Chương 7

Tiến trình (Process)

Phần 1: Quản lý tiến trình

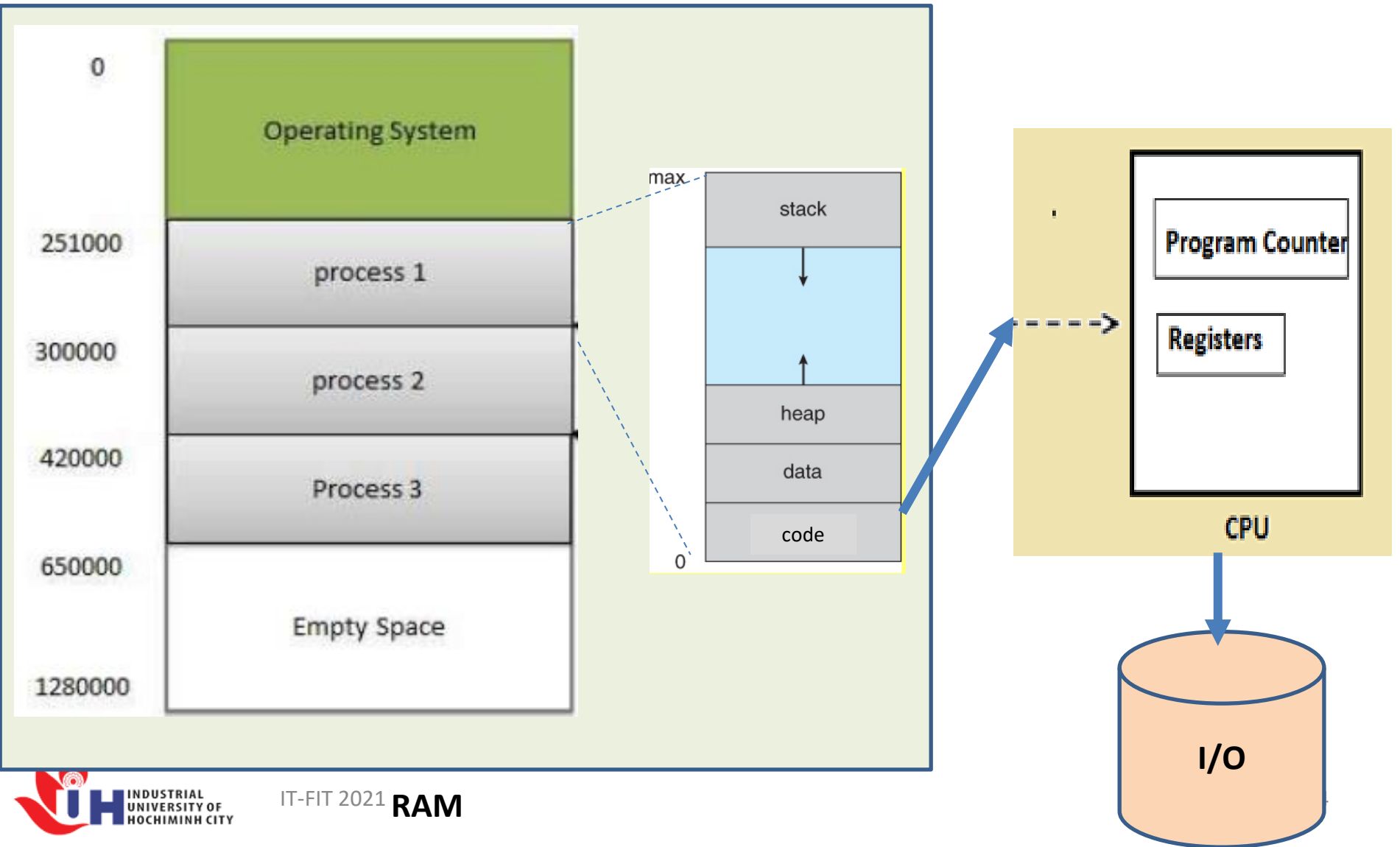
Nội dung

- Khái niệm tiến trình
  - Các trạng thái của tiến trình
  - Khối điều khiển tiến trình PCB
  - Tạo và kết thúc tiến trình
- Điều phối tiến trình
- Liên lạc giữa các tiến trình
- Đồng bộ tiến trình
- Deadlock

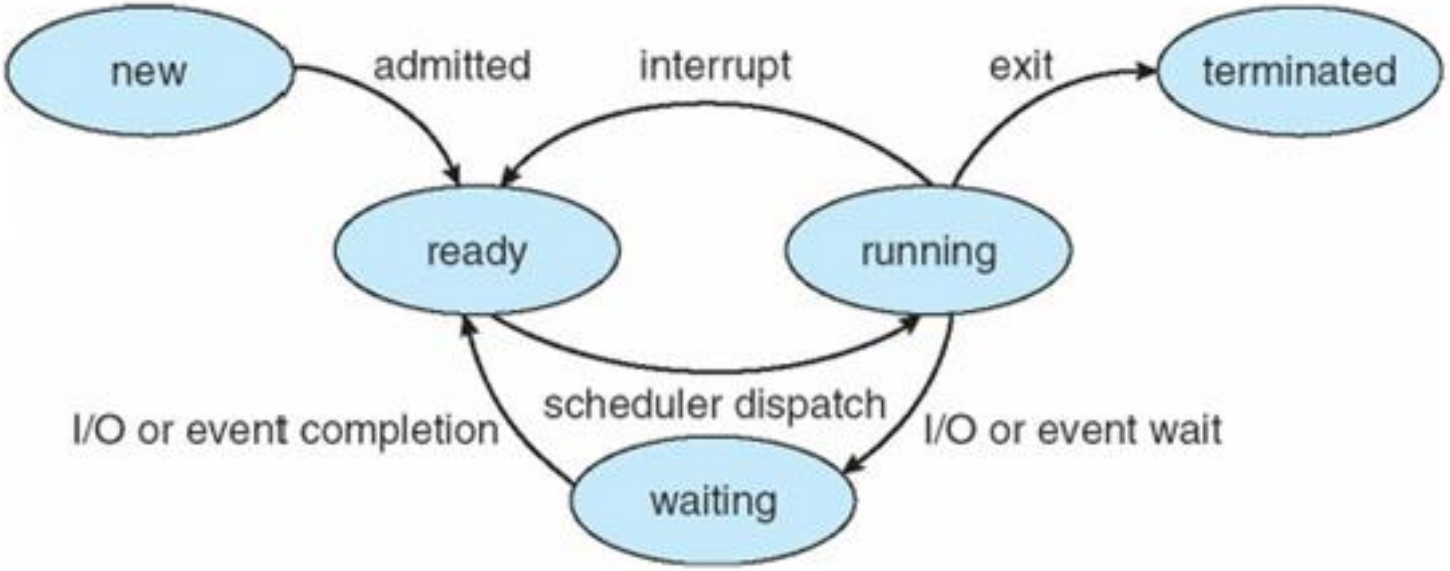
Khái niệm tiến trình

- Process – tiến trình : Là chương trình đang thực thi
  - Chương trình (program) : tĩnh
  - Tiến trình (process) : động
- Một process bao gồm ?

