là header của ảnh bitmap (thông tin), nếu ảnh được đánh chỉ số màu thì tiếp theo là một bảng màu mẫu và cuối cùng và nội dung màu từng bit của bức ảnh. Vị trí dữ liệu của ảnh được chứa trong header file, và những thông tin như độ rộng , cao, số màu… được chứa trong nội dung của header ảnh.
    - Header file : Header của file bao gồm các trường sau đây :

```
typedef struct {
  char bfType[2];   /* Luôn là 'BM' để định nghĩa là file BMP */
  unsigned char bfSize[4]; /* Kích thước của file bằng byte*/
  unsigned char bfReserved1[2]; /* luôn là 0 */
  unsigned char bfReserved2[2]; /* luôn là 0 */
```

```
  unsigned char bfOffBits[4];    /* Độ dời đến vùng dữ liệu của ảnh
*/
} bmfh;
```

Độ dài của vùng header file là 14 byte.

- Header ảnh : Bao gồm những trường sau đây

```
typedef struct {
  unsigned char biSize[4]; /*Kích thước của vùng struct này : 40*/
  unsigned char biWidth[4];/*Chiều rộng */
  unsigned char biHeight[4];  /*Chiều cao */
  unsigned char biPlanes[2];    /* Số plane  */
  unsigned char biBitCount[2];  /* Số bit trên một pixel */
     unsigned char stuff1[16];
/*bao gồm : loại nén (4byte),kích thước ảnh (4), độ phân giải theo
chiều ngang (4) ,dọc(4) */
  unsigned char biClrUsed[4];        /* số lượng màu sử dụng */
unsigned char biClrImportant[4];   /* Số lượng màu quan trọng với
ảnh */
} bmih;
```

Độ dài của phần này là 40 byte.

- Ảnh 24 bit màu : Dữ liệu quan trọng trong việc đọc file này chính là ảnh 24 bit màu. Trong trường hợp này vùng dữ liệu của ảnh sẽ nằm ngay sau vùng header ảnh, tức là không có bảng màu. Mỗi pixel sẽ được biểu diễn bởi 3 byte tương ứng với blue, green, red theo thứ tự.
- Bảng màu : Nếu `bmih.biClrUsed <> 0` đứng ngay sau vùng header ảnh sẽ là bảng màu bao gồm `bmih.biClrUsed` phần tử, mỗi phần tử có cấu trúc sau :

```
typedef struct {
  unsigned char rgbBlue; /*xanh da trời*/
  unsigned char rgbGreen;   /* Xanh lá */
  unsigned char rgbRed;  /* Phần đỏ */
  unsigned char rgbReserved;  /*0 */
} rgbquad;
```

Ngay sau bảng màu sẽ là dữ liệu của ảnh. Trong trường hợp này, mỗi pixel được biểu diễn bởi một byte chính là chỉ số màu trong bảng màu.

3. THỰC HÀNH

3.1 Thiết bị kbpd

Với bài thí nghiệm 9, chúng ta đã vẽ được 1 compobox, bây giờ chúng ta sẽ thực hiện phần xử lý với các phím nhấn

- DOWN
- UP
- ENTER

Khi một phím được nhấn thì ta sẽ thể hiện sự thay đổi bằng cách vẽ lại một combobox mới. Điều quan trọng là sau khi vẽ lại các hình ảnh này chúng ta phải cập nhật lại giá trị của hình ảnh.

