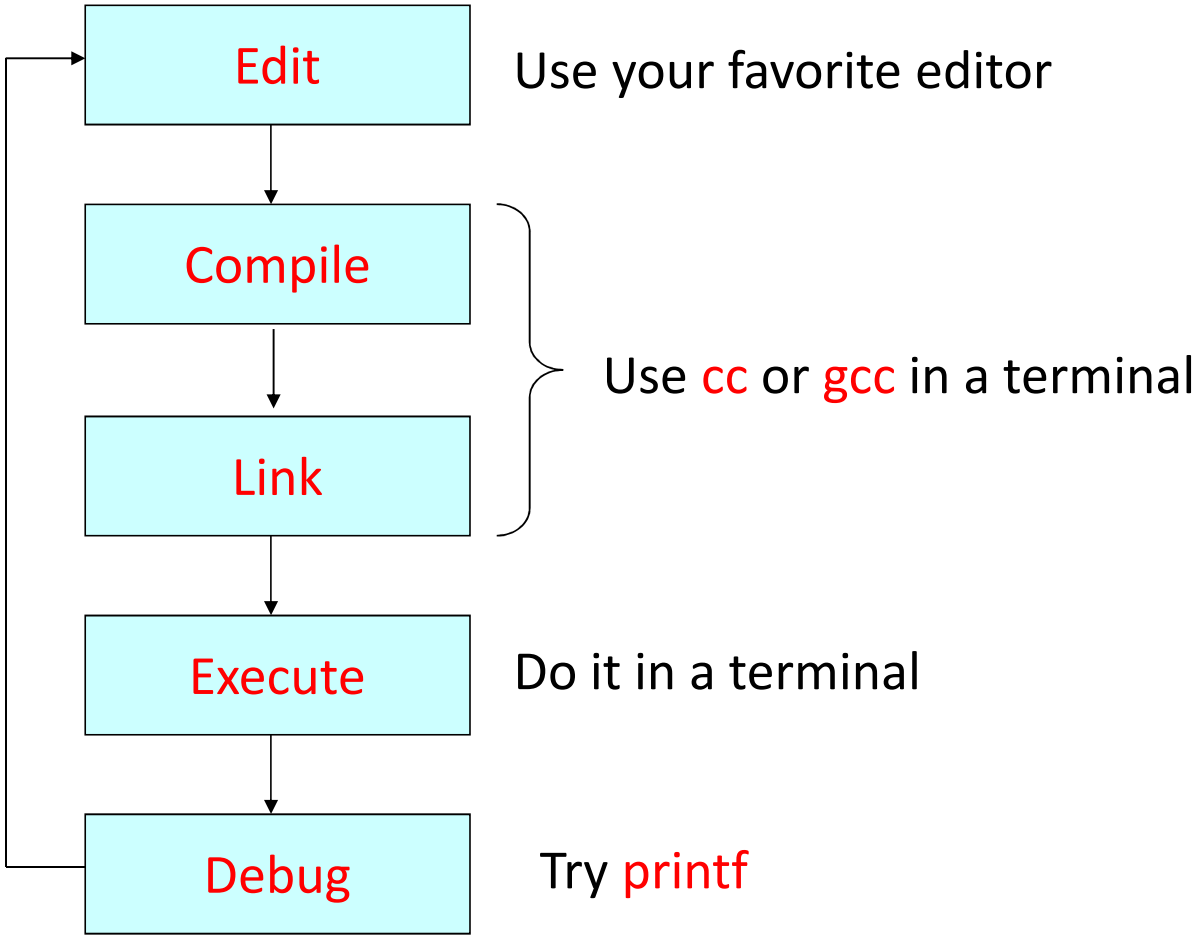
Network Programming: Unix Programming

Li-Hsing Yen
NYCU

Ver. 1.0.0

程式發展流程



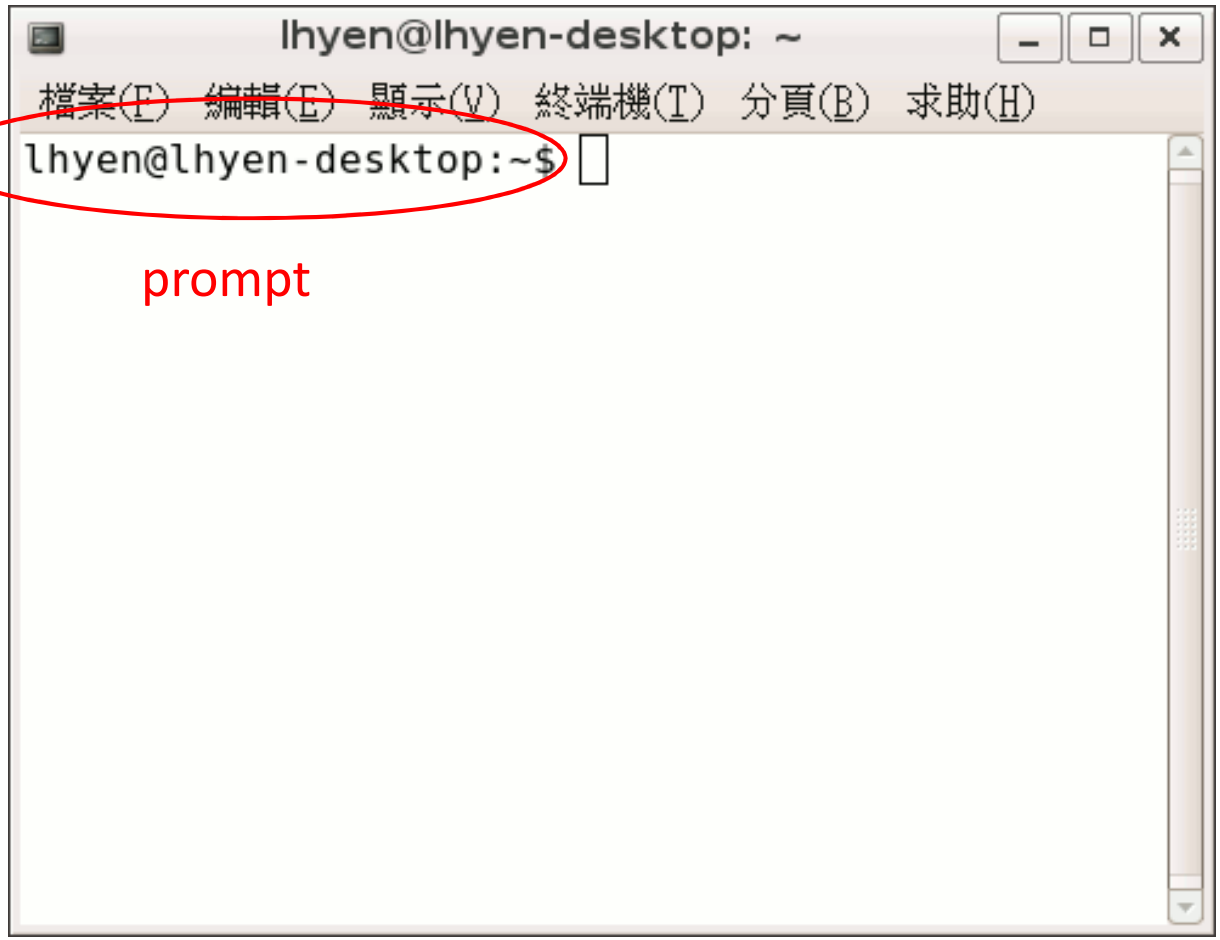
Use your favorite editor