Khái niệm tiến trình

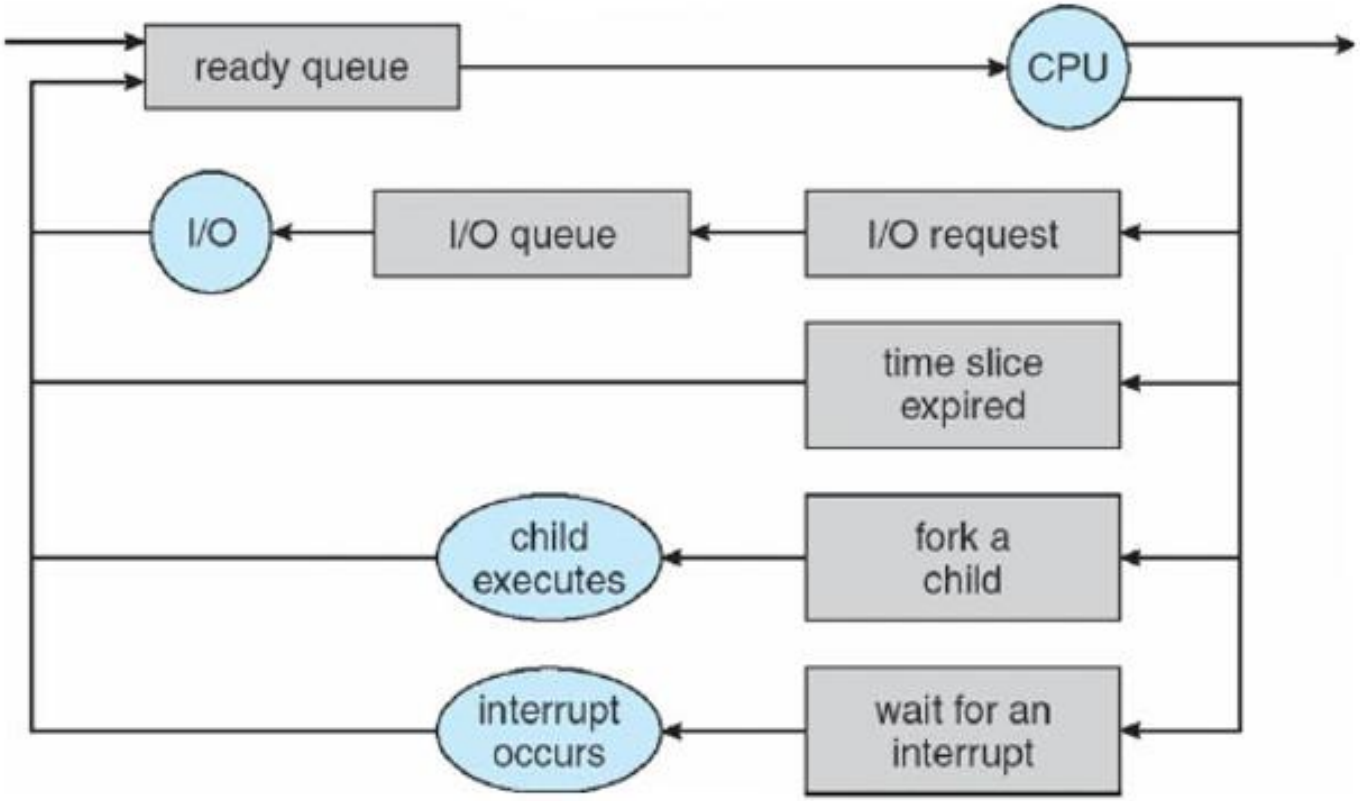


Các trạng thái của Tiến trình



- As a process executes, it changes *state*
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

Các trạng thái của Tiến trình



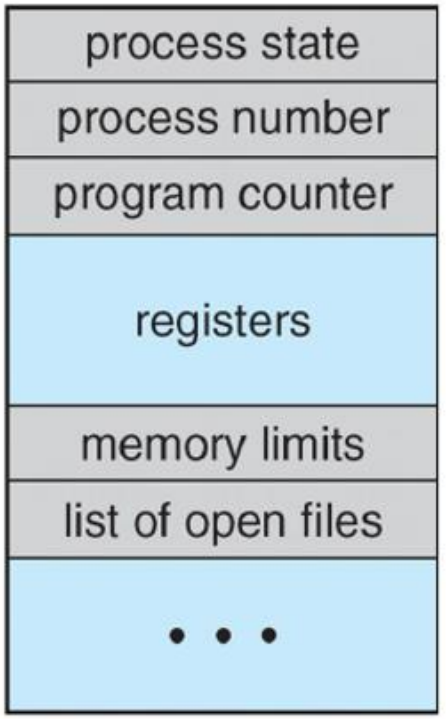
Process Control Block (PCB)

- Được tạo và quản lý bởi OS

Information associated with each process

(also called **task control block**)

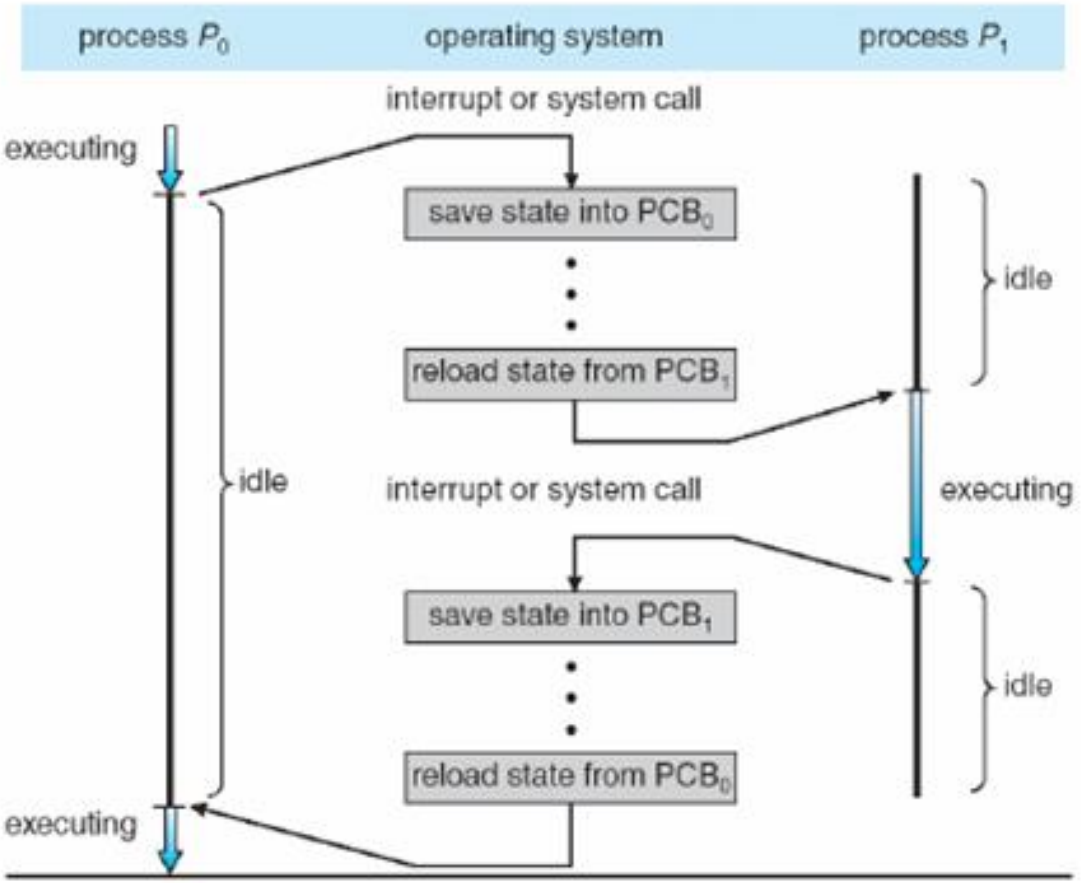
- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



Process Control Block (PCB)

The significant point about the process control block is that it contains sufficient information so that it is possible to interrupt a running process and later resume execution as if the interruption had not occurred. The process control block is the key tool that enables the OS to support multiple processes and to provide for multiprocessing. When a process is interrupted, the current values of the program counter and the processor registers (context data) are saved in the appropriate fields of the corresponding process control block, and the state of the process is changed to some other value, such as *blocked* or *ready* (described subsequently). The OS is now free to put some other process in the running state. The program counter and context data for this process are loaded into the processor registers and this process now begins to execute.

