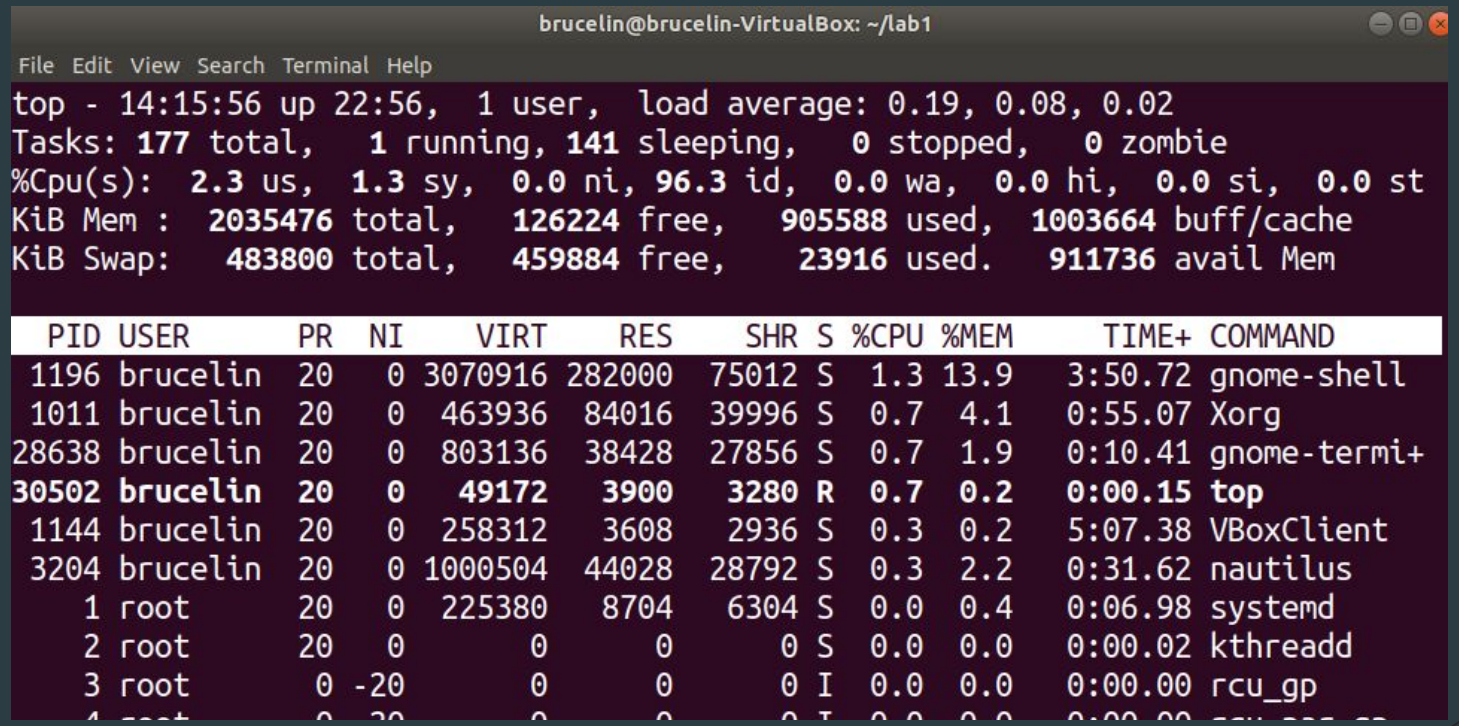
Introduction to Parallel Computing (II) Process Concepts

Cheng-Hung Lin

Linux processes

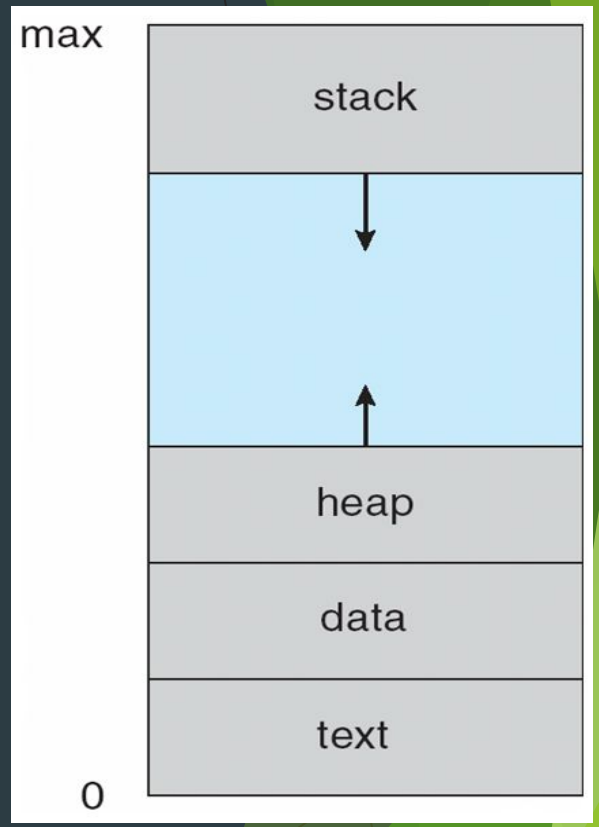
- Process is a unit of running program
- Each process has some information, like process ID, owner, priority, etc
- Example: Output of “top” command



行程(Process)

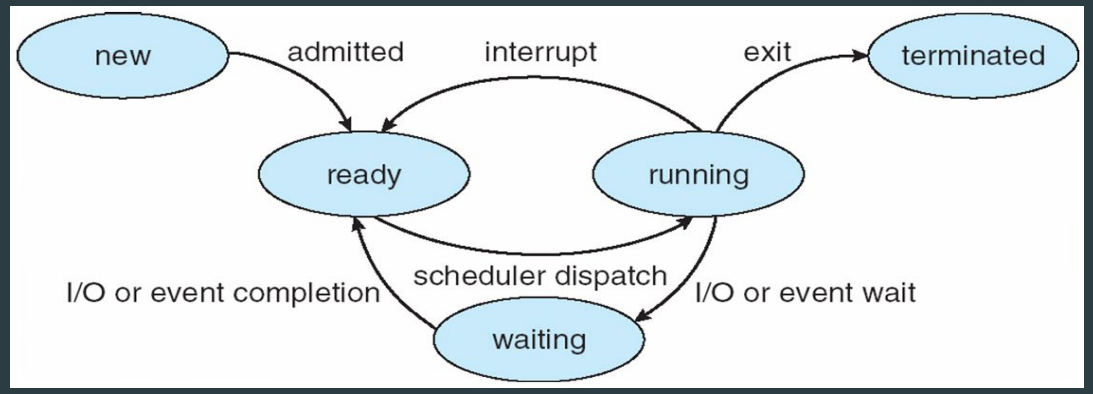
- 行程指的是正在執行的程式。行程不只是程式碼(有時也稱為本文區，text section)而已。它還包含代表目前運作的程式計數器 (Program counter)數值和處理器的暫存器內容。

- 行程還包括存放暫用資料 (譬如:**副程式的參數**、**返回位址**，及暫時性變數)的**行程堆疊 (stack)**，以及包含整體變數的**資料區間 (data section)**。行程也包含**堆積 (heap)**，堆積就是在行程執行期間動態配置的記憶體，行程記憶體結構如圖。



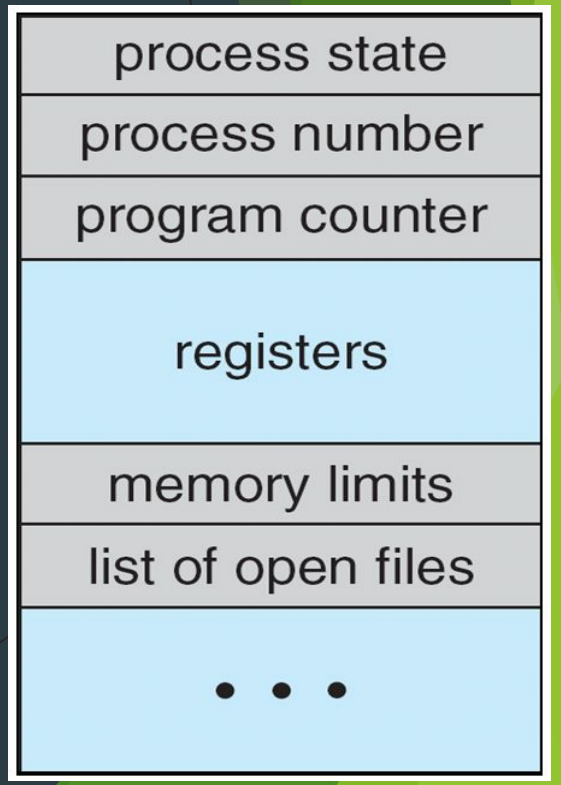
行程狀態(Process states)