Chương trình của chúng ta gồm có một task chính, task đó sẽ kiểm tra sự thay đổi của các phím nhấn , khi có bất kì sự thay đổi nào trên các phím nhấn thì sẽ gọi một function để thực hiện việc cập nhật sự thay đổi đó.

```
//-------------------------------------main_task-----------------------------------------------
        LOCAL BMP devbmp;
        LOCAL RECT r_upd, rect1, rect2, rect3;
        INT PrevKeyStat;
        LOCAL int map_base[5]={2,3,4,5,6}; //các màu để vẽ khi một thành phần
        được chọn
        EXPORT void main_task( INT stacd, VP exinf )
        {
            UB      devnm[8];         /* device name */
            ID      dds, ddb;         /* device descriptor */
            PdRange range;            /* screen range */
            INT     asiz;             /* read/write size */
            ER      ercd;             /* error code */

            /* open SCREEN device */
            strcpy( devnm, "SCREEN" );
            dds = tk_opn_dev( devnm, TD_UPDATE ); /* open device in update
        mode */
            if (dds < E_OK) goto ext;           /* fail to open */
          /* read device specific image area */
            ercd = tk_srea_dev( dds, DN_SCRBMP, &devbmp, sizeof(BMP), &asiz
        );
            if (ercd < E_OK || asiz != sizeof(BMP)) goto ext;

            /* color map setting */
            switch (devbmp.pixbits >> 8) { /* bit count per 1 pixel */
            case 8 :
                ercd = tk_swri_dev( dds, DN_SCRCOLOR, (VP)colormap,
        sizeof(colormap), &asiz );
                if (ercd < E_OK || asiz != sizeof(colormap)) goto ext;
                break;
            }
          /* open kbpd device */
          strcpy( devnm, "kbpd" );
            ddb = tk_opn_dev( devnm, TD_UPDATE );/* open device */
            if (ddb < E_OK) goto ext;            /* fail to open */

            /* write PD range */
            range.xmax = devbmp.bounds.c.right;
            range.ymax = devbmp.bounds.c.bottom;
            ercd = tk_swri_dev( ddb, DN_PDRANGE, &range, sizeof(PdRange),
        &asiz );
            if (ercd < E_OK || asiz != sizeof(PdRange)) goto ext;

            /* init_mainialize screen */
            init();
            tk_swri_dev(dds, DN_SCRUPDRECT, &r_upd, sizeof(RECT), &asiz);
          PrevKeyStat = 0;

          /* khoi dong gia tri cho combobox */
          Tcombo_box  test;
          test.count = 4;
          test.name = (char**)malloc (4 * sizeof(char*));
          test.name[0]="Element1";
          test.name[1]="Element2";
          test.name[2]="Element3";
          test.name[3]="Element4";
          test.width = 240;
```

```
   test.height = 105;
   init_combo(&test);
   /* ve combobox */
   combo_box(&test,0,200,&devbmp);// 1/3 kich thuoc man hinh

    /* event loop */
    for(exit_flag = 0; !exit_flag; ) {
   keyproc( ddb,&test,0,200,dds); // kiem tra xem co' phim nao duoc
nhan hay khong?
       // và tương ứng với nó ta có thể vẽ lại  màn hình
       /* update screen */
  if (r_upd.c.left < r_upd.c.right && r_upd.c.top < r_upd.c.bottom)
      tk_swri_dev(dds, DN_SCRUPDRECT, &r_upd, sizeof(RECT), &asiz);
        tk_dly_tsk(10);
    }

    /* close kbpd device */
    ercd = tk_cls_dev( ddb, 0 );
    if (ercd < E_OK) goto ext;          /* fail to close */

    /* close SCREEN device */
    ercd = tk_cls_dev( dds, 0 );
    if (ercd < E_OK) goto ext;          /* fail to close */