Chuyển ngữ cảnh (Context Switch)



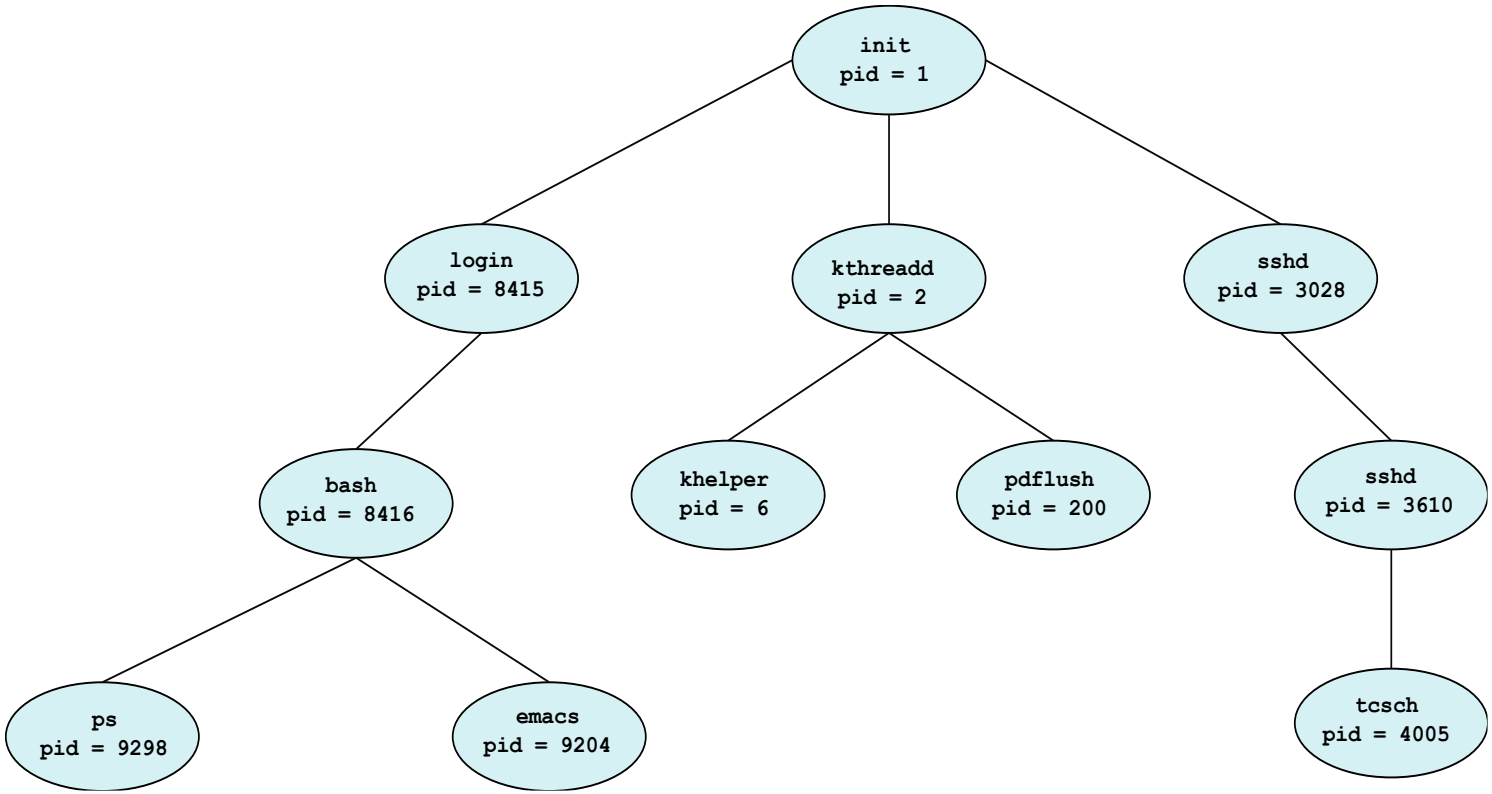
Chuyển ngữ cảnh (Context Switch)

- When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**
- **Context** of a process represented in the PCB
- **Context-switch time** is overhead; the system does no useful work while switching

Các thao tác trên process
Tạo tiến trình

- Một process tạo ra một/nhiều new process
  - Resource sharing options
    - Parent and children share all resources
    - Children share subset of parent' s resources
    - Parent and child share no resources
  - Execution options
    - Parent and children execute concurrently
    - Parent waits until children terminate
  - Address space
    - Child duplicate of parent
    - Child has a program loaded into it
- Tiến trình cha tạo các tiến trình con => tạo các tiến trình cháu => hình thành cây tiến trình (Linux)

Các thao tác trên process
Tạo tiến trình

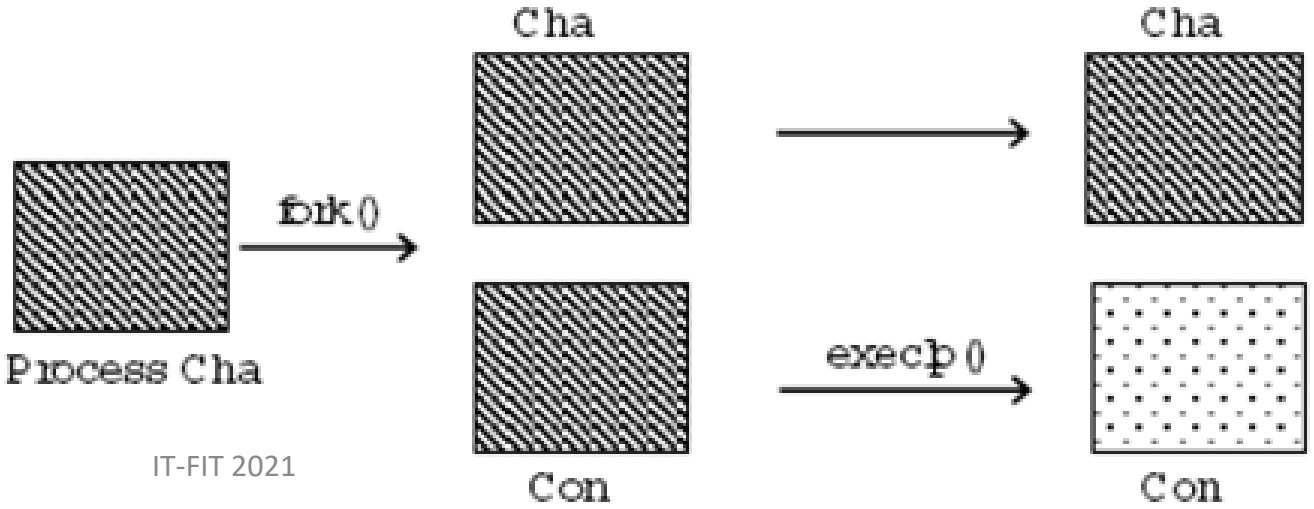


A Tree of Processes in Linux

Các thao tác trên process

Tạo tiến trình

- Tiến trình đang chạy gọi một system call của OS để tạo new process, và chỉ định chương trình chạy trong new process
  - Ví dụ trong UNIX/Linux
    - System call fork() tạo một process mới
    - System call execlp() dùng sau fork() để nạp một chương trình mới vào không gian nhớ của process mới



Chương trình sử dụng fork() để sinh ra tiến trình con

```
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 */
            /* parent will wait for the child to complete */
            wait (NULL);
            printf ("Child Complete");
            exit(0);
    }
}
```

//Wait for termination, returning the pid

```
//pid t pid; int status;
//pid = wait(&status);
```

Các thao tác trên process
Kết thúc tiến trình

- Tiến trình kết thúc khi :
  - Normal exit (tự nguyện)
  - Error exit (tự nguyện)
  - Fatal error exit (ép buộc)
  - Được kết thúc bởi một tiến trình khác (ép buộc)

- Gọi một system call để thực hiện kết thúc một tiến trình
  - Trong Unix : exit() => kết thúc tự nguyện
    kill() => kết thúc ép buộc

Các thao tác trên process
Kết thúc tiến trình

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - **cascading termination**

Các thao tác trên process
Kết thúc tiến trình

- Một số tình huống :

(1) Cha kết thúc -> tất cả các con kết thúc

(2) Cha kết thúc , nhưng con chưa kết thúc -> **orphan**

(3) Con kết thúc, nhưng cha chưa kết thúc hoặc chưa gọi hàm wait() -> **zombie**

(4) Con bị treo , cha chờ con kết thúc -> cả hai còn trong hệ thống -> cần kết thúc ép buộc