- 行程在執行時會改變其狀態。行程的狀態 (state)部份是指該行程目前的動作，每一個行程可能會處於以下數種狀態之一:
  - **新產生 (new):**該行程正在產生中。
  - **執行 (running):**該行程正在執行。
  - **等待(waiting):**等待某件事件的發生(譬如輸出入完成或接收到一個信號)。
  - **就緒 (ready):**該行程正等待指定一個處理器。
  - **結束 (terminated):**該行程完成執行。



行程控制表

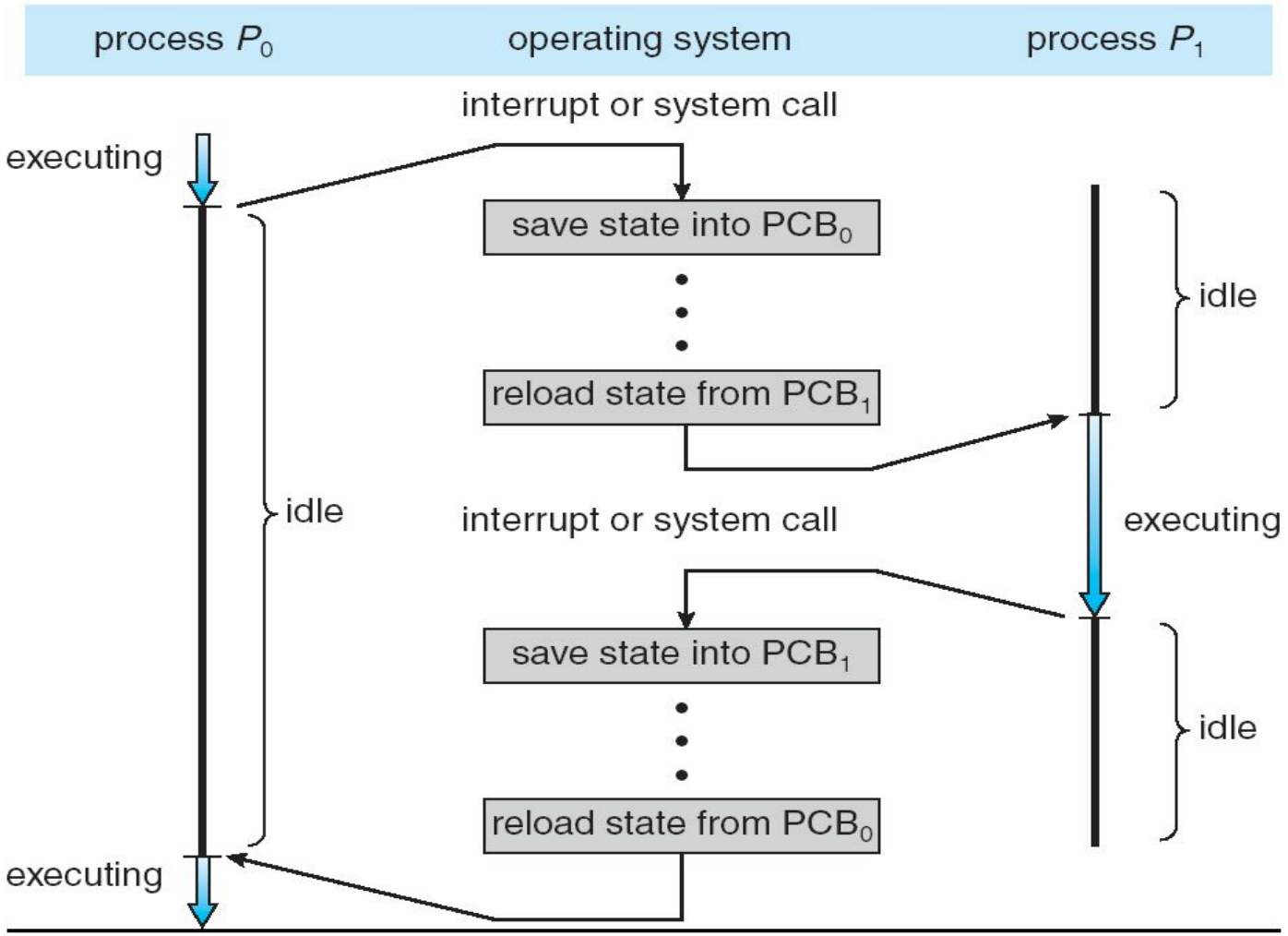
- 每一個行程在作業系統之中都對應著一個**行程控制表 (Process control block (PCB)** 或稱**任務控制表 (task control block)**, 如圖。
- 行程控制表 (PCB)記載所代表的行程之相關資訊, 包括:
  - 行程狀態:可以是new、ready、running、waiting或halted等。
  - 程式計數器(program counter):指明該行程接著要執行的指令位址。



行程控制表(cnt.)

- CPU暫存器:其數量和類別，完全因電腦架構而異。包括累加器 (accumulator)、索引暫存器 (index register)、堆疊指標 (stack pointer)以及一般用途暫存器 (general-purpose register)等，還有一些狀況代碼 (condition code)。當中斷發生時，這些狀態資訊以及程式執行計數器必須儲存起來，以便稍後利用這些儲存的資訊，使程式能於中斷之後順利地繼續執行 。
- CPU排班法則相關資訊:包括行程的優先順序 (Priority)、排班佇列(scheduling queue)的指標以及其它的排班參數。
- 記憶體管理資訊:這些資訊包括如基底暫存器(base register)和限制暫存器(limit register)，分頁表(Page table)值的資訊統所使用的記憶系統區段表(segment table)。
- 帳號資訊:包括了CPU和實際時間的使用數量、時限、帳號工作或行程號碼。
- 輸入/輸出狀態資訊:包括配置給行程的輸入/輸出裝置、開啟檔案(list of open files)的 等等。

CPU switch from process to process



行程排班(Process Scheduling)

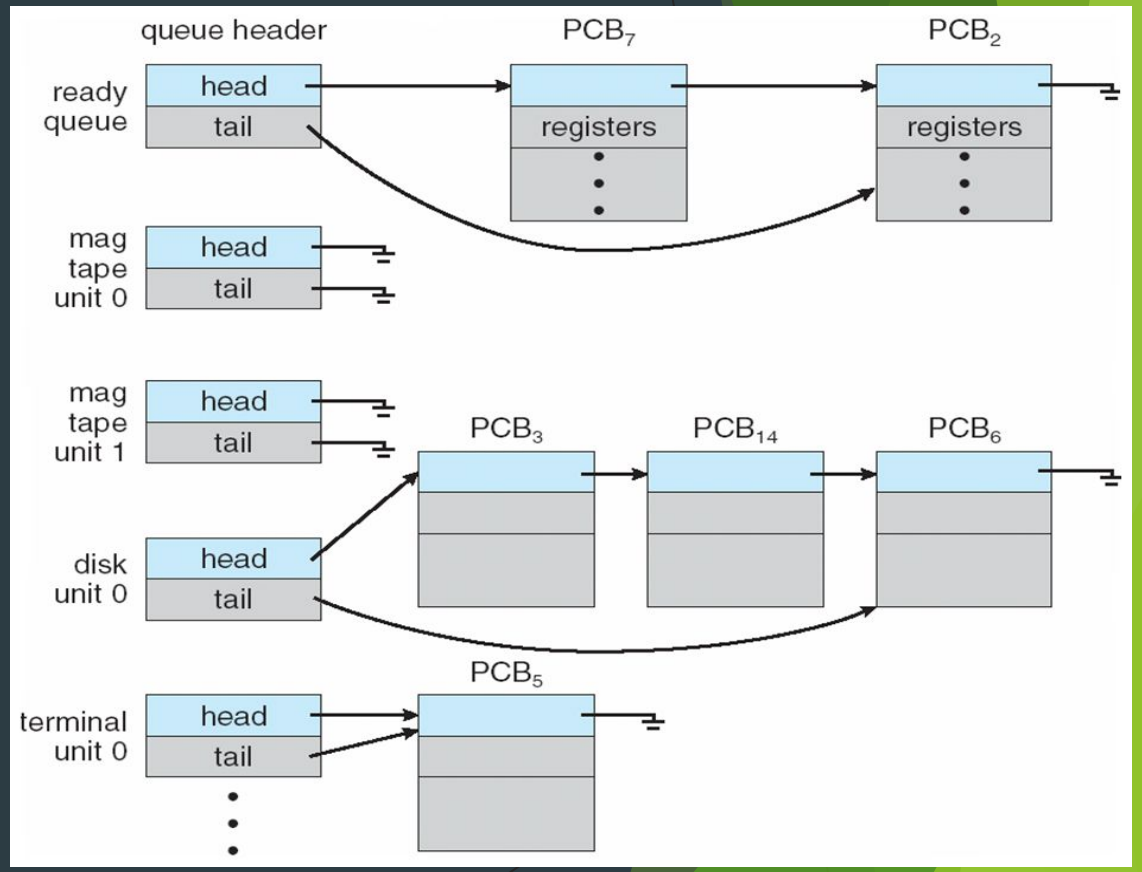
- 多元程式規劃(multiprogramming)系統的主要目的，是隨時有一個行程在執行，藉以提高CPU的使用率。分時(time sharing)系統的目的是將CPU在不同行程之間不斷地轉換，以便讓使用者可以在自己的行程執行時**與它交談**。