Use cc or gcc in a terminal

Do it in a terminal

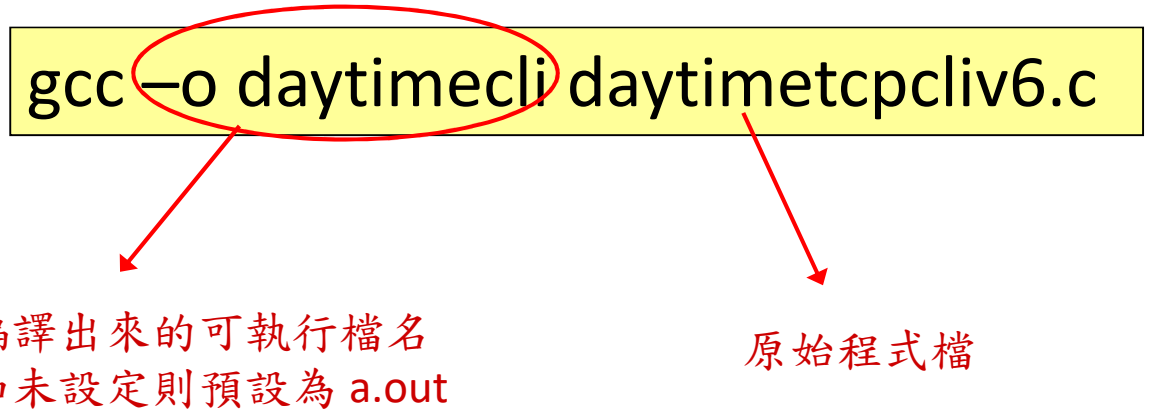
Try printf

Terminal


lhyen@lhyen-desktop: ~
檔案(F) 編輯(E) 顯示(V) 終端機(T) 分頁(B) 求助(H)
lhyen@lhyen-desktop:~$
prompt

