

Chương 1

1.1 Định nghĩa hệ điều hành

- Máy tính số là máy nhiều cấp, trong đó Scđp chính yếu là:

- + Vật lý (phần cứng - hardware)

- + Chương trình hệ thống (system program)

- + Chương trình ứng dụng (application program)

Hệ điều hành là gì?

Có 2 định nghĩa được dùng y nhiều nhất

1) Là 1 máy tính luân lý mở rộng (extended machine)
đây là gói nhùn từ ngoài vào.

- Giải các chi tiết béo, phần phức cài thử hiện.

- Cung cấp cho user dùng 1 máy luân lý
để dùng hơn và đặc lập với phần cứng.
(thông qua các lệnh system call)

2) Là 1 hệ quản lý cái tài nguyên của máy.
đây là gói nhùn bên trong.

- Phân chia việc dùng tài nguyên theo thời gian, mỗi chương trình dùng tài nguyên trong 1 khoảng thời gian rất nhỏ cho người khác dùng.

- Phân chia tài nguyên theo không gian: mỗi

Chương trình dùng 1 vùng nhớ để nguyên

1.2 Lịch sử HDH

1. First generation (1945-1955)

- vacuum tube, plug board
- Inventor: Aiken (USA), Zuse (Germany)
- Chưa có HDH.

2. Second generation 1955-1965

- Transistor.
- Batch system.

3. third generation 1965-1980

- ICs (. Integrated Circuits)
- multi programming, spooling, time-sharing

4. Fourth generation. 1980 - present

- LSI (Large Scale Integration)
- HDH cho FC.

Nhắc lại phần cứng máy tính

processors

- Special register
 - Program counter
 - Stack pointer
 - Program Status Word (PSW)
 - Kernel Mode
 - User Mode

- TRAP instruction
- System call

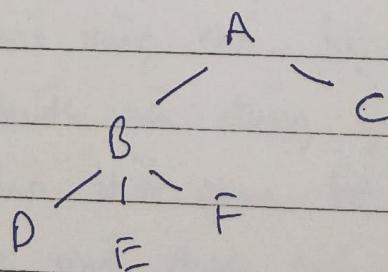
Memory

Typical access time	Typical capacity
1 n sec - Register	<1 kB
2 n sec - Cache	1 MB
10 n sec - Main memory	64-512 MB
10 m sec - Magnetic disk	5-50 GB
100 sec - Magnetic tape	20-100 GB

Các ý niệm chủ đạo

HĐH cần quản lý các tài nguyên sao cho việc sử dụng chúng bởi các chương trình được tin cậy, an toàn, hiệu quả và đột lặp với tính chất vật lí của chúng.

Process: Trong lúc hoạt động, process có thể tạo ra nhiều process khác (process can) và có thể tiếp tục.



Module của HDM quản lý việc phân chia thời gian cho các chương trình chạy được gọi là Scheduler.

Vấn đề tuy xuất hiện tại nguyên chung chung

Race là hiện tượng lỗi bất định có thể xảy ra khi 2 hay nhiều process thay xuất 1 tài nguyên chung đồng thời.

Để tránh race, chúng ta sử dụng mutual exclusion để không cho các vùng critical session này chạy đồng thời mà phải tuân tự hoàn việc chạy của chúng.