- 為了達到這個目的，行程排班程式(process scheduler)為CPU選擇一個可用的行程(可能由一組可用行程)。

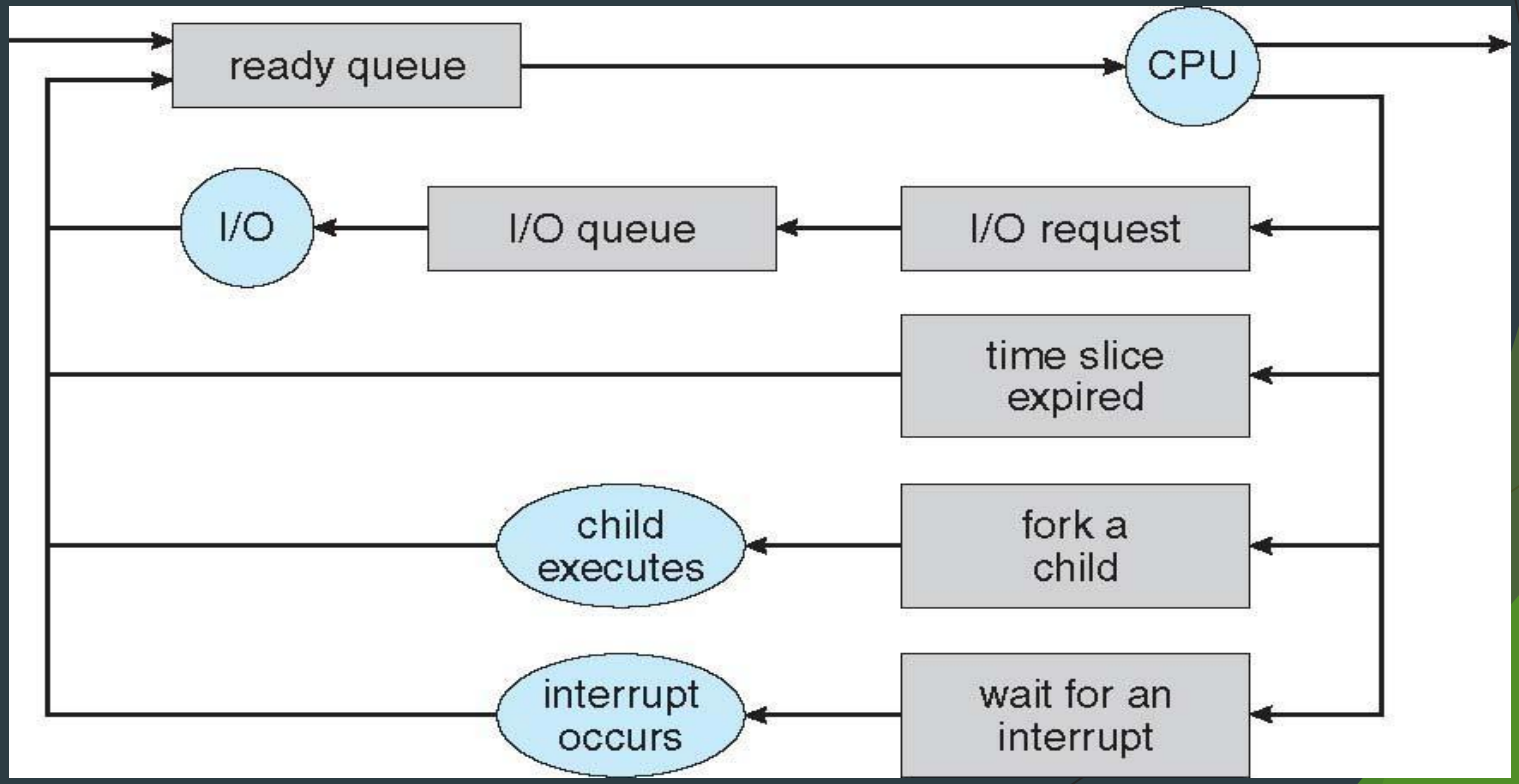
- 單一處理器系統，不可能有一個以上的行程同時執行。如果有多個行程，其它的都必須在旁邊等待一直到CPU有空，才可能重新排列。

排班佇列(scheduling queue)

- 行程進入系統時，是放在工作佇列(job queue)之中，此佇列是由所有系統中的行程所組成。位於主記憶體中且就緒等待執行的行程是保存在一個所謂就緒佇列 (ready queue)的串列。
- 這個佇列一般都是用鏈接串列的方式儲存。在ready queue前端保存著指向這個串列的第一個和最後一個PCB的指標。



- 一個新的行程最初是置於ready queue中。就一直在ready queue中等待，直到選來執行或被分派 (dispatched)。一旦這個行程配置CPU並且進行執行，則會有若干事件之一可能發生:
  - 行程可發出I/0要求，然後置於一個I/0佇列中。
  - 行程可產生出一個新的子行程並等待後者的結束。
  - 行程可強行地移離CPU(如用中斷的結果一樣)，然後放回ready queue中。



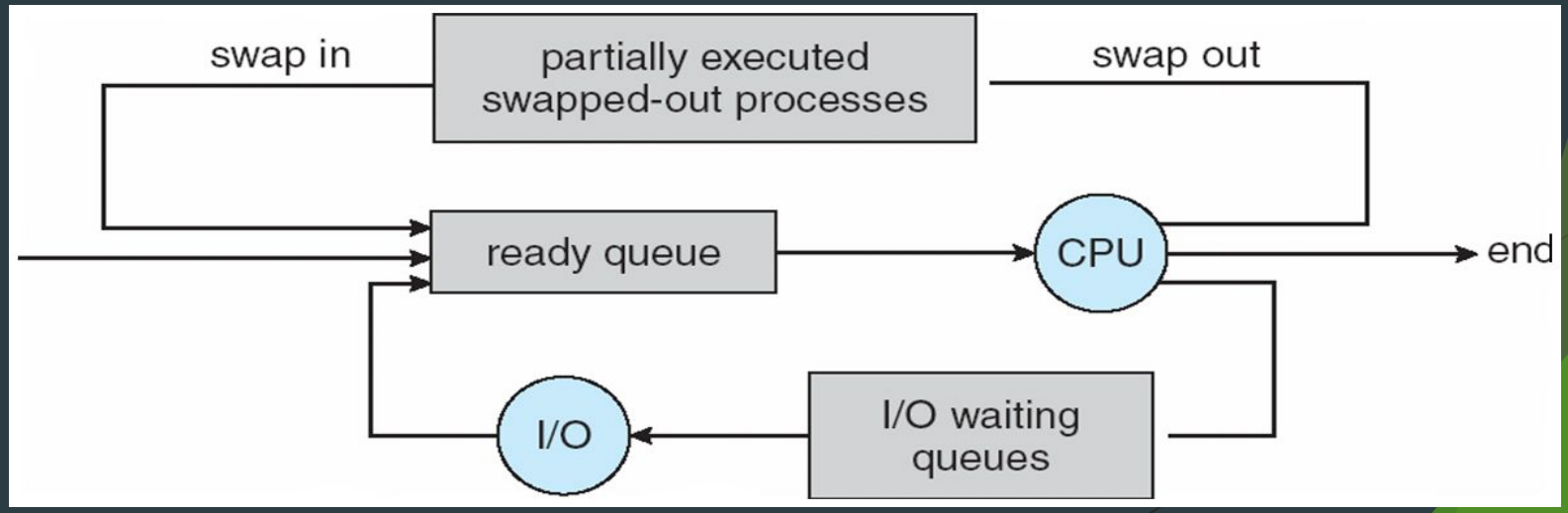
Schedulers

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU
- The long-term scheduler controls the **degree of multiprogramming**

- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

排班程式

- 一個行程在它整個生命期裏將在各個不同的排班佇列間遷移。作業系統必須按排班次序從這些佇列選取行程。行程的選取將由適當的排班程式 (scheduler)來執行。
- 分時系統，可能會採用一種額外的、間接方式來排班。
- 中程排班程式 (medium-term scheduler)背後的最主要觀念就是有時後可以將行程從記憶體中有效地移開(並且從對CPU的競爭中移開)、並減低多元程式規劃的程度。
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**



內容轉換Context Switch

- 中斷使作業系統改變CPU目前的工作而執行核心常式, 這樣的作業常發生在一般用途系統上。當中斷發生時, 系統需要儲存目前在CPU上執行行程的內容 (context), 所以當作業完成時, 它可以還原內容, 本質就是暫停行程, 再取回行程。
- 轉換 CPU至另一項行程時必須將舊行程的狀態儲存起來, 然後再載入新行程的儲存狀態。這項任務稱為內容轉換(context switch)。
- 內容轉換是額外負擔(overhead); 系統在做內容轉換時, 沒辦法作任何有用的工作
  - 作業系統與PCB越複雜, 內容轉換所需的時間越長
- 內容轉換所需的時間取決於硬體支援
  - 有些硬體提供多組暫存器可以允許多個內容轉換同時執行

行程的操作

- 系統中的各個行程可以並行 (concurrently) 地執行，而且也要能動態地產生或刪除。作業系統必須提供行程產生和結束的功能。

行程的產生

- 一個行程的執行期間，可以利用產生行程的系統呼叫來產生數個新的行程。原先的行程就叫做父行程 (Parent process)，而新的行程則叫做子行程(children process)。
- 每一個新產生的行程可以再產生其它的行程，這可以形成一幅行程樹(tree of processes)。
- 每個行程有自己的 process identifier (pid)

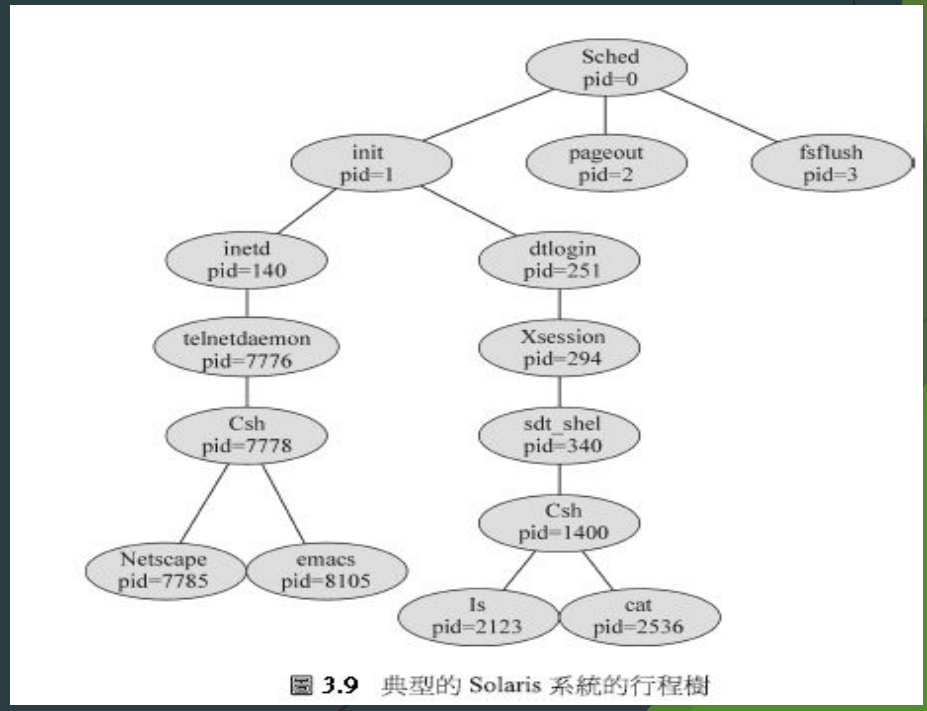


圖 3.9 典型的 Solaris 系統的行程樹

Process Creation

- 當一個父行程產生一個子行程, 有兩種執行的可能性
  - 父行程與子行程同時執行
  - 父行程等待直到子行程結束
- 對於資源分享, 有三種可能性
  - 父行程與子行程共享所有資源
  - 子行程分享父行程的部分資源
  - 父行程與子行程不共享資源

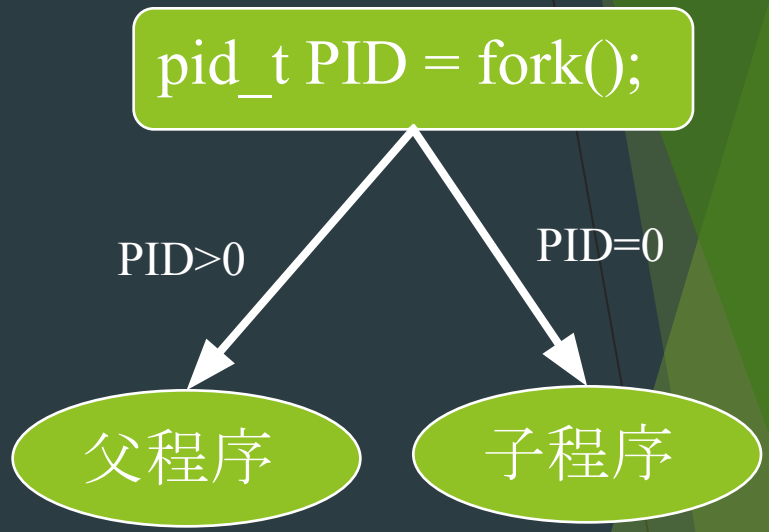
Process Creation (Cont)

- 記憶體空間(address-space)
  - 子行程複製父行程的記憶體空間
  - 子行程負載新的程式

- UNIX examples
  - **fork** system call creates new process
  - A new process is created by the fork() system call.
  - The new process consists of a copy of the address space of the original process.
  - **exec** system call used after a **fork** to replace the process' memory space with a new program

fork 的函數雛型 (man page 定義)

- #include <unistd.h>
- pid_t fork(void);
- fork() 可能會有以下三種回傳值 :
  - -1 : 發生錯誤
  - 0 : 代表為子程序
  - 大於 0 : 代表為父程序, 其回傳值為子程序的 Process ID
  - 注意 : 其回傳值是 pid_t , 不是 int 哦 !