Multiprocess Architecture

Chrome Browser

- Many web browsers ran as single process (some still do)
  - If one web site causes trouble, entire browser can hang or crash
- Google Chrome Browser is multiprocess with 3 categories
  - **Browser** process manages user interface, disk and network I/O
  - **Renderer** process renders web pages, deals with HTML, Javascript, new one for each website opened
    - Runs in **sandbox** restricting disk and network I/O, minimizing effect of security exploits
  - **Plug-in** process for each type of plug-in

https://blog.chromium.org/2008/09/multi-process-architecture.html

Multiprocess Architecture
Chrome Browser

Image Name	PID	Command Line
Avira.Systray.exe	4372	"C:\Program Files\Avira\Launcher\Avira.Systray.exe" /connectToHost
avshadow.exe	920	"C:\Program Files\Avira\AntiVir Desktop\avshadow.exe" avshadowcontrol0_00000308
chrome.exe	1548	"C:\Program Files\Google\Chrome\Application\chrome.exe" --type=crashpad-handler "--user-data-dir=
chrome.exe	3080	"C:\Program Files\Google\Chrome\Application\chrome.exe" --type=renderer --field-trial-handle=1232,4
chrome.exe	3184	"C:\Program Files\Google\Chrome\Application\chrome.exe" --type=renderer --field-trial-handle=1232,4
chrome.exe	3408	"C:\Program Files\Google\Chrome\Application\chrome.exe" --type=watcher --main-thread-id=4712 --o
chrome.exe	4108	"C:\Program Files\Google\Chrome\Application\chrome.exe" --type=gpu-process --field-trial-handle=123
chrome.exe	4620	"C:\Program Files\Google\Chrome\Application\chrome.exe" --type=renderer --field-trial-handle=1232,4
chrome.exe	4716	"C:\Program Files\Google\Chrome\Application\chrome.exe"
chrome.exe	4908	"C:\Program Files\Google\Chrome\Application\chrome.exe" --type=renderer --field-trial-handle=1232,4
chrome.exe	5048	"C:\Program Files\Google\Chrome\Application\chrome.exe" --type=renderer --field-trial-handle=1232,4
chrome.exe	5696	"C:\Program Files\Google\Chrome\Application\chrome.exe" --type=renderer --field-trial-handle=1232,4
chrome.exe	7208	"C:\Program Files\Google\Chrome\Application\chrome.exe" --type=renderer --field-trial-handle=1232,4

Tham khảo

- Some of the Win32 calls for managing processes, threads, and fibers. (tham khảo Tanenbaum , p.920)

Win32 API Function	Description
CreateProcess	Create a new process
CreateThread	Create a new thread in an existing process
CreateFiber	Create a new fiber
ExitProcess	Terminate current process and all its threads
ExitThread	Terminate this thread
ExitFiber	Terminate this fiber
SwitchToFiber	Run a different fiber on the current thread
SetPriorityClass	Set the priority class for a process
SetThreadPriority	Set the priority for one thread
CreateSemaphore	Create a new semaphore
CreateMutex	Create a new mutex
OpenSemaphore	Open an existing semaphore
OpenMutex	Open an existing mutex

Tham khảo

- Some system calls relating to processes (tham khảo Tanenbaum , p.737)

System call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, opts)	Wait for a child to terminate
s = execve(name, argv, envp)	Replace a process' core image
exit(status)	Terminate process execution and return status
s = sigaction(sig, &act, &oldact)	Define action to take on signals
s = sigreturn(&context)	Return from a signal
s = sigprocmask(how, &set, &old)	Examine or change the signal mask
s = sigpending(set)	Get the set of blocked signals
s = sigsuspend(sigmask)	Replace the signal mask and suspend the process
s = kill(pid, sig)	Send a signal to a process
residual = alarm(seconds)	Set the alarm clock
s = pause( )	Suspend the caller until the next signal

Bài tập

- Chrome Browser có thiết kế multiprocess vì lý do gì ?

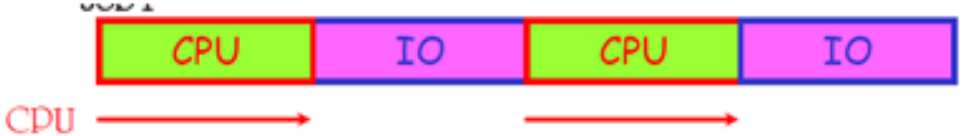
Chương 7

Tiến trình (Process)

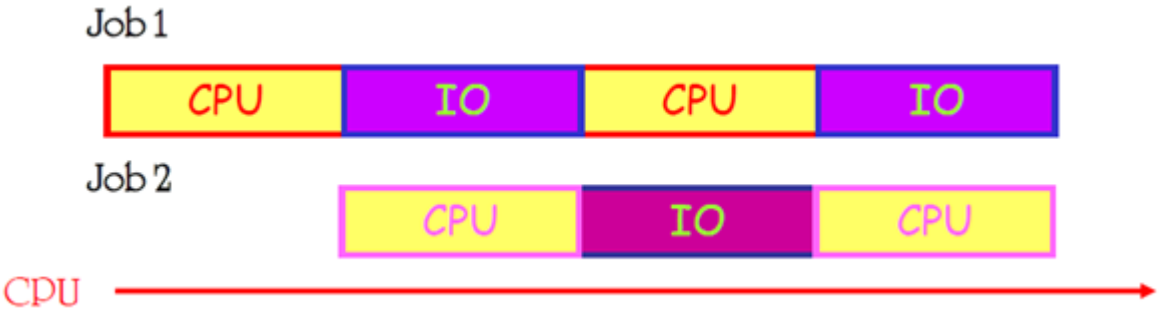
Phần 2: Các giải thuật về tiến trình

Multiprogramming và multitasking

- Batch systems

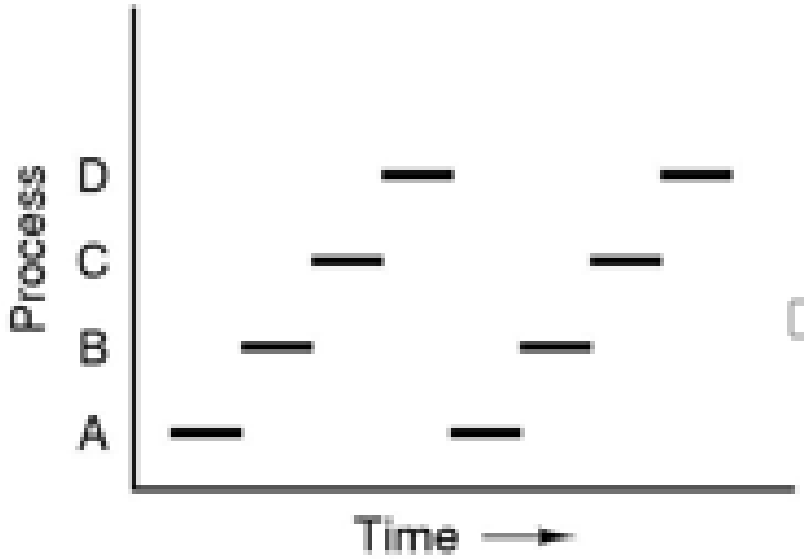


- Đa chương (multiprogramming)



Multiprogramming và multitasking

- Đa nhiệm (multitasking) hay time-sharing



Nhiều process "cùng được" xử lý thông qua cơ chế chuyển đổi CPU
Thời gian mỗi lần chuyển đổi diễn ra rất nhanh, đủ để thấy nhiều process đang chạy đồng thời