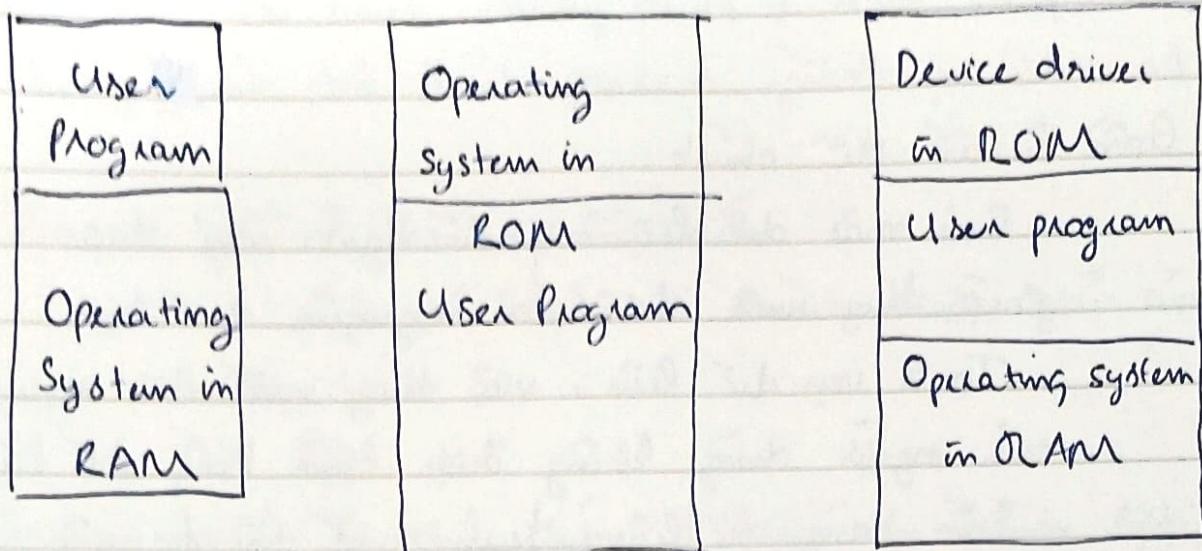
Deadlock và giải quyết.

Deadlock là tình trạng của hệ thống mà ở đó có ít nhất 2 process đang dừng chờ lân nhau và bị kẹt mãi mãi ở trạng thái này.

Trường hợp xấu nhất là mọi process đều bị dừng và chờ lân nhau, hệ thống sẽ bị té liệt mãi mãi

Quản lý bộ nhớ trong hệ điều hành

Có 3 cách quản lý bộ nhớ RAM và vùng nhớ của 1 process đang chạy



Quản lý bộ nhớ - Phân vùng tĩnh

Chia bộ nhớ ra nhiều partition với độ lớn khác nhau để chạy nhiều process đồng thời không chung khung.

a) Mỗi partition có riêng chờ các process đồng thời chung dung lượng bộ nhớ.

b) Dùng 1 hàng chờ cho mọi process.

Quản lý bộ nhớ - Phân vùng động

Vùng nhớ lục địa nguyên. Mỗi khi có process xin cấp phát vùng nhớ, hệ thống sẽ tạo 1 partition có kích thước vừa đúng yêu cầu, phân còn lại để trống. Theo thời gian, bộ nhớ có thể bị bẩn mài bén nhiều vùng nhớ được trả lại bởi các process.

Ta có thể khắc phục vấn đề này bằng cách
sắp xếp lại các vùng nhớ sao cho vùng nhớ
trống là duy nhất & liên tục. (compactage)

Ám mìn hệ thống

Gồm 3 vấn đề chính:

- Bảo mật dữ liệu: mỗi người chỉ được phép truy xuất 1 ss' tài nguyên qui định.
- Toàn vẹn dữ liệu: việc truy xuất tài nguyên của người dùng không được làm hỏng dữ liệu
- Sắp xếp dữ liệu: truy xuất tài nguyên phải luôn được thực hiện trong thời gian ngắn

Các lời gọi dịch vụ HJM "System call"

- System call gần giống với gọi hàm bình thường
nhưng khác biệt lớn nhất là sự thay đổi quyền
truy xuất tài nguyên.
- Trước system call: Low priority
- Khi system call: High
- Sau: trả priority như ban đầu (low)

Chương 2:

2.1 Giới thiệu Process

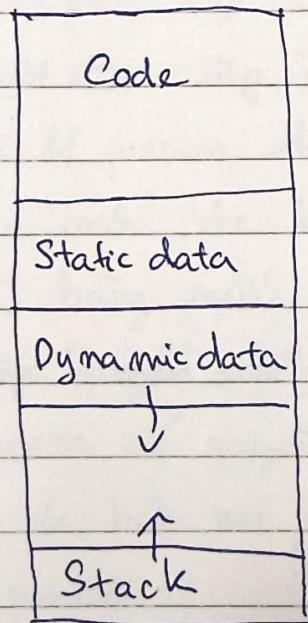
Khi chương trình được nạp vào RAM và CPU

bắt đầu thi hành chương trình ở điểm nhập thi
chương trình trở thành process.