    /* terminate and delete task */
ext:
    tk_exd_tsk();
}

--------------------------hàm quản lý bàn phím ----------------------
--------
// old keycode
#define O_SW1_1 0x8c
#define O_SW1_2 0x8b
#define O_SW1_3 0x89
#define O_SW1_4 0x8a
#define O_SW1_5 0x1c
#define O_KEY_UP   O_SW1_3
#define O_KEY_DOWN O_SW1_4
#define O_KEY_ENTER O_SW1_5
// new keycode
#define SW1_1 0x6c
#define SW1_2 0x6b
#define SW1_3 0x69
#define SW1_4 0x6a
#define SW1_5 0x6d
#define KEY_UP   SW1_3
#define KEY_DOWN SW1_4
#define KEY_ENTER SW1_5
EXPORT void keyproc( int dd ,Tcombo_box * combobox, int x1, int y1)
{
    UB KeyMap[KEYMAX/8];
    INT asiz, ercd, keystat;

#defineKeyOn(n)  (KeyMap[(n) >> 3] & (0x80 >> ((n) & 7)))

    /* read current key status */
    ercd = tk_srea_dev( dd, DN_KEYMAP, KeyMap, KEYMAX/8, &asiz );
    if (ercd < E_OK || asiz != KEYMAX/8) return;
```

```
    /* test for button press */
    keystat = 0;
     /* save current key status */
    if (KeyOn(KEY_ENTER) || KeyOn(O_KEY_ENTER)) keystat |= 0x1;  /*
Enter */
    if (KeyOn(KEY_DOWN) || KeyOn(O_KEY_DOWN)) keystat |= 0x2; /* Down
*/
    if (KeyOn(KEY_UP) || KeyOn(O_KEY_UP)) keystat |= 0x4; /* Up */

    /* button handling: Handle it if it is pressed in this time and
not pressed in last time */
    if ((keystat & 0x1) && !(PrevKeyStat & 0x1)) key_enter(); /*
Enter */
    else if ((keystat & 0x2) && !(PrevKeyStat & 0x2))
select_keydown(combobox,x1,y1,dds);/* keydown */
    else if ((keystat & 0x4) && !(PrevKeyStat & 0x4))
select_keyup(combobox,x1,y1,dds); /* keyup */

    /* save current key status */
    PrevKeyStat = keystat;
}
EXPORT void key_enter()
{
  exit_flag = 1; // thoat ra khoi chuong trinh
  fill_rect( devbmp.bounds.c.left, devbmp.bounds.c.top,
              devbmp.bounds.c.right, devbmp.bounds.c.bottom,
              WHITE, &devbmp );

}
EXPORT void select_keydown(Tcombo_box * combobox,int x1,int y1)
{
  combobox->flag = (combobox->flag + 1) % combobox->count;
//xac dinh thanh phan duoc cho.n
  if (combobox->display!=2) combobox->display = combobox->display+1;
  combo_box(combobox,x1,y1,&devbmp);
}
EXPORT void select_keyup(Tcombo_box * combobox, int x1, int y1)
{
  if(combobox->flag == 0) combobox->flag = combobox->count -1;
  else combobox->flag = combobox->flag -1;
  if (combobox->display!=0) combobox->display = combobox->display-1;
  combo_box(combobox,x1,y1,&devbmp);
}
------------------------------------------------------------------------
----------------------
-------Hàm vẽ lại combo_box tương ứng (source code cho cả bài 9)-----
--
LOCAL void combo_box(Tcombo_box *combobox,int x1, int y1,BMP *p_bmp )
{
  int vitrimau = rand()%5;
  while(vitrimau==combobox->color) vitrimau = rand()%5;
  combobox->color = vitrimau;
  int color = map_base[vitrimau];
  int stt1,stt2,stt3; //vi tri cua ca'c thanh phan trong combobox se~
duoc hien thi
  if(combobox->count <= 0) return;
  int width_of_e = 0;//kich thuoc cua 1 phan tu
  int vitritext_X = (int)(combobox->width/3);
```

```
  if(combobox->count >= 3)
  {

     width_of_e = (int)(combobox->height/3);
     int vitritext= (width_of_e - 16)/2;

     //----ve thanh phan thu 0-----
     rect1.c.left   = x1;
     rect1.c.right  = x1+combobox->width;
     rect1.c.top = y1+2;
     rect1.c.bottom= y1+ width_of_e;

     //-----ve thanh phan thu 1-----
     rect2.c.left   = x1;
     rect2.c.right  = x1+combobox->width;
     rect2.c.top = y1+ width_of_e + 2;