Example

- 程式在呼叫 fork 建立新的行程之後，讓父行程與子行程兩個行程都輸出一樣的文字訊息。
- 使用 gcc 編譯後，直接執行：

```
gcc -o fork1 fork1.c
./fork1
```

- 父行程與子行程都執行同一個 printf，各輸出一行文字訊息，所以結果就會是兩行一樣的文字訊息

```
Hello world!
Hello world!
```

```
// fork1.c
#include <stdio.h>
#include <unistd.h>
int main() {
  // 建立子行程
  fork();

  // 從這裡開始變成兩個行程

  // 兩個行程執行同樣的程式
  printf("Hello world!n");

  return 0;
}
```

重複建立子行程

- 由於每次呼叫 fork 時, 就會讓程式行程的數量變成原來的兩倍, 所以如果重複呼叫 fork 的話, 形成數量就會以指數的速度增長。
- 編譯與執行:

```
gcc -o fork2 fork2.c
./fork2
Hello world!
Hello world!
Hello world!
Hello world!
```

- 執行兩次 fork 之後, 就會產生四個程式行程

```
#include <stdio.h>
#include <unistd.h>
int main() {
  // 建立子行程
  fork();

  // 建立子行程的子行程
  fork();

  // 所有行程都輸出一樣的訊息
  printf("Hello world!n");

  return 0;
}
```

區分父行程與子行程

- 這個範例是示範使用 fork 的傳回值來判斷父行程與子行程, 讓兩個行程分別執行不同的程式碼:
- 編譯與執行:

gcc -o fork3 fork3.c

./fork3

- 由於我們無法預測作業系統會先執行哪一個行程, 所以這個程式的輸出有可能是

Parent process!

Child process!

- 或是

Child process!

Parent process!

```
#include <stdio.h>
#include <unistd.h>
int main() {
  pid_t pid;

  // 建立子行程
  pid = fork();

  if (pid == 0) {
    // 子行程
    printf("Child process!n");
  } else if (pid > 0) {
    // 父行程
    printf("Parent process!n");
  } else {
    // 錯誤
    printf("Error!n");
  }

  return 0;
}
```

多行程程式的變數與資料

- 不同行程之間的變數與資料都是互相獨立的, 所以在其中一個行程中更改變數中的資料, 並不會影響另外一個行程:
- 編譯與執行:

gcc -o fork4 fork4.c

./fork4

Parent has x = 0

Child has x = 2

- 雖然這裡的 x 是一個全域變數(global variable), 但是兩個行程互相獨立, 所以不會互相影響。

```
#include <stdio.h>
#include <unistd.h>
int x = 1; // 建立一個變數
int main() {
  pid_t pid;

  pid = fork();

  if (pid == 0) {
    // 子行程的變數
    printf("Child has x = %dn", ++x);
  } else if (pid > 0) {
    // 父行程的變數
    printf("Parent has x = %dn", --x);
  } else {
    printf("Error!n");
  }

  return 0;
}
```

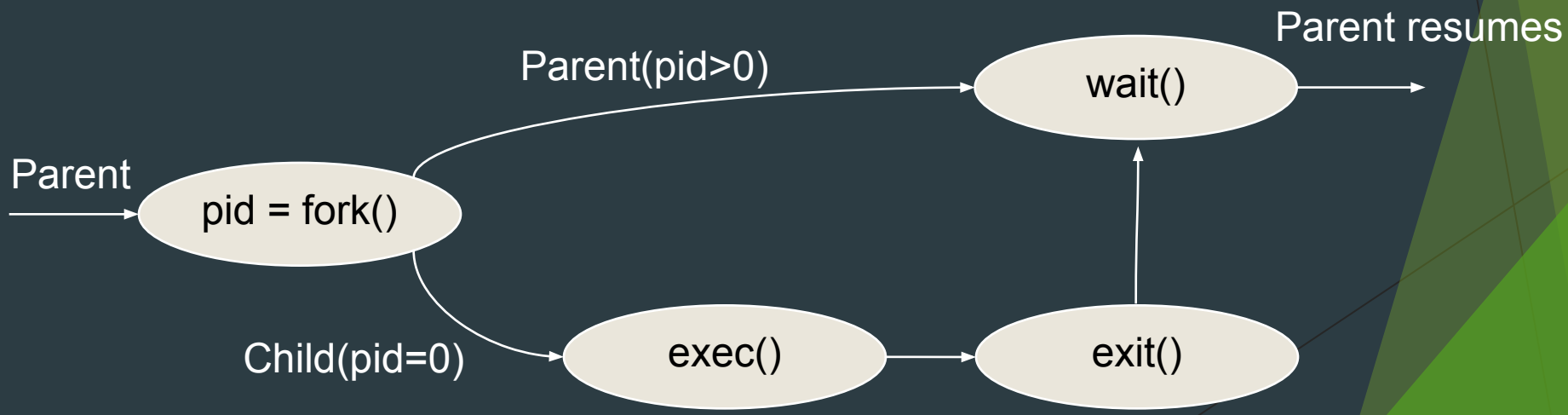
```
#include <sys/wait.h>
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
      pid_t  pid;
      /* fork another process */
      pid = fork();
      if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
      }
      else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
      }
      else { /* parent process */
        wait (NULL);
        printf ("Child Complete");
        exit(0);
      }
}
```

Child process

Parent will wait for the child to complete

行程的結束(Process termination)

- 一個行程在執行完最後一個敘述, 以及使用**系統呼叫 exit()** 要求作業系統將自己刪除時結束。這個行程的所有資源 (包括實體記憶體、虛擬記憶體、開啟檔案, 以及輸入輸出緩衝區) 都由作業系統收回。
- 一個父行程可以基於若干理由將子行程中止：
  - 子行程已經使用超過配置的資源數量。
  - 指派給子行程的工作已經不再需要。
  - 父行程結束, 而作業系統不允許子行程在父行程結束之後繼續執行。



Process Termination

- Wait for termination, returning the pid:

```
pid_t pid;
int status;
pid = wait(&status);
```

- If no parent waiting, then terminated process is a **zombie**
- If parent terminated, processes are **orphans**

僵屍行程 (Zombie Process)

- 在UNIX中使用fork()創建行程時，將複製父行程的地址空間。如果父行程使用wait()，那麼父行程將暫停執行，直到子進程終止。
- 在子行程終止時，會發出一個“SIGCHLD”信號，該信號會由內核傳遞給父行程。父行程在收到“SIGCHLD””後，會從程序表(process table)中刪除子行程資訊。
- 若其父行程不使用wait()等待子行程結束，當子行程結束時，父行程不會知道子行程狀態，因此程序表(process table)裡面仍會保留子行程資訊，這個子行程狀態就是所謂的僵屍行程。

Example

```
File  Edit  View  Search  Terminal  Help
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    // Fork returns process id
    // in parent process
    pid_t child_pid = fork();

    // Parent process
    if (child_pid > 0)
        sleep(50);

    // Child process
    else
        exit(0);

    return 0;
}
```

- The child finishes its execution using exit() system call while the parent sleeps for 50 seconds, hence doesn't call wait() and the child process's entry still exists in the process table.
- Ctrl+z 背景執行(或./zombie &)
- $ps 顯示執行中的行程
- $fg 前景執行

```
brucelin@brucelin-VirtualBox:~/linux_programming/ch03$ ps
  PID TTY          TIME CMD
28662 pts/1    00:00:00 bash
31994 pts/1    00:00:00 zombie
31995 pts/1    00:00:00 zombie <defunct>
31996 pts/1    00:00:00 ps
```

Example of Zombie process

```
brucelin@titan53:~/fork$ ./zombie
pid=726742, ppid=726741
^Z
[1]+  Stopped                 ./zombie
brucelin@titan53:~/fork$ ps
    PID TTY          TIME CMD
 715502 pts/15   00:00:00 bash
 726741 pts/15   00:00:00 zombie
 726742 pts/15   00:00:00 zombie <defunct>
 726775 pts/15   00:00:00 ps
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(){
        int i;
        pid_t pid;

        pid=fork();

        if(pid==0){
                printf("pid=%d, ppid=%d\n", getpid(), getppid());
                exit(0);
        }
        else if(pid>0){
                sleep(50);
                exit(0);
        }
        else{
                exit(1);
        }
        return 0;
}
```

孤兒行程 (Orphan Process)

- 當一個父行程沒有等待子行程結束就自行結束，而其子行程尚未結束，這個子行程就是所謂孤兒行程(orphan process)。

- 但是，一旦其父行程死亡，孤兒行程將很快被init行程(pid = 1)認養。

Example

```
brucelin@br
File  Edit  View  Search  Terminal  Help
#include<stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    // Create a child process
    int pid = fork();

    if (pid > 0)
        printf("in parent process\n");

    // Note that pid is 0 in child process
    // and negative if fork() fails
    else if (pid == 0)
    {
        sleep(30);
        printf("in child process\n");
    }

    return 0;
}
```