Compiling Your C Program

- 可用 cc 或 gcc 來編譯

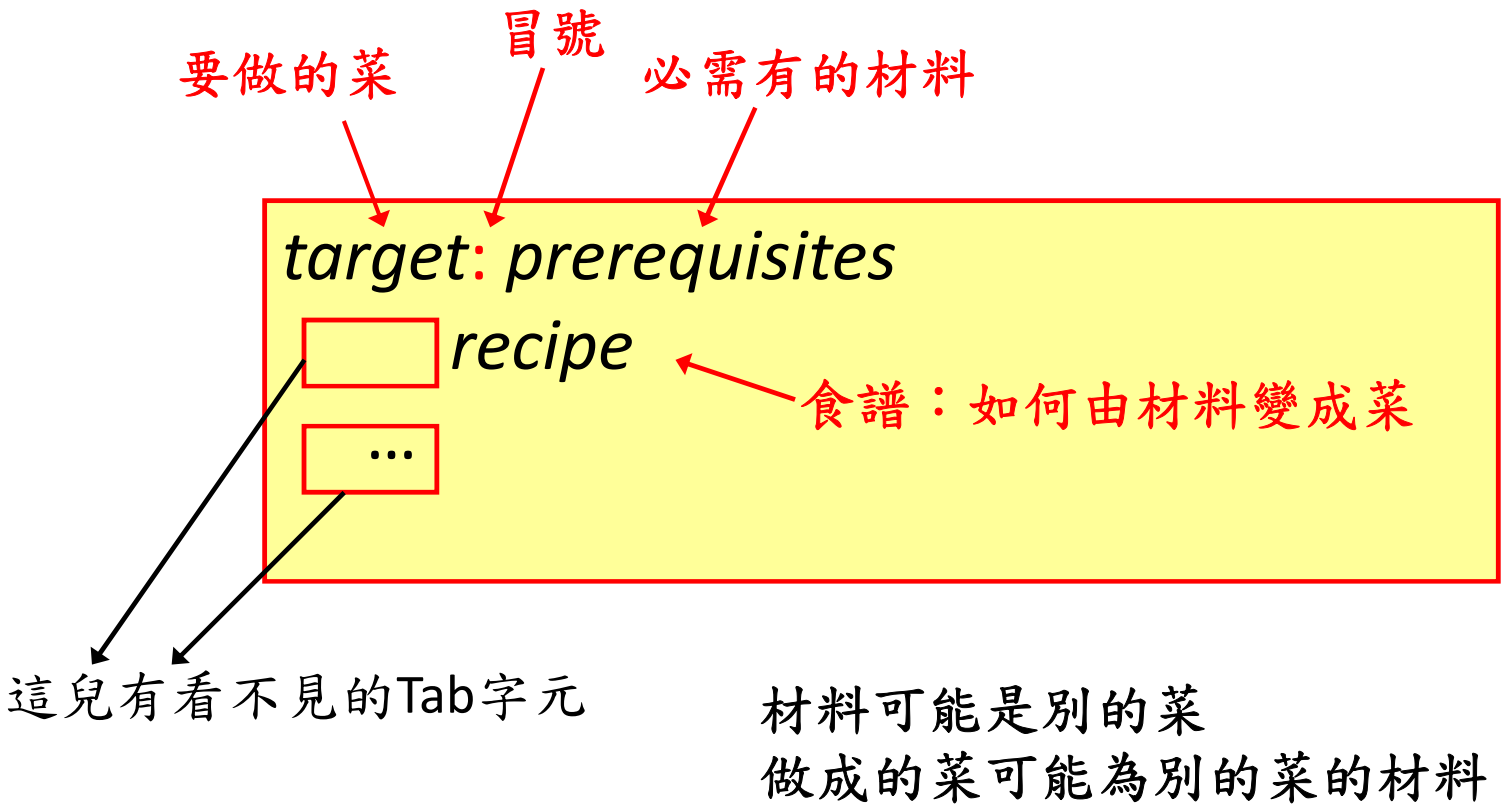


尚可藉由command line argument設定library路徑名稱，編譯最佳化程度，
其它要一併編譯或連結的程式碼與目的檔等

使用make

- 許多大型的程式專案由許多程式檔、標頭檔(header file)與函式庫(library)組成
- 如果其中某個檔案有更新就全部編譯連結是很沒有效率的
- make工具程式幫助我們管理程式專案，追蹤那些檔案從上次編譯後有變更過而需要重新編譯
- make由文字檔`Makefile`讀取相關資訊。`Makefile`與程式檔置於同一目錄