     rect2.c.bottom= y1+ width_of_e * 2 ;

     //-----ve thanh phan thu 2------
     rect3.c.left   = x1;
     rect3.c.right  = x1+combobox->width;
     rect3.c.top = y1+ width_of_e*2 + 2;
     rect3.c.bottom= y1+ combobox->height-2 ;

     //----------------------
     //-------------------
     if(combobox->display == 0)
     {
        stt1=combobox->flag;//thanh phan thu 0
        stt2=(combobox->flag +1 ) % combobox->count;
        stt3=(combobox->flag +2 ) % combobox->count;
        //----------------------
  fill_rect( rect1.c.left,
rect1.c.top,rect1.c.right,rect1.c.bottom,color, p_bmp );
  draw_text( combobox->name[stt1],rect1.c.left + vitritext_X ,
rect1.c.top + vitritext,255,p_bmp );
  fill_rect( rect2.c.left,
rect2.c.top,rect2.c.right,rect2.c.bottom,0, p_bmp );
  draw_text( combobox->name[stt2],rect2.c.left + vitritext_X ,
rect2.c.top + vitritext,255,p_bmp );
  fill_rect( rect3.c.left,
rect3.c.top,rect3.c.right,rect3.c.bottom,0, p_bmp );
  draw_text( combobox->name[stt3],rect3.c.left + vitritext_X ,
rect3.c.top + vitritext,255,p_bmp );
     }
     else if (combobox->display == 1)
     {
        stt1 = combobox->flag -1;
        stt2 = combobox->flag;//thanh phan thu 1
        stt3 = (combobox->flag +1 ) % combobox->count; //thanh phan
thu 2
        if (stt1 < 0 ) stt1 = stt1 + combobox->count;
        //----------------------
  fill_rect( rect1.c.left,
rect1.c.top,rect1.c.right,rect1.c.bottom,0, p_bmp );
  draw_text( combobox->name[stt1],rect1.c.left + vitritext_X ,
rect1.c.top + vitritext,255,p_bmp );
  fill_rect( rect2.c.left,
rect2.c.top,rect2.c.right,rect2.c.bottom,color, p_bmp );
```

```
  draw_text( combobox->name[stt2],rect2.c.left + vitritext_X ,
rect2.c.top + vitritext,255,p_bmp );
  fill_rect( rect3.c.left,
rect3.c.top,rect3.c.right,rect3.c.bottom,0, p_bmp );
  draw_text( combobox->name[stt3],rect3.c.left + vitritext_X ,
rect3.c.top + vitritext,255,p_bmp );
     }
     else if (combobox->display == 2)
     {
       stt1 = combobox->flag -2;
       stt2 = combobox->flag -1;
       stt3 = combobox->flag ;
       if (stt1 <0 ) stt1 = stt1 + combobox->count;
       if (stt2 <0 ) stt2 = stt2 + combobox->count;
  fill_rect( rect1.c.left,
rect1.c.top,rect1.c.right,rect1.c.bottom,0, p_bmp );
  draw_text( combobox->name[stt1],rect1.c.left + vitritext_X ,
rect1.c.top + vitritext,255,p_bmp );
  fill_rect( rect2.c.left,
rect2.c.top,rect2.c.right,rect2.c.bottom,0, p_bmp );
  draw_text( combobox->name[stt2],rect2.c.left + vitritext_X ,
rect2.c.top + vitritext,255,p_bmp );
  fill_rect( rect3.c.left,
rect3.c.top,rect3.c.right,rect3.c.bottom,color, p_bmp );
  draw_text( combobox->name[stt3],rect3.c.left + vitritext_X ,
rect3.c.top + vitritext,255,p_bmp );
     }
     ///----------------ve duong ngan cach giua cac  thanh phan
     ///------------ve duong bat ki
  fill_rect( rect1.c.left, rect1.c.top-2,rect1.c.right,rect1.c.top
,4, p_bmp );
  fill_rect( rect1.c.left, rect1.c.bottom,rect1.c.right,rect2.c.top
,4, p_bmp );
  fill_rect( rect2.c.left, rect2.c.bottom,rect2.c.right,rect3.c.top
,4, p_bmp );
  fill_rect( rect3.c.left,
rect3.c.bottom,rect3.c.right,rect3.c.bottom+2 ,4, p_bmp );
  fill_rect( rect1.c.left, rect1.c.top -2,rect1.c.left
+2,rect3.c.bottom +2 ,4, p_bmp );
  fill_rect( rect1.c.right -2, rect1.c.top -
2,rect1.c.right,rect3.c.bottom +2 ,4, p_bmp );
  }
// hãy hiện thực thêm phần nếu như ở combo_box có it hơn 3 thành phần
}
```