Process gồm 2 thành phần chính

- Danh sách các lệnh cấu thành thuật giải của chương trình
- Dữ liệu

Process tuần tự chia 1 luồng thi hành lệnh
cho 1 chương trình từ điểm nhập đến điểm
kết thúc.



Danh sách mã lệnh

explicit variable

vùng dữ liệu cấp phát động

kích thước biến động theo time

Tạo process

Các trường hợp hệ thống tự tạo:

- 1) Khi khởi động HĐH, các process hệ thống được tạo để quản lý hệ thống.
- 2) Khi người dùng mở phần mềm
- 3) Khi phần mềm gọi Create Process để tạo process mới theo yêu cầu.

Xoá process

1) Nội tại:

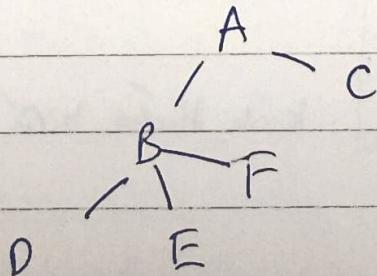
- Khi chương trình kết thúc bình thường
- Gặp lỗi lặp trình viên xử lý trước
- Gặp lỗi ko thuộc xử lý, hệ thống tự xoá.

2) Bên ngoài:

- Một process khác yêu cầu hệ thống kết thúc process.

Cây phân cấp Process

Mô tả mối quan hệ "tạo mới" giữa các process:



Linux đã dùng khái niệm "process group" để quản lý các process

Windows ko quản lý, mọi process đều ngang nhau

Trạng thái process

Các trạng thái的基本:

- Running: đang chiếm CPU
- Ready: chờ được CPU xử lý
- Blocked: chờ hoàn tất I/O

Quản lý trạng thái:

- HPM dùng bảng quản lý process (linked list) để thêm/xóa hiện quyết.

Các phương pháp lập lịch

1) Round-robin:

- Xử lý process theo thứ tự hàng chờ
- Sau mỗi блок thời gian, process quay lại cuối hàng nếu như chưa hoàn tất.

2) Dựa vào quyền ưu tiên:

- Process có quyền ưu tiên cao được chạy trước
- Độ ưu tiên xác định dựa trên user hoặc đặc tính process.

3) Multi-level Queue.

- Process được phân chia vào các hàng chờ theo độ ưu tiên.

- Xử lý theo thứ tự từ hàng hàng từ cao đến low

4) Shortest Job Next:

- Process ngắn nhất chạy trước.

Chương 3 : Tương tranh giữa các process

1. Giới thiệu về tương tranh

Hệ đa chương (multi-tasking) : Các process chạy song song với không gian làm việc độc lập, giúp bảo vệ trinh can thiệp lẫn nhau.

Giao tiếp giữa các process:

- Truy xuất bể nhớ dùng chung (shared memory) : Các process chia sẻ tài nguyên dùng chung cần có chế kiểm soát lỗi race condition
- Gửi/nhận thông tin (message passing)

Critical Session (CS) : là đoạn code dùng truy xuất tài nguyên dùng chung, dễ gây lỗi nếu không kiểm soát.

2. Luật tìm tương hỗ (mutual exclusion)

Nguyên tắc : tại một thời điểm, chỉ một process duy trì cập vùng CS.

Phương pháp :

- Dừng chờ đợi: (busy waiting) : Process

lệnh dừng kiểm tra điều kiện biến tuy cập CS nhưng vẫn chiếm CPU

- Dừng chờ thuỷ động (sleep / wake up) : Process "ngủ" khi phải chờ, giải phóng CPU và được đánh thức khi có thể tiếp tục.