Process Scheduler

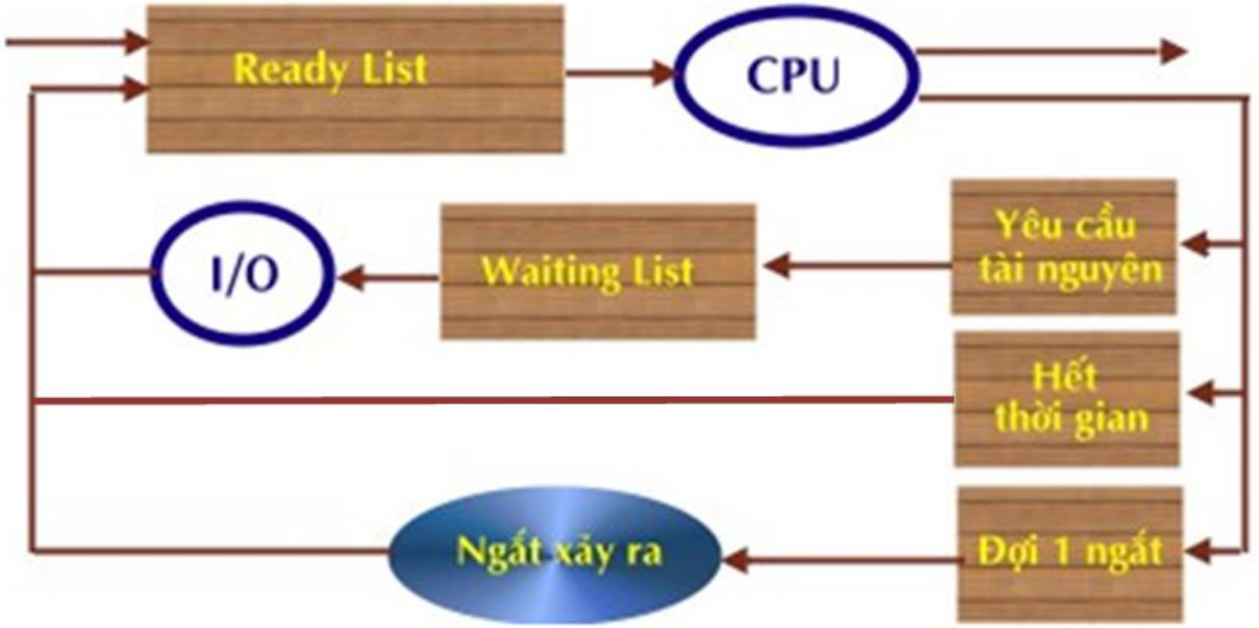
- Trong hệ multiprogramming và multitasking, **bộ điều phối (scheduler)** thực hiện chức năng **điều phối** các tiến trình
  - Chọn tiến trình được xử lý bởi CPU trong số các tiến trình đang chờ được xử lý.
  - Quản lý các hàng đợi read queue, I/O device queues
  - Chuyển ngữ cảnh

Điều phối được thực hiện khi nào ?

- Bộ điều phối cần ra quyết định vào thời điểm :
  - Khi một tiến trình kết thúc hay chuyển sang trạng thái blocked (chờ I/O , …) => chọn tiến trình nào trong hàng đợi ready ?
  - Một ngắt I/O xuất hiện từ một thiết bị I/O báo đã hoàn tất
  - Ngắt định kỳ xuất hiện từ bộ đếm thời gian

Điều phối được thực hiện khi nào ?

Hàng đợi Ready và I/O waiting



- Hàng đợi Ready list : bao gồm các proccess ở trạng thái sẵn sàng tiếp nhận CPU
- Hàng đợi Waiting list : bao gồm nhiều hàng đợi - mỗi tài nguyên có một hàng đợi riêng

Mục tiêu điều phối

- Mục tiêu điều phối:
  - Công bằng
    - Không có tiến trình nào phải chờ vô hạn
  - Tối đa thời gian sử dụng CPU (efficiency)
  - Thông lượng tối đa (throughput)
    - tối đa số công việc được xử lý trong 1 đơn vị thời gian
  - Cực tiểu thời gian hoàn thành (turnaround time)
    - Là tổng thời gian chờ trong Ready List + thời gian thực thi CPU + thời gian thực hiện I/O
  - Cực tiểu thời gian chờ (waiting time) trong Ready List
  - Cực tiểu thời gian đáp ứng (response time)
  - Đảm bảo tính ưu tiên
  - …

Nguyên tắc điều phối

- Nguyên tắc điều phối độc quyền và không độc quyền
  - Điều phối độc quyền (non-preemptive)
    - Điều phối không can thiệp vào thời gian dùng CPU của process (*)
    - thích hợp với hệ xử lý theo lô
  - Điều phối không độc quyền (preemptive)
    - Điều phối có can thiệp vào thời gian dùng CPU của process (*)
    - thích hợp với hệ thống tương tác và real-time

Tổ chức điều phối

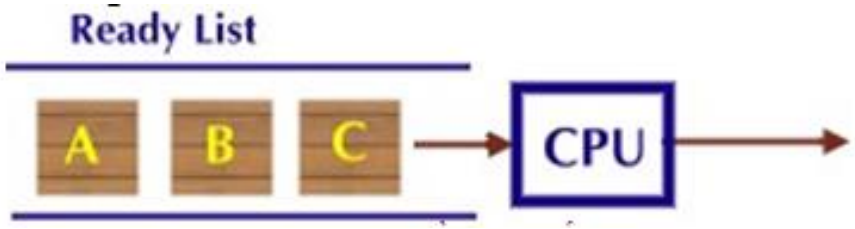
- Chuyển ngữ cảnh
  - Khi CPU chuyển sang thực thi một proccess khác, hệ thống phải lưu trạng thái của proccess hiện tại và nạp trạng thái của proccess mới sẽ thực thi
  - Ngữ cảnh của một proccess được biểu diễn trong khối PCB của proccess đó
  - Thời gian chuyển ngữ cảnh là 1 phí tổn

Khảo sát
các chiến lược điều phối

- **FIFO hay FCFS (first come first server)**
- **Round Robin (phân phối xoay vòng)**
- **Điều phối với độ ưu tiên**
- **SJF (Shortest-job-first)**
- Chiến lược điều phối với nhiều mức độ ưu tiên

Chiến lược FIFO

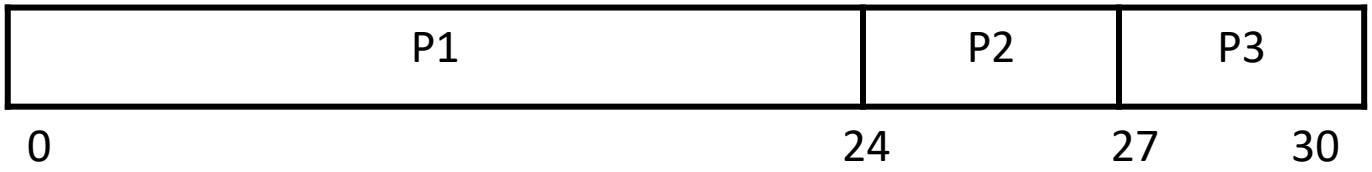
- Tiến trình vào Ready list trước => được cấp CPU trước
- CPU được giải phóng chỉ khi
  - tiến trình kết thúc xử lý
  - khi có một yêu cầu nhập/xuất



Chiến lược FIFO

Tiến trình	Thời điểm vào RL	Thời gian xử lý
P1	0	24
P2	1	3
P3	2	3

- Thứ tự cấp phát CPU



- Thời gian chờ P1 =0 , P2 = 24-1 , P3 = 27 - 2
- Thời gian chờ trung bình : (0+23+25)/3 = 16 miliseconds

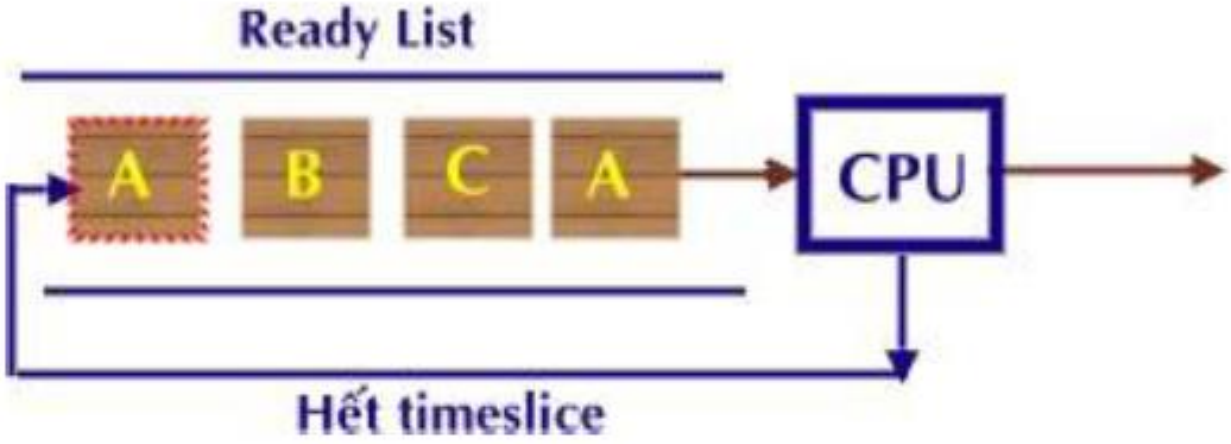
Chiến lược FIFO

- Nhận xét :
  - Thời gian chờ trung bình :
    - Phụ thuộc vào thời gian xử lý của các proccess
    - phụ thuộc vào thứ tự của các process trong RL (*)
  - Là giải thuật điều phối theo nguyên tắc độc quyền
    => Giải thuật không phù hợp với hệ time-sharing
    => Phù hợp với batch systems

Round Robin

(phân phối xoay vòng)

- Các process được xử lý xoay vòng
  - một khoảng thời gian sử dụng CPU như nhau gọi là time quantum (hay time slice)
  - Hết thời gian quantum dành cho process, HDH thu hồi CPU và cấp cho process kế tiếp trong RL. Process được đưa trở lại vào RL chờ đến lượt .