3.2 Load bitmap

Vì quá trình đọc file bitmap phức tạp do đó ở đây chỉ thực hiện quá trình load thử một file ảnh lên màn hình mà thôi.

Sau đây là source code cho các hàm đọc và load bitmap .Công việc còn lại của các bạn là hãy dùng các hàm này để load một file ảnh bất kì lên trên màn hình.

----------Các định nghĩa----------

```
typedef struct {
  char bfType[2];
  unsigned char bfSize[4];
  unsigned char bfReserved1[2];
  unsigned char bfReserved2[2];
```

```
  unsigned char bfOffBits[4];
} bmfh;

typedef struct {
  unsigned char biSize[4];
  unsigned char biWidth[4];
  unsigned char biHeight[4];
  unsigned char biPlanes[2];
  unsigned char biBitCount[2];
  unsigned char stuff1[16];
  unsigned char biClrUsed[4];
  unsigned char biClrImportant[4];
} bmih;
-----------Các hàm đọc file và load hình bitmap ----------------------
---
EXPORT unsigned long ** pixels0 = NULL;
EXPORT int h=0;
EXPORT int w=0;
EXPORT ID bmp_tid = -1;
LOCAL RECT r_ext, r_next, r_pic;
LOCAL BMP devbmp;
LOCAL RECT r_upd;

LOCAL int charArray2Int(unsigned char * b, int s, int f)
{
  int ret = 0;
  int i;
  int shift = 0;

  for (i = s; i <= f; i++) {
    ret = ret | ((b[i] & 0xff) << shift);
    shift += 8;
  }
  return ret;
}

EXPORT unsigned long ** read_BMP(char* file_name,int *h,int *w)
{
  unsigned long ** pixels = NULL;
  FILE *  bfile;
  bfile = fopen(file_name, "r");
  if (bfile == NULL) {
    fprintf(stderr, "Unable to open specified file: %s\n",
file_name);
    fprintf(stderr, "Errno: %d, errstring: %s\n", errno,
strerror(errno));
    Kfree(pixels);return NULL;
  }

  bmfh * hdr = (bmfh *)Kmalloc(sizeof(bmfh));
  fprintf(stderr, "sizeof(bmfh) = %d\n", sizeof(bmfh));

  // Read BMP
  int nread;
  nread = fread(hdr, 1, sizeof(bmfh), bfile);
  if (nread != sizeof(bmfh)) {
    fprintf(stderr, "Unable to read hdr, returned %d (%s)\n",
      nread, strerror(errno));
    Kfree(pixels);return NULL;
```

```
  }
  int size;
  int offset;
  size = charArray2Int(hdr->bfSize, 0, 3);
  offset = charArray2Int(hdr->bfOffBits, 0, 3);
  fprintf(stderr, "Type: %c%c, Size: %d, Offset: %d\n",
    hdr->bfType[0], hdr->bfType[1], size, offset);

  bmih * info = (bmih *)Kmalloc(sizeof(bmih));
  nread = fread(info, 1, sizeof(bmih), bfile);
  if (nread != sizeof(bmih)) {
    fprintf(stderr, "Unable to read info, returned %d (%s)\n",
      nread, strerror(errno));
    Kfree(pixels);return NULL;
  }
  int biSize;
  int height;
  int width;
  int bitsInPixel;
  int biClrUsed;
  biSize = charArray2Int(info->biSize, 0, 3);
  height = charArray2Int(info->biHeight, 0, 3);
  width = charArray2Int(info->biWidth, 0, 3);
  bitsInPixel = charArray2Int(info->biBitCount, 0, 1);