3. Phương pháp dừng chờ chủ động

1) Dựa trên interrupt: CPU sẽ có chân IRQ

(Interrupt Request) nhận tín hiệu ngắt từ thiết bị bên ngoài.

Khi có tín hiệu, CPU:

1) Tạm dừng chương trình hiện tại

2) Xử lý yêu cầu từ thiết bị (tính p.vụ ngoại)

3) Quay lại chương trình đang chạy.

Quản lý CS trong thực tế

In_Control() và Out_Control():

- Khi muốn vào CS, gọi In_Control() để kiểm soát việc truy cập

- Khi hoàn thành, gọi Out_Control() để giải phóng CS cho các process khác.

Đặc điểm:

- Ngắt CPU để bảo vệ vùng gang (CS)

- Hiệu quả nhưng mang rủi ro lỗi tràn sai

2) Phương pháp dùng biến khóa

Ý tưởng:

- Sử dụng biến khóa (process_in_CS) để kiểm tra quyền truy cập CS.
- Nếu biến = 0, process vào CS và set biến = 1
- Khi thoát CS, set lại biến = 0

Vấn đề:

- Có thể thất bại khi 2 process kiểm tra và set đồng thời, dẫn đến cả 2 cùng vào CS. Đây là lỗi do không đảm bảo tính nguyên tử (atomicity) khi thực hiện.

3) Phương pháp dùng lệnh TSL (Test and Set Lock)

Giải pháp:

- TSL đảm bảo tính nguyên tử: kiểm tra và set biến trong một lệnh.

VD:

TSL al; process_in_CS;

if (al=0) break;

- đảm bảo chỉ 1 process truy cập CS tại 1 thời điểm

4) Phương pháp luân phiên

Ý tưởng:

- Sử dụng biến turn để kiểm soát thứ tự process vào CS
 - Mỗi process phải chờ đến lượt (turn) của mình mới được vào CS
- Ưu điểm: đảm bảo công bằng giữa các process
- Nhược điểm: Không phù hợp khi các process có nhu cầu truy cập CS khác nhau.

5) Phương pháp Peterson

- Cài đặt:

- Sử dụng mảng interested [] để ghi nhận ý định vào CS của các process

- Mỗi process:

1. Ichai bao gio muon vao CS

2. Kiem tra ve cho nhieu process khac cung mun

Ưu điểm:

Công bằng hơn, process nào vào trước sẽ được xử lý trước

Hạn chế:

Chỉ áp dụng cho 2 process, cần thuật toán phức tạp hơn nếu có nhiều process.

4. Bài toán sản xuất - tiêu dùng

Mô tả:

- Sản xuất: Tạo sản phẩm và đưa vào kho

- Tiêu dùng: Lấy sản phẩm từ kho sử dụng

Yêu cầu:

1. Tránh tranh chấp khi truy cập kho chứa
2. Đồng bộ hóa độ sản xuất và tiêu dùng để tránh tình trạng:

- Kho đầy: Sản xuất bị chặn
- Kho rỗng: Tiêu dùng bị chặn

Ý tưởng giải quyết:

- Kiểm tra điều kiện (kho đầy / rỗng) trước khi thực hiện.
- Sử dụng hàm sleep() để dừng khi không thể tiếp tục và & wakeup() để đánh thức process.

5. Các phương pháp sleep / wakeup.

1) Phương pháp dùng Semaphore

Semaphore là đối tượng hệ thống, gồm:

- Biến semaphore s luôn giá trị nguyên dương
- Hàm down(s):

Giai giảm giá trị s. Nếu không giảm dc, process phải chờ đến khi giảm được. Hàm này

có tính nguyên tử (không chia cắt)

- Hàm up(s):

Tăng giá trị s. Nếu s=1, đánh thức các process đang chờ. Hàm thực hiện rất nhanh và cũng có tính nguyên tử.