Basic Elements in `Makefile`



Variables in `Makefile`

```
CC = gcc
CFLAGS = …
LIBS = ../libunp.a -lpthread
…
```

Define variables

```
prog: prog.o
   $(CC) $(CFLAGS) -o $@ prog.o $(LIBS)
```

= gcc

目前的target

intro/Makefile

Makefile

```
include ../Make.defines

PROGS = daytimetcpcli daytimetcpcli1 daytimetcpcli2 daytimetcpcli3 \
        daytimetcpsrv daytimetcpsrv1 daytimetcpsrv2 daytimetcpsrv3 \
        daytimetcpcliv6 daytimetcpsrvv6 \
        byteorder

all: ${PROGS}

daytimetcpcli:   daytimetcpcli.o
        ${CC} ${CFLAGS} -o $@ daytimetcpcli.o ${LIBS}

daytimetcpcli1:  daytimetcpcli1.o
        ${CC} ${CFLAGS} -o $@ daytimetcpcli1.o ${LIBS}

daytimetcpcli2:  daytimetcpcli2.o
        ${CC} ${CFLAGS} -o $@ daytimetcpcli2.o ${LIBS}

daytimetcp
        ${                                      LIBS}
```

加進上層目錄的Make.defines內容

沒寫完換行

定義PROGS變數

主要目標為 all。只要冒號後面的檔案有更新的，all就要更新(all其實是假目標)

只要daytimetcpcli2.o比較新，daytimetcocli2就要更新

目前的目標(daytimetcpcli2)

更新daytimetcocli2的方法

Make.defines

- 上頁的Makefile會用到的Make.defines檔定義了與作業系統與編譯器有關的變數
- 例: Linux系統上

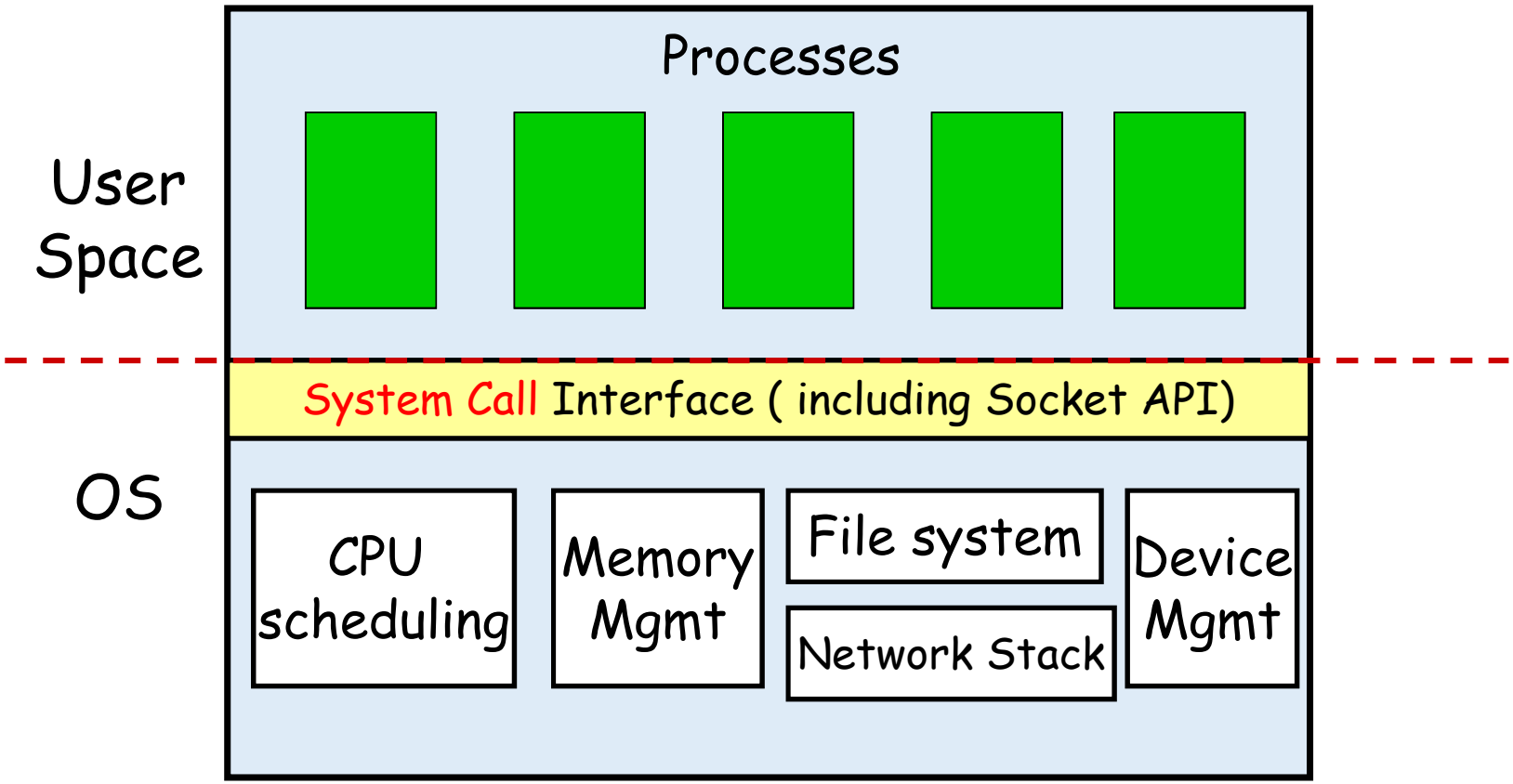
```
...
CC = gcc
CFLAGS = -I../lib -g -O2 -D_REENTRANT -Wall
LIBS = ../libunp.a -lpthread
LIBS_XTI = ../libunpxti.a ../libunp.a -lpthread
RANLIB = ranlib
...
```