- 當一個父行程沒有等待子行程結束就自行結束，而其子行程尚未結束，這個子行程就是所謂孤兒行程(orphan process)。
- 這個例子中，父行程很快就結束，但子行程仍在執行。

```
brucelin@brucelin-VirtualBox:~/linux_programming/ch03$ ./orphan
in parent process
brucelin@brucelin-VirtualBox:~/linux_programming/ch03$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
brucelin 28662 28638  0  三 09 pts/1  00:00:00 bash
brucelin 32156   991  0 17:38 pts/1    00:00:00 ./orphan
brucelin 32157 28662  0 17:38 pts/1    00:00:00 ps -f
brucelin@brucelin-VirtualBox:~/linux_programming/ch03$ in child process
```

Example of orphan process

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(){
        int i;
        pid_t pid;

        pid=fork();

        if(pid==0){
                printf("pid=%d, ppid=%d\n", getpid(), getppid());
                sleep(50);
                printf("pid=%d, ppid=%d\n", getpid(), getppid());
        }
        else{
                exit(0);
        }

}
```

```
brucelin@titan53:~/fork$ ./orphan
pid=727179, ppid=727178
brucelin@titan53:~/fork$ ps
    PID TTY          TIME CMD
 715502 pts/15   00:00:00 bash
 727179 pts/15   00:00:00 orphan
 727187 pts/15   00:00:00 ps
brucelin@titan53:~/fork$ pid=727179, ppid=1
```

init

Quiz 1

- What is the result?

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <wait.h>
int value = 5;
int main(){
        pid_t pid;
        pid = fork();

        if(pid == 0){
                value += 15;
        }
        else{
                wait(NULL);
                printf("Parent: value: %d\n", value);
        }
        return 0;
}
```

Quiz 1

- What is the result?
- Answer is 5 as the child and parent processes each have their own copy of value.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <wait.h>
int value = 5;
int main(){
        pid_t pid;
        pid = fork();

        if(pid == 0){
                value += 15;
        }
        else{
                wait(NULL);
                printf("Parent: value: %d\n", value);
        }
        return 0;
}
```

Quiz 2

- What are the values of lines A, B, C, and D?

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <wait.h>

int main(){
        pid_t pid, pid1;
        pid = fork();
        if(pid<0){
                fprintf(stderr, "Fork failed");
                return 1;
        }
        else if(pid == 0){
                pid1 = getpid();
                printf("child: pid = %d\n", pid);//Line A
                printf("child: pid1 = %d\n", pid1);//Line B
        }
        else{
                pid1 = getpid();
                printf("parent: pid = %d\n", pid);//Line C
                printf("parent: pid1 = %d\n", pid1);//Line D
        wait(NULL);
        }
```

Quiz 2

- What are the values of lines A, B, C, and D?
- ANS:

```
parent: pid = 4148
parent: pid1 = 4147
child: pid = 0
child: pid1 = 4148
```

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <wait.h>

int main(){
        pid_t pid, pid1;
        pid = fork();
        if(pid<0){
                fprintf(stderr, "Fork failed");
                return 1;
        }
        else if(pid == 0){
                pid1 = getpid();
                printf("child: pid = %d\n", pid);//Line A
                printf("child: pid1 = %d\n", pid1);//Line B
        }
        else{
                pid1 = getpid();
                printf("parent: pid = %d\n", pid);//Line C
                printf("parent: pid1 = %d\n", pid1);//Line D
        wait(NULL);
        }
```

Quiz 3

- How many processes are created?

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <wait.h>

int main(){
        int i;
        for(i=0; i<4; i++){
                fork();
        }

        return 0;

}
```

Quiz 3

- How many processes are created?
- ANS: 16

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <wait.h>

int main(){
        int i;
        for(i=0; i<4; i++){
                fork();
        }

        return 0;

}
```

Quiz 4

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#define SIZE 5
int nums[SIZE] = {0,1,2,3,4};
int main()
{
        int i;
        pid_t pid;
        pid = fork();
        if (pid == 0) {
                for (i = 0; i < SIZE; i++) {
                        nums[i] *= -i;
                        printf("CHILD %d\n",nums[i]); /* LINE X */
                }
        }
        else if (pid > 0) {
                wait(NULL);
                for (i = 0; i < SIZE; i++)
                        printf("PARENT: %d\n",nums[i]); /* LINE Y */
        }
        return 0;
}
```

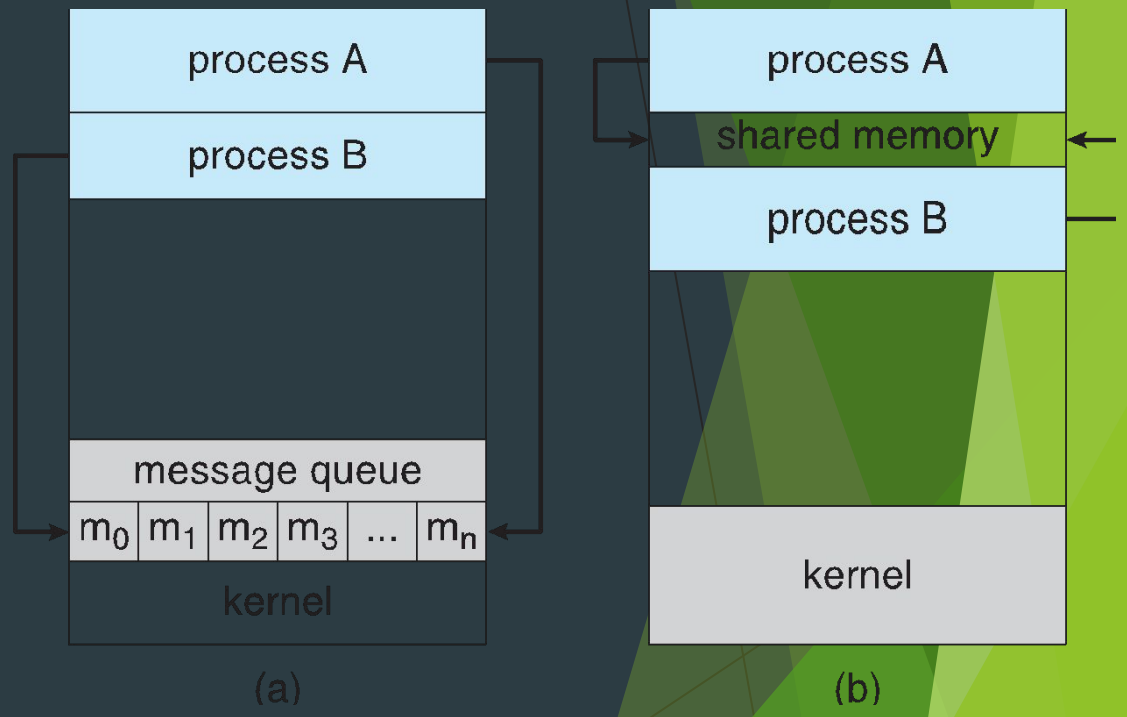
Quiz 4

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#define SIZE 5
int nums[SIZE] = {0,1,2,3,4};
int main()
{
        int i;
        pid_t pid;
        pid = fork();
        if (pid == 0) {
                for (i = 0; i < SIZE; i++) {
                        nums[i] *= -i;
                        printf("CHILD %d\n",nums[i]); /* LINE X */
                }
        }
        else if (pid > 0) {
                wait(NULL);
                for (i = 0; i < SIZE; i++)
                        printf("PARENT: %d\n",nums[i]); /* LINE Y */
        }
        return 0;
}
```