Cách sử dụng semaphore trong bài toán Rain water

- Tính dừng:

- Dùng 1 Semaphore như phần đc bảo vệ vùng CS
- Producer và Consumer gọi down() và up() để kiểm soát vào / ra vùng CS.
- Tại mỗi thời điểm chỉ có 1 process vào CS, các process khác phải chờ.

Hạn chế:

Có nguy cơ deadlock nếu không cẩn thận sắp xếp thứ tự lệnh down() và up().

2) Phương pháp sử dụng Monitor

Monitor là đối tượng hệ thống và:

- Các biến điều khiển chỉ có thể sử dụng qua:
 - wait(cond): Pass phái chờ trên điều kiện cond
 - signal(cond): Pass - đánh thức P chờ trên cond
- Chỉ 1 process được phép thực thi Monitor tại 1 thời điểm (tự động loại trừ tương hỗ)

Cách giải quyết bài toán SX - TD:

- Monitor quản lý kho chứa và các hàm:
 - insert(item): Thêm sản phẩm. Kho đầy, gọi wait()
 - remove(): Lấy sản phẩm. Kho rỗng, gọi wait()
 - Sử dụng notify() đánh thức p nếu cần.

Vấn đề:

- Dễ sử dụng hơn Semaphore
- Tự động đảm bảo tính loại trừ tương hỗ.

Code Java minh họa

ko có

CHƯƠNG 4:

DEADLOCK & XỬ LÝ

1. Định nghĩa deadlock

Là trạng thái của hệ thống mà ở đó có ít nhất 2 process đang dùng chờ lẫn nhau, và như thế chúng không thể chạy tiếp được.

2. Bốn điều kiện cần và đủ để gây ra deadlock

1. Loại trừ tương hỗ: đoạn code CS tuy xuất hiện nguyên dùng chung của các process chạy đồng thời.

2. Process giữ tài nguyên cũ đang chiếm dụng trong khi có gắng xin thêm tài nguyên mới.

3. Hệ thống có dùng tài nguyên "non-preemptive": là loại tài nguyên mà sau khi đã giao cho 1 process nào đó tuy xuất, hệ thống không được quyền lấy lại tài nguyên để cho process khác tuy xuất.

4. Đầu xuất hiện vòng bế tắc giữa các process chờ nhau.

3. Bốn chiến lược giải quyết deadlock.

1. Phối hợp: Không làm gì, hệ thống hy vọng không xảy ra deadlock. Nếu có, thì chấp nhận.

2. Phát hiện và chữa trị (Detection & Recovery):

Hệ thống kiểm tra định kỳ hoặc khi rảnh để phát hiện deadlock. Nếu phát hiện, sẽ thực hiện các biện pháp khắc phục để hệ thống hoạt động bình thường trở lại.

3. Né tránh (Avoidance): Trước khi cấp tài nguyên, kiểm tra xem có nguy cơ gây deadlock không. Nếu phát hiện có, trì hoãn để tránh deadlock.

4. Phòng ngừa (Prevention): Áp dụng quy tắc nghiêm ngặt trong việc cấp phát tài nguyên, đảm bảo deadlock không thể xảy ra.

4. Chiến lược phát hiện & chữa trị deadlock

1) Giải thuật phát hiện deadlock đơn giản

(áp dụng khi mỗi lần tài nguyên chỉ có tối đa một tài nguyên)

- Hệ thống xây dựng đồ thị phụ thuộc để quản lý quan hệ giữa các process và tài nguyên.

- Trong đó thi:

- Nút: đại diện cho process hoặc tài nguyên

- Cung có hướng: Biểu diễn sự phụ thuộc

- Dead lock xảy ra khi có chu trình trong đó事儿, nghĩa là các process chờ lẫn nhau, không có tiến triển.

* Ví dụ minh họa:

- T1: P_1 xin truy xuất $R_1 \rightarrow$ thành công \rightarrow thêm cung từ $R_1 \rightarrow P_1$.