Makefile 中會用到

configure

- 我們寫的c程式可以跨各種unix平台(BSD, Linux, SunOS, etc.)編譯
- 但不同的作業系統設定及編譯器所使用的參數略有差異，理論上我們應該為每個可能的作業系統平台撰寫不同版本的Makefile
- 安裝時執行的configure程式幫忙我們設定目前所用系統的正確參數(寫到Make.defines)中

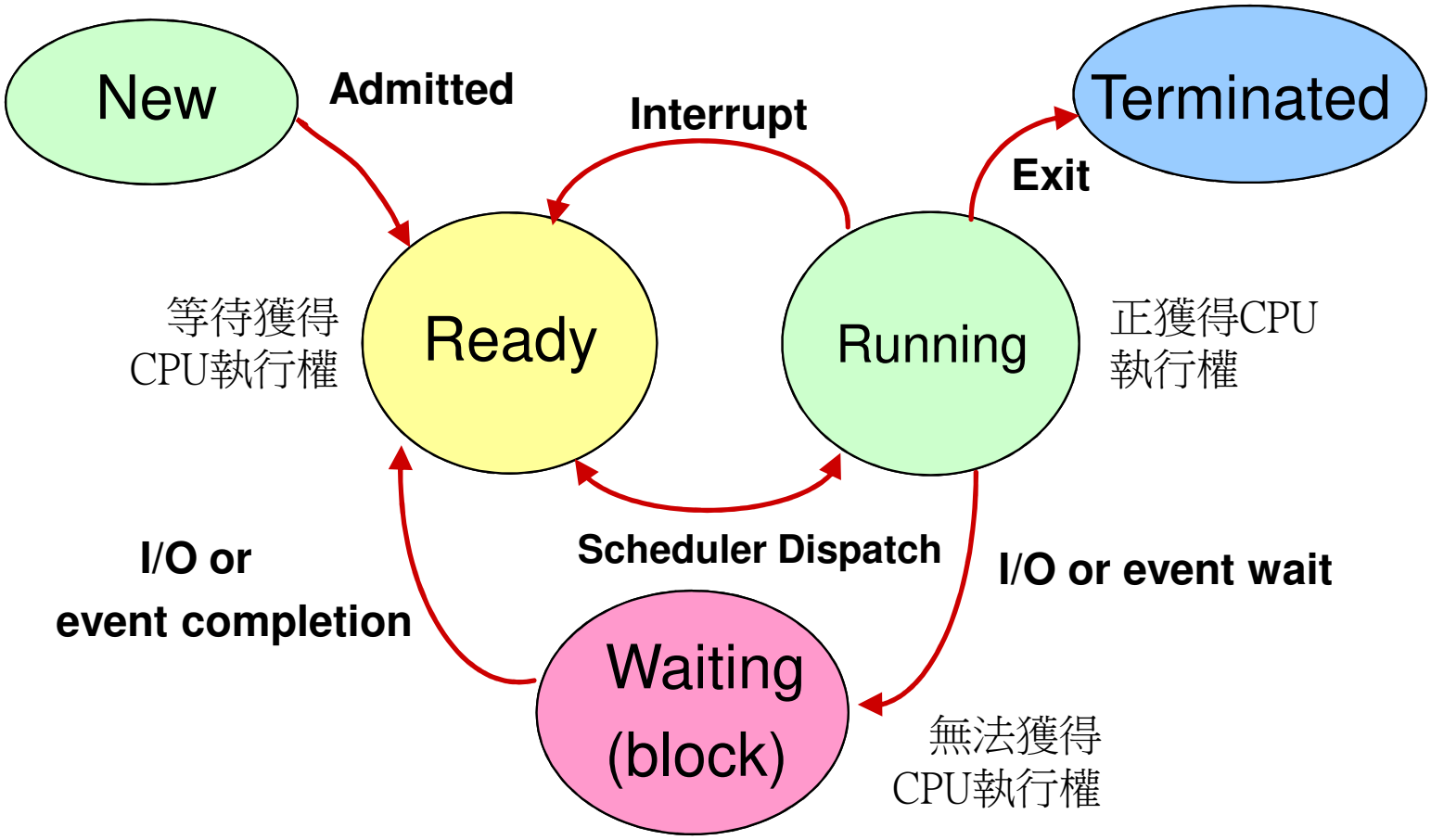
應用程式與作業系統關係



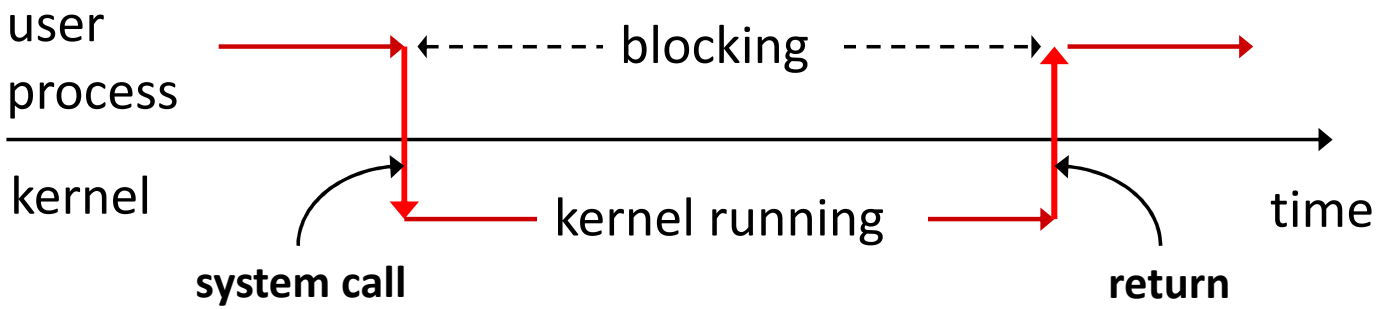
System Call

- 作業系統直接管理硬體、介面、檔案系統等，將其稱作系統資源 (system resource)
- 應用程式不能直接存取系統資源（安全與強固性的考量）
- 應用程式要使用系統資源時，須透過system call請求作業系統服務
- 對應用程式的程式撰寫者而言，這些system call就像是普通的 library function

Process State Diagram



Blocking Process



- 當user process呼叫system call時，如果此system call不能立即return（如需等待I/O），則此process處於block (waiting)狀態
- 處於block狀態的process不會被分配CPU執行時間

前景與背景執行

- 應用程式預設在前景(foreground)執行
  - 可由keyboard(或其它stdin裝置)取得使用者的輸入
  - 可輸出資料到螢幕(或其它stdout裝置)
- 某些程式(如Server程式)不需和console使用者互動，故可設定為背景(background)執行
  - 不由keyboard(或其它stdin裝置)取得使用者的輸入
  - 不輸出資料到螢幕(或其它stdout裝置)
  - 主要是和Client程式互動