```
CHILD 0
CHILD -1
CHILD -4
CHILD -9
CHILD -16
PARENT: 0
PARENT: 1
PARENT: 2
PARENT: 3
PARENT: 4
```

行程間通訊

- 行程合作(process cooperation)的理由
  - **資訊共享**: 數個使用者可能對相同的一項資訊(例如, 公用檔案)有興 趣, 因此須提供一個環境允許使用者能同時使用這些資源。
  - **加速運算**: 如果希望某一特定工作執行快一點, 則可以分成一些子工作, 每一個子工作都可以和其它子工作平行地執行。
  - **模組化**: 希望以模組的方式來建立系統, 把系統功能分配到數個行程。
  - **方便性**:即使是單一使用者也可能同時執行數項工作。



Producer process using POSIX shared-memory API

- POSIX shared-memory is organized using memory-mapped files(記憶體映射檔案)
- Create shared memory object using shm_open()

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>

int main()
{
        const int SIZE = 4096;
        const char *name = "OS";
        const char *message0= “Hello";
        const char *message1= “World!";
        int shm_fd;
        void *ptr;
```

```
/*建立shared memory object的名稱並設定權限為-rw-rw-rw- ,
        成功的話回傳一個integer file descriptor*/
shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

/* 設定 shared memory 大小為4096 bytes */
ftruncate(shm_fd, SIZE);

/* 建立memory-mapped file包含這個shared memory object */
ptr = mmap(0,SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);

sprintf(ptr,"%s", message0);
ptr += strlen(message0);
sprintf(ptr,"%s", message1);
ptr += strlen(message1);

return 0;
}
```

Consumer process

```
int main()
{
        const char *name = "OS";
        const int SIZE = 4096;

        int shm_fd;
        void *ptr;
        int i;

        /* 開啟shared memory segment，並設定為read-only*/
        shm_fd = shm_open(name, O_RDONLY, 0666);
```

Consumer process

```
        /* 建立memory-mapped file包含這個shared memory object */
    ptr = mmap(0,SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);

    /* now read from the shared memory region */
    printf("%s", (char*)ptr);

    /* remove the shared memory segment */
    shm_unlink(name) ;

    return 0;
}
```

共用記憶體系統Shared-Memory Solution

- 為了闡述合作行程的觀念，讓我們來看 "生產者-消費者"的問題。生產者(producer)行程產生資訊，消費者(consumer)行程消耗掉這些資訊。

- Shared data

```
#define BUFFER_SIZE 10
typedef struct {
    . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- **Solution is correct, but can only use BUFFER_SIZE-1 elements**

```
//producer
item nextProduced;
while (true) {
        /* Produce an item in nextProduced*/
        while (((in + 1) % BUFFER SIZE count)  == out);
                /* do nothing -- no free buffers */
        buffer[in] = nextProduced;
        in = (in + 1) % BUFFER SIZE;
}
```

```
//consumer
item nextConsumed;
while (true) {
        while (in == out) ; // do nothing -- nothing to
    consume

        nextConsumed = buffer[out];
        out = (out + 1) % BUFFER SIZE;
}
```

Circular queue

```
item nextProduced;
while (true) {
        /* Produce an item in nextProduced*/
         while (((in + 1) % BUFFER SIZE)  == out);  //buffer full
         buffer[in] = nextProduced;
         in = (in + 1) % BUFFER SIZE;
}
```

```
  item nextConsumed;
  while (true) {
           while (in == out) ; // buffer is empty
           nextConsumed = buffer[out];
           out = (out + 1) % BUFFER SIZE;
  }
```

Producer using shared memory

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#define N 8

int main(int argc, char* argv[]){

        const int SIZE = N*sizeof(int);
        const char *shm_buffer_name = "shm_buffer";
        const char *shm_in_name = "shm_in";
        const char *shm_out_name = "shm_out";

        if(argc!=2){
                printf("using command: %s [integer]", argv[0]);
                exit(1);
        }

        int shm_fd, shm_in, shm_out;
        static int *buffer, *in, *out;
```

```
        /* create the shared memory segment */
shm_fd = shm_open(shm_buffer_name, O_CREAT | O_RDWR, 0666);
if (shm_fd == -1) {
        printf("shared memory failed\n");
        exit(-1);
}

shm_in = shm_open(shm_in_name, O_CREAT | O_RDWR, 0666);
if (shm_in == -1) {
        printf("shared memory failed\n");
        exit(-1);
}

shm_out = shm_open(shm_out_name, O_CREAT | O_RDWR, 0666);
if (shm_out == -1) {
        printf("shared memory failed\n");
        exit(-1);
}

/* configure the size of the shared memory segment */
ftruncate(shm_fd, SIZE);
ftruncate(shm_in, sizeof(int));
ftruncate(shm_out, sizeof(int));
```

```
        /* now map the shared memory segment in the address space of the process */
    buffer = (int *)mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (buffer == MAP_FAILED) {
        printf("Map buffer failed\n");
        return -1;
    }

    in = (int*)mmap(0, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED, shm_in, 0);
    if (in == MAP_FAILED) {
        printf("Map in failed\n");
        return -1;
    }
    out = (int*)mmap(0, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED, shm_out, 0);
    if (out == MAP_FAILED) {
        printf(" Map out failed\n");
        return -1;
    }

    while((*in+1)%N==*out){
        printf("buffer is full!\n");
        exit(1);
    }

    buffer[*in] = atoi(argv[1]);
    printf("Produce buffer[%d]: %d\n", *in, buffer[*in]);
    *in = (*in + 1)%N;

    printf("Next in:%d, out:%d\n", *in, *out);
    return 0;
}
```

Example

```
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./producer 1
in:1, out:0
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./producer 2
in:2, out:0
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./producer 3
in:3, out:0
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./producer 4
in:4, out:0
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./producer 5
in:5, out:0
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./producer 6
in:6, out:0
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./producer 7
in:7, out:0
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./producer 8
buffer is full!
```

Consumer using shared memory

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#define N 8

int main()
{
        const int SIZE = 8*sizeof(int);
        const char *shm_buffer_name = "shm_buffer";
        const char *shm_in_name = "shm_in";
        const char *shm_out_name = "shm_out";
        int shm_fd, shm_in, shm_out;
        int *buffer, *in, *out;


        int i;
```

```
/* open the shared memory segment */
        shm_fd = shm_open(shm_buffer_name, O_RDONLY, 0666);
        if (shm_fd == -1) {
                printf("shared memory failed\n");
                exit(-1);
        }

        shm_in = shm_open(shm_in_name, O_CREAT | O_RDWR, 0666);
        if (shm_in == -1) {
                printf("shared memory failed\n");
                exit(-1);
        }
        shm_out = shm_open(shm_out_name, O_CREAT | O_RDWR, 0666);
        if (shm_out == -1) {
                printf("shared memory failed\n");
                exit(-1);
        }

        /* configure the size of the shared memory segment */
        ftruncate(shm_fd, SIZE);
        ftruncate(shm_in, sizeof(int));
        ftruncate(shm_out, sizeof(int));
```

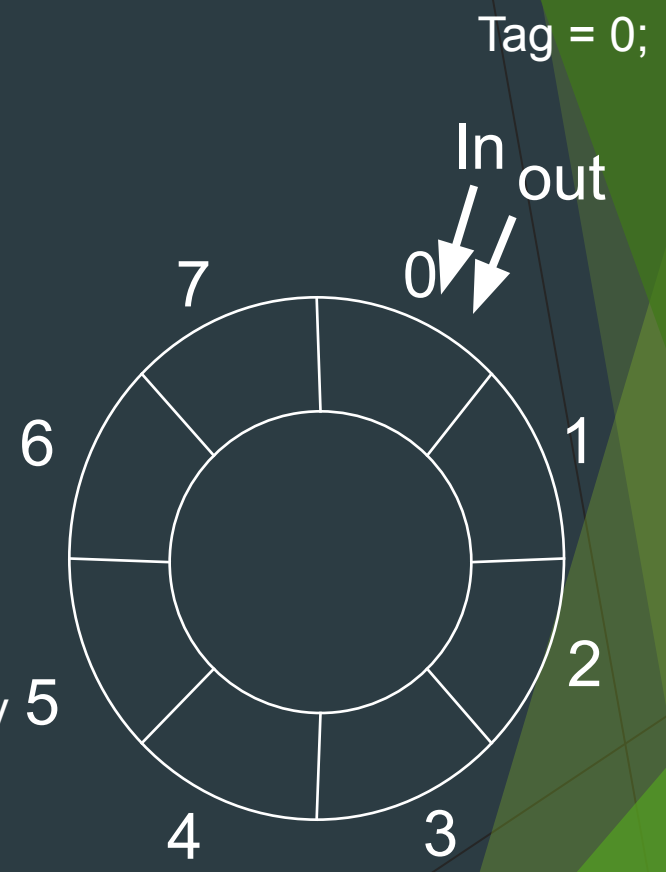
```
        /* now map the shared memory segment in the address space of the process */
    buffer = mmap(0,SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
    if (buffer == MAP_FAILED) {
        printf("Map failed\n");
        exit(-1);
    }
    in = (int*)mmap(0, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED, shm_in, 0);
    if (in == MAP_FAILED) {
        printf("Map in failed\n");
        return -1;
    }
    out = (int*)mmap(0, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED, shm_out, 0);
    if (out == MAP_FAILED) {
        printf(" Map out failed\n");
        return -1;
    }