Round Robin

(phân phối xoay vòng)

Tiến trình	Thời điểm vào RL	Thời gian xử lý
P1	0	24
P2	1	3
P3	2	3

- Nếu **quantum = 4 miliseconds**
- Thứ tự cấp phát CPU

P1	P2	P3	P1	P1	P1	P1	P1
0	‘4	7	10	14	18	22	26 30

- Thời gian chờ P1 =0 , P2 = 3 , P3 = 5
- Thời gian chờ trung bình : (0+6+3+5)/3 = 4.66 miliseconds

Round Robin
(phân phối xoay vòng)

- CPU được giải phóng khi
  - Hết thời gian quantum
  - Tiến trình kết thúc
  - Tiến trình bị khóa

- Nhận xét :
  - Là giải thuật điều phối không độc quyền
  - Thiết kế phù hợp với time-sharing systems và Interactive systems
  - Độ dài của quantum ntn cho hợp lý ?
    - Turnaround time also depends on the size of the time quantum

Round Robin

Độ dài của quantum ntn cho hợp lý ?

q large ⇒ FIFO

q small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high

if the time quantum is extremely large, the RR policy is the same as the FCFS policy. In contrast, if the time quantum is extremely small (say, 1 millisecond), the RR approach can result in a large number of context switches.

Thus, we want the time quantum to be large with respect to the context-switch time. If the context-switch time is approximately 10 percent of the time quantum, then about 10 percent of the CPU time will be spent in context switching. In practice, most modern systems have time quanta ranging from 10 to 100 milliseconds. The time required for a context switch is typically less than 10 microseconds; thus, the context-switch time is a small fraction of the time quantum.

Điều phối với độ ưu tiên

- Mỗi tiến trình được gán một độ ưu tiên
  - Độ ưu tiên do hệ điều hành qui định (loại chương trình, người sở hữu ,…)
- Tiến trình có độ ưu tiên cao hơn sẽ được dùng CPU trước; Các tiến trình có độ ưu tiên bằng nhau được sắp lịch theo FCFS
- Có thể thực hiện bằng Cơ chế độc quyền hay không độc quyền

Điều phối với độ ưu tiên

Ví dụ : (độ ưu tiên 1 > độ ưu tiên 2> độ ưu tiên 3)

Tiến trình	Thời điểm vào RL	Độ ưu tiên	Thời gian xử lý
P1	0	3	24
P2	1	1	3
P3	2	2	3

Sử dụng thuật giải độc quyền, thứ tự cấp phát CPU như sau :

P1	P2	P3
0	‘24	27 30

Sử dụng thuật giải không độc quyền, thứ tự cấp phát CPU như sau :

P1	P2	P3	P1
0	‘1	4	7 30

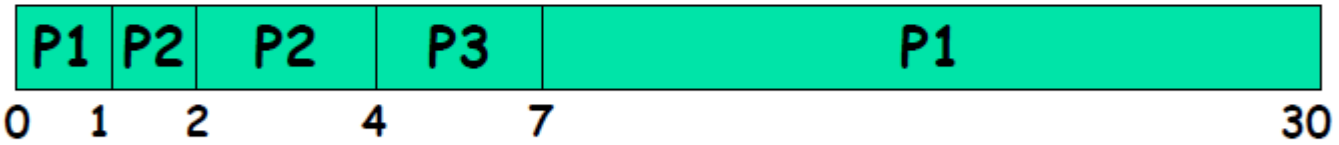
Điều phối với độ ưu tiên

Minh họa độ ưu tiên (khôngđộc quyền)

P	T_{RL}	Priority	CPU burst
P1	0	2	24
P2	1	0	3
P3	2	1	3

P	TT	WT
P1	30	0+(7-1)
P2	4-1	0
P3	7-2	4-2

$Avg_{WT} = (6+0+2)/3 = 2.66$



0:00 P1 vào, P1 dùng CPU

0:01 P2 vào (độ ưu tiên cao hơn P1)

P2 dành quyền dùng CPU

0:02 P3 vào (độ ưu tiên thấp hơn P2)

P3 không dành được quyền dùng CPU

0:4 P2 kết thúc, P3 dùng CPU

0:7 P3 dừng, P1 dùng CPU

0:30 P1 dừng

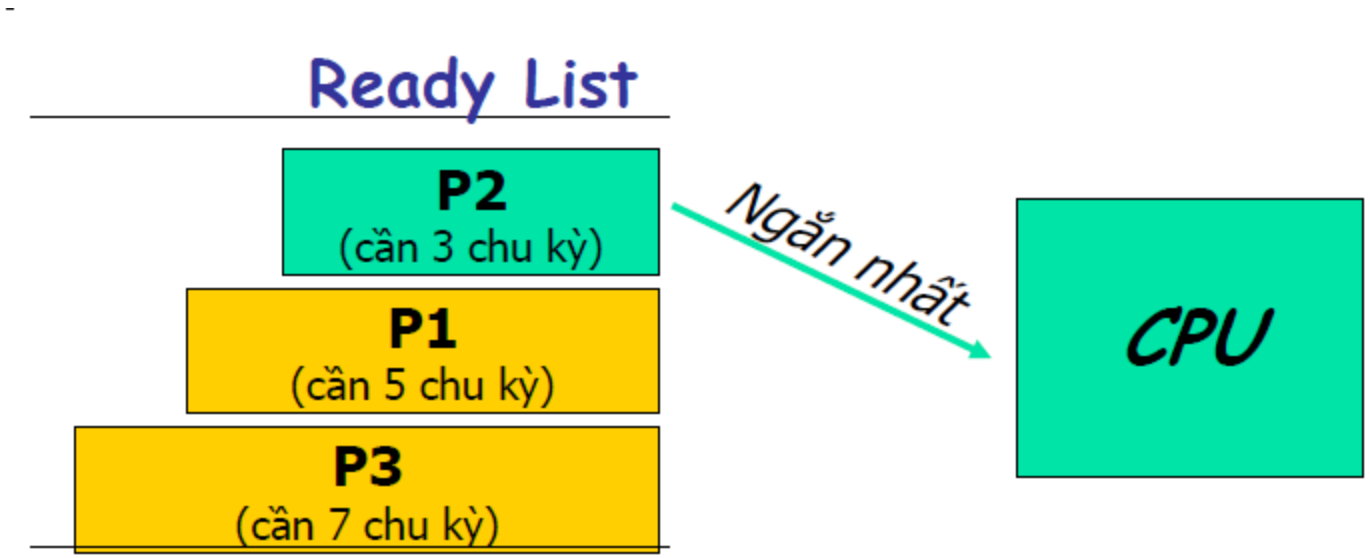
Điều phối với độ ưu tiên

- Nhận xét :
  - Các tiến trình có độ ưu tiên thấp có thể phải chờ CPU vô hạn !