  biClrUsed = charArray2Int(info->biClrUsed, 0, 3);
  fprintf(stderr,
    "Size: %d, Width: %d, Height: %d, BitCount: %d, ClrUsed: %d\n",
    biSize, width, height, bitsInPixel, biClrUsed);

  int i, j;
  unsigned long * rgbColorTable = NULL;
  if (bitsInPixel < 24) {
    if (biClrUsed == 0) {
      biClrUsed = 1 << bitsInPixel;
    }
    rgbColorTable = (unsigned long *)Kmalloc(biClrUsed *
sizeof(int));
    for (i = 0; i < biClrUsed; i++) {
      nread = fread(&(rgbColorTable[i]), 1, 4, bfile);
      //fprintf(stderr, "rgbcolor[%d] = %8.8lx\n", i,
rgbColorTable[i]);
    }
  }

  if (bitsInPixel != 24 && bitsInPixel != 8) {
    fprintf(stderr, "Unsupported bit size of %d\n", bitsInPixel);
    Kfree(pixels);return NULL;
  }

  int bytesPerPixel = bitsInPixel / 8;
  int bytesPerRow = width * bytesPerPixel;
  bytesPerRow = (bytesPerRow & 0x3) ?
    ((bytesPerRow >> 2) + 1) << 2 : bytesPerRow;
  int pad = bytesPerRow - (width * bytesPerPixel);


  pixels = (unsigned long **)Kmalloc(height * sizeof(unsigned long
*));
  unsigned char pixelbytes[4];
```

```
  for (i = 0; i < height; i++) {
    // Start by Kmallocing the memory for a row
    pixels[i] = (unsigned long *)Kmalloc(width * sizeof(unsigned
long));
    for (j = 0; j < width; j++) {
      nread = fread(pixelbytes, 1, bytesPerPixel, bfile);
      if (nread != bytesPerPixel) {
   fprintf(stderr, "Unable to read pixel bytes, returned %d (%s)\n",
     nread, strerror(errno));
   Kfree(pixels);return NULL;
      }
      if (bytesPerPixel == 3) {
   pixels[i][j] = (unsigned long)charArray2Int(pixelbytes, 0, 2);
      }
      else {
   if (pixelbytes[0] >= biClrUsed) {
     fprintf(stderr, "pixel color index out of range: %d at
(%d,%d)\n",
       pixelbytes[0], i, j);
     Kfree(pixels);return NULL;
   }
   pixels[i][j] = rgbColorTable[pixelbytes[0]];
      }
    }

    // Read the pad bytes that word align the row of pixels

    nread = fread(pixelbytes,1, pad, bfile);
  }
   *w=width;
   *h=height;
   return pixels;

}
EXPORT void show_BMP(unsigned long ** pixels,int x,int y,int
height,int width)
{
   if(height>r_upd.c.bottom) // exceed the range
     height = r_upd.c.bottom ;
   if(width>r_upd.c.right) // exceed the range
     width = r_upd.c.right ;
   int i,j;
     if(pixels!=NULL)
   {
     for(i=0;i<height;i++)
     {
       for(j=x;j<width+x;j++)
       {
         int c=pixels[i][j-x];
         draw(j,height+y-i,c,&devbmp);
       }
     }
   }
   r_upd.c.left=x;
   r_upd.c.right = x+width;
   r_upd.c.top=y;
   r_upd.c.bottom = y+height;
```

```
}
```

Hãy viết một hàm main.c để thực hiện quá trình đọc bitmap rồi load file ảnh đó lên màn hình. Dùng các hàm đã được cung cấp sẵn bên trên.