    while(*in==*out){
        printf("buffer is empty!\n");
        exit(1);
    }
    printf("consumed buffer[%d]: %d\n", *out, buffer[*out]);
    *out = (*out + 1) % N;
     printf("in:%d, next out:%d\n", *in, *out);

    return 0;
}
```

```
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./consumer
consumed: 1
in:7, out:1
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./consumer
consumed: 2
in:7, out:2
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./consumer
consumed: 3
in:7, out:3
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./consumer
consumed: 4
in:7, out:4
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./consumer
consumed: 5
in:7, out:5
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./consumer
consumed: 6
in:7, out:6
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./consumer
consumed: 7
in:7, out:7
brucelin@brucelin-VirtualBox:~/OS/ch03$ ./consumer
buffer is empty!
```

Solution

```
item nextProduced;
tag = 0;
while (true) {
        /* Produce an item in nextProduced*/
         while (in  == out && tag==1);  //buffer full
        buffer[in] = nextProduced;
         in = (in + 1) % BUFFER SIZE;
       if(in==out) tag = 1;
}
```

```
item nextConsumed;
while (true) {
        while (in == out && tag==0) ; // buffer is empty
       nextConsumed = buffer[out];
        out = (out + 1) % BUFFER SIZE;
        if(out==in) tag = 0;
}
```



Homework

- Modify the producer and consumer processes according to the solution on the previous page.

訊息傳遞系統Message Passing

- 訊息傳遞提供了允許行程互相溝通和彼此同步而不需要共享相同的位址空間。
- 訊息傳遞設施提供至少兩種操作:send (訊息) 和receive (訊息)。
- 如果兩個行程 P 與 Q 要互相聯繫, 則它們必須互相傳送與接收訊息。為了使它們可這樣做, 因此在它們間必須存在一個通訊鏈。

命名

- **直接聯繫 (direct communication)**方法中，每一個要傳送或接收訊息的行程必須先確定聯繫接收者或傳送者的名稱。在這個體系之中，send 與 receive的基本運算定義如下:
  - send (P, message)傳送一個訊息(message)至行程P。
  - receive (Q, message)自行程Q接收一個訊息(message)。

- **間接式聯繫 (indirect communication)**之中，需藉著信箱(mailbox，也叫作埠port)來傳送與接收訊息。這種 send 與 receive 的基本運算之定義如下:
  - send (A, message) 將訊息 (message)傳送至信箱A。
  - receive (A, message) 自信箱A接收一個訊息 (message)。

同步化

- 訊息傳遞可以是等待(blocking)或非等待(nonblocking)，也稱為同步(synchronous)和非同步 (asynchronous)。
  - 等待傳送 (blocking send):傳送行程等待著，直到接收行程或信箱接收訊息。
  - 非等待傳送 (nonblocking send):傳送行程送出訊息，即重新操作。
  - 等待接收 (blocking receive):接收者等待，直到有效訊息出現。
  - 非等待接收 (nonblocking receive):接收者收到有效訊息或無效資料。

POSIX

- **POSIX**是IEEE為要在各種UNIX作業系統上執行的軟體，而定義API的一系列互相關聯的標準的總稱，其正式稱呼為IEEE 1003，而國際標準名稱為ISO／IEC 9945。
- 此標準源於一個大約開始於1985年的項目。POSIX這個名稱是由理察·斯托曼應IEEE的要求而提議的一個易於記憶的名稱。它基本上是**Portable Operating System Interface**（可移植作業系統介面）的縮寫，而**X**則表明其對Unix API的傳承。

- Linux基本上逐步實作了POSIX相容，但並沒有參加正式的POSIX認證。

- 微軟的Windows NT至少部分實作了POSIX相容。

Quiz 5 using shared memory

- Parent process shares data with child process using shared memory

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
static int *glob_var;
int main(void)
{
        glob_var = mmap(NULL, sizeof (int), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
        *glob_var = 1;
        if (fork() == 0) {
        *glob_var = 5;
        exit(EXIT_SUCCESS);
          } else {
        wait(NULL);
        printf("%d\n", *glob_var);
        munmap(glob_var, sizeof *glob_var);
          }
          return 0;
}
```

Quiz 6 using shared memory

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/mman.h>
#define SIZE 5

int main()
{
	int i;
	pid_t pid;
	int *nums;
	nums = mmap(NULL, SIZE*sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED  |MAP_ANONYMOUS, -1, 0);
	for(i=0; i<SIZE; i++){
		nums[i] = i;
	}
	pid = fork();
	if (pid == 0) {
		for (i = 0; i < SIZE; i++) {
			nums[i] *= -i;
			printf("CHILD %d\n",nums[i]); /* LINE X */
		}
	}
	else if (pid > 0) {
		wait(NULL);
		for (i = 0; i < SIZE; i++)
			printf("PARENT: %d\n",nums[i]); /* LINE Y */
	}
	return 0;
}
```

Quiz 6 using shared memory

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/mman.h>
#define SIZE 5

int main()
{
        int i;
        pid_t pid;
        int *nums;
        nums = mmap(NULL, SIZE*sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED  |MAP_ANONYMOUS, -1, 0);
        for(i=0; i<SIZE; i++){
                nums[i] = i;
        }
        pid = fork();
        if (pid == 0) {
                for (i = 0; i < SIZE; i++) {
                        nums[i] *= -i;
                        printf("CHILD %d\n",nums[i]); /* LINE X */
                }
        }
        else if (pid > 0) {
                wait(NULL);
                for (i = 0; i < SIZE; i++)
                        printf("PARENT: %d\n",nums[i]); /* LINE Y */
        }
        return 0;
}
```

```
CHILD 0
CHILD -1
CHILD -4
CHILD -9
CHILD -16
PARENT: 0
PARENT: -1
PARENT: -4
PARENT: -9
PARENT: -16
```

Quiz 7: POSIX shared memory

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/mman.h>
#include <fcntl.h>

#define SIZE 5

int main()
{
        int i;
        pid_t pid;
        int shm_fd;//shared memory file descriptor
        int* ptr;//pointer to shared memory object
        const char* name = "sharedMem";//name of shared memory
        const int SHM_SIZE = SIZE * sizeof(int);//size of shared memory
        /*Create the shared memory object*/
        shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

        /*Configure the size of the shared memory*/
        ftruncate(shm_fd, SHM_SIZE);

        /*memory map the shared memory object*/
        ptr = (int*)mmap(0, SHM_SIZE, PROT_WRITE | PROT_READ, MAP_SHARED, shm_fd, 0);

        for(i=0; i < SIZE; i++){
            ptr[i] = i;
        }
```

```
    pid = fork();
    if (pid == 0) {
        for (i = 0; i < SIZE; i++) {
            ptr[i] *= -i;
            printf("CHILD %d\n",ptr[i]); /* LINE X */
        }
    }
    else if (pid > 0) {
        wait(NULL);
        for (i = 0; i < SIZE; i++)
            printf("PARENT: %d\n",ptr[i]); /* LINE Y */
    }

    return 0;
}
```

```
    pid = fork();
    if (pid == 0) {
        for (i = 0; i < SIZE; i++) {
            ptr[i] *= -i;
            printf("CHILD %d\n",ptr[i]); /* LINE X */
        }
    }
    else if (pid > 0) {
        wait(NULL);
        for (i = 0; i < SIZE; i++)
            printf("PARENT: %d\n",ptr[i]); /* LINE Y */
    }

    return 0;
}
```

```
CHILD 0
CHILD -1
CHILD -4
CHILD -9
CHILD -16
PARENT: 0
PARENT: -1
PARENT: -4
PARENT: -9
PARENT: -16
```