    => Giải pháp ? Aging

Công việc ngắn nhất - SJF

(Shortest-job-first)



Là một dạng độ ưu tiên đặc biệt với độ ưu tiên

p_i = thời_gian_còn_lại($Process_i$)

➔ Có thể cài đặt độc quyền hoặc không độc quyền

Công việc ngắn nhất - SJF

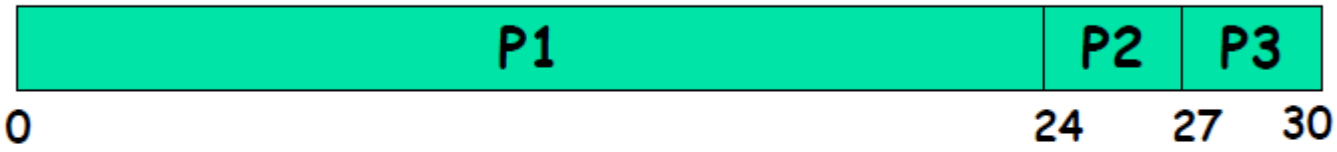
(Shortest-job-first)

Minh họa SJF (độc quyền)(1)

P	$T_{arriveRL}$	CPU burst
P1	0	24
P2	1	3
P3	2	3

P	TT	WT
P1	24	0
P2	27	24-1
P3	30	27-2

$Avg_{WT} = (23+25)/3 = 16$



0:00 P1 vào, P1 dùng CPU

0:01 P2 vào RL

0:02 P3 vào RL

0:24 P1 kết thúc, P2 dùng CPU

0:27 P2 dừng, P3 dùng CPU

0:30 P3 dừng

Công việc ngắn nhất - SJF

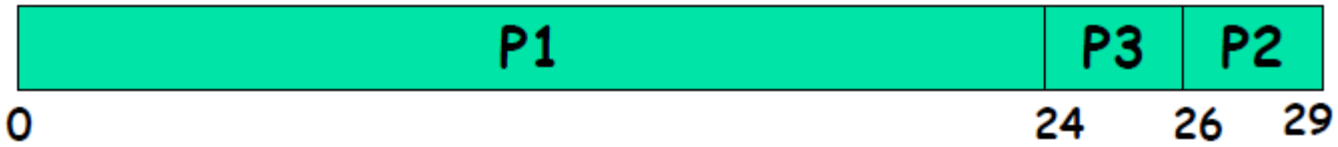
(Shortest-job-first)

Minh họa SJF (độc quyền)(2)

P	$T_{arriveRL}$	CPU burst
P1	0	24
P2	1	3
P3	1	2

P	TT	WT
P1	24	0
P2	29	26-1
P3	26	24-2

$Avg_{WT} = (24+22)/3 = 15.33$



0:00 P1 vào, P1 dùng CPU

0:01 P2 vào

0:01 P3 vào

0:24 P1 kết thúc, P3 dùng CPU

0:26 P3 dừng, P2 dùng CPU

0:29 P2 dừng

Công việc ngắn nhất - SJF

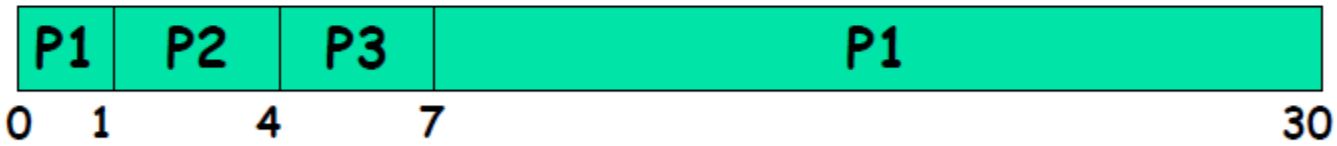
(Shortest-job-first)

Minh họa SJF (khôngđộc quyền) (1)

P	$T_{arriveRL}$	CPU burst
P1	0	24
P2	1	3
P3	2	3

P	TT	WT
P1	30	0+(7-1)
P2	4-1	0
P3	7-2	4-2

Avg_{WT} = (6+0+2)/3 = 2.66



0:00 P1 vào, P1 dùng CPU

0:01 P2 vào (độ ưu tiên cao hơn P1)

P2 dành quyền dùng CPU

0:4 P2 kết thúc, P3 dùng CPU

0:7 P3 dừng, P1 dùng CPU

0:30 P1 dừng

Công việc ngắn nhất - SJF

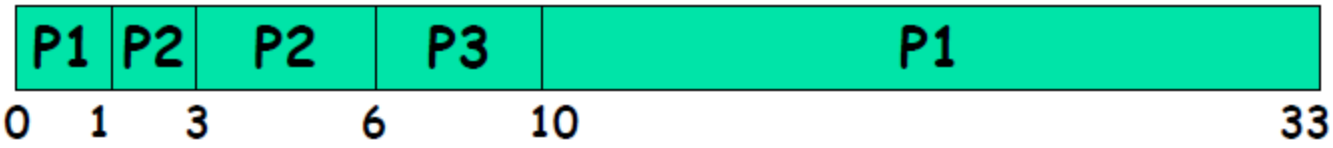
(Shortest-job-first)

Minh họa SJF (khôngđộc quyền) (2)

P	$T_{arriveRL}$	CPU burst
P1	0	24
P2	1	5
P3	3	4

P	TT	WT
P1	33	0+(10-1)
P2	6	0
P3	10	6-3

Avg_{WT} = (9+0+3)/3 = 4



0:00 P1 vào, P1 dùng CPU

0:01 P2 vào (độ ưu tiên cao hơn P1)

P2 dành quyền dùng CPU

0:03 P3 vào (độ ưu tiên < P2)

P2 dành quyền dùng CPU

0:6 P2 kết thúc, P3 dùng CPU

0:9 P3 dừng, P1 dùng CPU

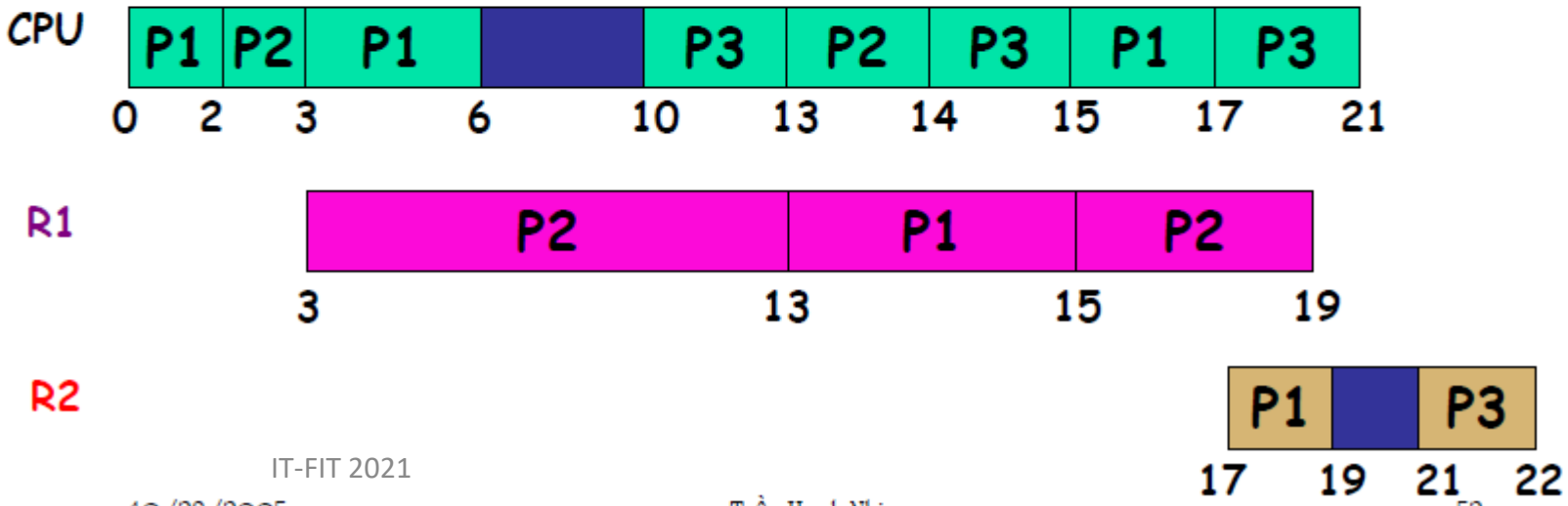
0:33 P1 dừng

Công việc ngắn nhất - SJF

(Shortest-job-first)

Minh họa SJF (nhiều chu kỳ CPU)

P	$T_{arriveRL}$	CPU1 burst	IO1 R	IO1 T	CPU2 burst	IO2 R	IO2 T
P1	0	5	R1	2	2	R2	2
P2	2	1	R1	10	1	R1	4
P3	10	8	R2	1	0	Null	0



Bài tập

Bài tập: Hãy điều phối

CPU: SJF không độc quyền. R1,R2: FIFO

Tiến trình	Thời điểm vào Ready list	CPU1	IO lần 1		CPU2	IO lần 2	
			Thời gian	Thiết bị		Thời gian	Thiết bị
P1	0	8	5	R1	1	0	Null
P2	2	1	8	R2	2	5	R1
P3	10	6	5	R1	2	3	R2
P4	11	3	20	R2	0	0	Null

Các giải thuật khác

- **Multilevel Queue Scheduling**
  - Các process được phân loại và có queue riêng cho mỗi nhóm và có Thuật toán lập lich cho mỗi queue
    - **foreground** (interactive) : dùng RR
    - **background** (batch) : dùng FCFS
  - Bộ lập lịch thực hiện phân lịch theo
    - Dựa trên độ ưu tiên của các process trong các queue : các process thuộc nhóm foreground có độ ưu tiên cao hơn background
    - Dùng time slice – mỗi queue nhận cố định một lượng thời gian dùng CPU . VD, 80% thời gian CPU dành cho các process thuộc queue foreground .

Các bộ điều phối khác

- **Multiple-Processor Scheduling**
- **Real-Time CPU Scheduling**

Tham khảo : Silberschatz, p. 278

Operating-System Examples

Silberschatz, p.290

- Linux
- Windows

Bài tập 1

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

- Tất cả đều đến ở thời điểm 0
- Xét các giải thuật FCFS, SFJ, và RR với quantum time = 10
- Giải thuật nào cho
  - thời gian đợi trung bình nhỏ nhất?
  - thông năng cao nhất?
  - thời gian quay vòng trung bình của process nhỏ nhất?

Tóm tắt

- Hệ multiprogramming và multitasking
- Bộ điều phối (scheduler)
  - Chức năng
  - Tổ chức điều phối
  - Mục tiêu đặt ra cho chiến lược điều phối
  - Nguyên tắc điều phối độc quyền và không độc quyền
- Các chiến lược điều phối : FCFS, RoundRobin, độ ưu tiên, SJF

Liên lạc giữa các tiến trình

Interprocess Communication (IPC)

- Các tiến trình trong hệ thống có thể độc lập hay có hợp tác với nhau
- Các tiến trình hợp tác với nhau xuất phát từ nhu cầu :
  - Chia sẻ thông tin
  - Tăng tốc độ tính toán
  - Cấu trúc module của chương trình
- Khi hợp tác , các tiến trình cần giao tiếp với nhau

Liên lạc giữa các tiến trình

- Do mỗi tiến trình sở hữu một không gian địa chỉ riêng => HĐH phải cung cấp cơ chế liên lạc giữa các tiến trình
- Các vấn đề nảy sinh trong liên lạc giữa các tiến trình :
  - Liên kết tường minh hay tiềm ẩn
  - Liên lạc theo chế độ đồng bộ hay bất đồng bộ
  - Liên lạc giữa các tiến trình trong 1 máy tính khác biệt với liên lạc giữa các tiến trình giữa các máy tính khác nhau

Các cơ chế liên lạc

- Signal
- Pipe
- Shared memory
- Message passing
- Socket

Interprocess Communication

- *Tham khảo : Silberschatz, p 818*
- **Signals**
  - Signals can be sent from any process to any other process, with **restrictions** on signals sent to processes owned by another user
  - informing a process that an **event** has occurred
  - In Linux, signals have always been the main mechanism for communicating asynchronous events among processes

Interprocess Communication

- **Signals**
  - Tín hiệu được sử dụng để thông báo cho 1/nhiều tiến trình về một sự kiệnnào đó xảy ra. Nhận xét :
    - Mang tính chất ko đồng bộ (1 tiến trình không biết trước thời điểm nhận tín hiệu)
    - Thông báo một biến cố - không trao đổi data
  - Tín hiệu có thể phát ra bởi :
    - Phần cứng (vd lỗi do các phép tính số học)
    - Kernel gửi đến tt (thông báo có một tbị I/O tự do)
    - Một tt gửi đến một tt khác (tt cha yêu cầu con kết thúc)
    - Người dùng nhấn phím (ctrl-C để break tt)
  - Xem danh sách các signal dùng kill –l
  - Mỗi signal tương ứng với 1 sự kiện

Interprocess Communication

- **Passing of Data among Processes**
  - **Pipe**
    - unidirectional byte streams which connect the **standard output** from one process into the **standard input** of another process
    - Ex. the shell which sets up these **temporary pipes** between the processes.

```
ls | pr  -l  20  -h   test        //linux
dir  |  sort  /R                  //windows
```

Interprocess Communication

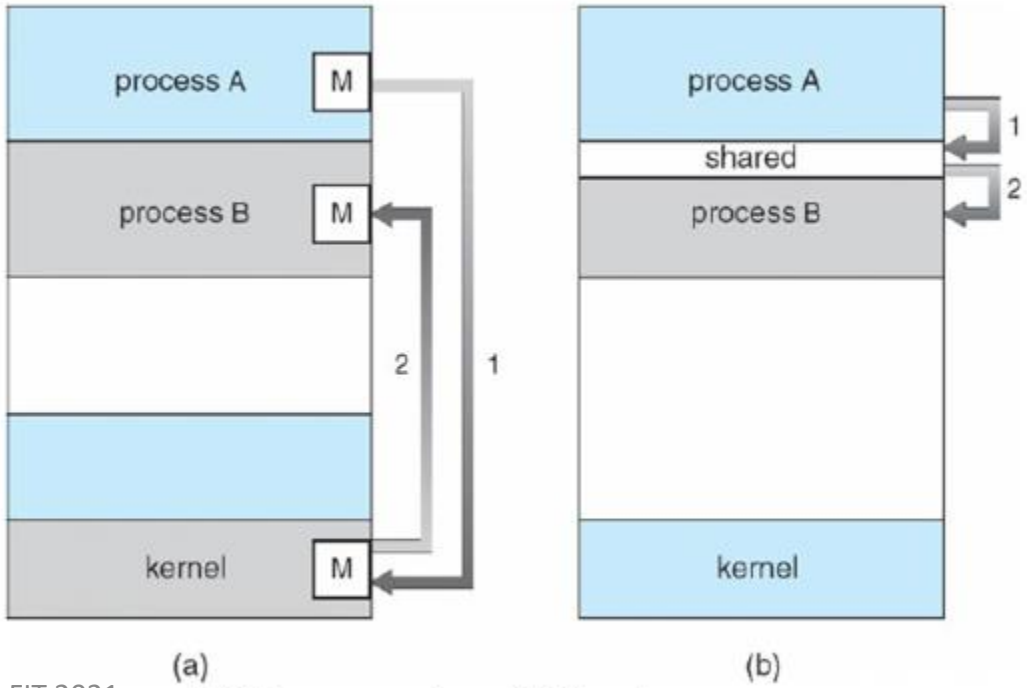
- **Passing of Data among Processes**
  - **shared memory**
    - fast way to communicate large or small amounts of data

Shared memory

Typically, a shared-memory region resides in the address space of the process creating the sharedmemory segment. Other processes that wish to communicate using this sharedmemory segment must attach it to their address space. Recall that, normally, the operating system tries to prevent one process from accessing another process's memory. Shared memory requires that two or more processes agree to remove this restriction. They can then excbange information by reading and writing data in the shared areas. The **form** of the data and **the location** are determined by these processes and are not under the operating system's control. The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

Interprocess Communication

- **Passing of Data among Processes**
  - **Message passing**
    - Có thể dùng trong môi trường phân tán



(a) Message passing (b) Shared memory

Interprocess Communication

- Trong shared-memory = chia sẽ biến chung (thường được tạo trong 1 process)
  - người lập trình ứng dụng chịu trách nhiệm tạo, kiểm soát … ,
  - nhiệmvụ của OS chỉ là tạo vùng nhớ chung.

- Trong message passing : trao đổi messages ; OS chịu trách nhiệm chính

- => OS có thể hỗ trợ cả 2 pp.

Tham khao : IPC share memory (http://www.cs.cf.ac.uk/Dave/C/node27.html)