- T2: P_2 xin truy xuất $R_2 \rightarrow$ thành công \rightarrow thêm cung từ $R_2 \rightarrow P_2$

- T3: P_1 xin truy xuất R_2 (P_2 gửi) \rightarrow thêm cung từ $P_1 \rightarrow R_2$, bị dừng chờ.

- T4: P_2 xin truy xuất R_1 (P_1 gửi) \rightarrow thêm cung từ $P_2 \rightarrow R_1$, bị dừng chờ.

\Rightarrow Đến đến dead lock.

2) Giải thuật tổng quát

(Áp dụng khi mỗi loại tài nguyên có nhiều bản sao)

- Hệ thống duy trì cấp phát và quản lý như cũ để theo dõi trạng thái tài nguyên.

- Kiểm tra định kỳ xem có trạng thái ko an toàn (tức là nếu cấp phát có thể gây dead lock)

- Nếu phát hiện dead lock, hệ thống chọn một process để hủy hoặc thu hồi.

5. Giải thuật phát hiện dead lock tổng quát.

1) Các thông số chính trong giải thuật phát hiện dead lock.

- Vector tổng tài nguyên (E):

Mô tả số lượng tài nguyên tổng thể của hệ thống (E_1, E_2, \dots, E_m). Đây là giá trị cố định, không thay đổi trong suốt quá trình vận hành.

- Vector tài nguyên rảnh (A):

Số lượng tài nguyên chưa được sử dụng (A_1, A_2, \dots, A_n)

Luôn tuân thủ: $A_j \leq E_i$, nghĩa là tài nguyên rảnh không thể vượt quá tổng tài nguyên.

- Ma trận phân bố (C):

Biểu diễn số lượng tài nguyên đã được cấp phát cho các tiến trình:

$C_{ij} =$ số lượng tài nguyên loại j mà tiến trình i đang chiếm giữ.

- Ma trận yêu cầu (R):

Biểu diễn tài nguyên mà các tiến trình cần thêm nhưng chưa được cấp phát (vì chưa có sẵn)

$R_{ij} =$ số lượng tài nguyên loại j mà tiến trình i đang yêu cầu.

2) Điều kiện tài nguyên

- Tên nguyên tổng thể E_j được tính bằng:

$$E_j = A_j + \sum_{i=1}^n C_{ij}$$

E_j : Số lượng tài nguyên loại j .

A_j : Số tài nguyên loại j còn rảnh.

$\sum C_{ij}$: Tổng số tài nguyên loại j đang được sử dụng bởi các tiến trình.

3) Ý tưởng phát hiện dead lock:

- Dựa trên cái ma trận và vector, hệ thống kiểm tra vòng phu thuộc tài nguyên giữa các tiến trình.

Quy trình kiểm tra:

1. Tìm kiếm tiến trình i mà tất cả yêu cầu trong hàng R_i đều nhỏ hơn hoặc bằng tài nguyên rảnh A :

$$H_j, R_{ij} \leq A_j$$

2. Nếu tìm thấy, coi như tiến trình i hoàn tất giao phong tài nguyên nó đang giữ và cập nhật:

$$A_j = A_j + C_{ij} \text{ (trả lại tài nguyên)}$$

3. Lặp lại đến khi

- Toàn bộ tiến trình đều được hoàn tất
(hệ thống không bị dead lock)

- Không tiến trình nào thỏa điều kiện
(dead lock tồn tại)

4) Chữa trị deadlock

Khi phát hiện deadlock, có 3 chiến lược:

1. Tịch thu tài nguyên (Preemption):

- Thu hồi tài nguyên từ các tiến trình để cấp cho các tiến trình khác.

- Không thể nhận tài nguyên là non-preemptive (Ví dụ: Máy in đang in dữ)

2. Hủy tiến trình (Process Termination)

- Chọn một hoặc nhiều tiến trình để hủy và giải phóng tài nguyên

- Cảnh báo nhắc:

- Hủy tiến trình có ảnh hưởng nhất

- Tránh làm mất dữ liệu hoặc gây mất mát quan hệ thống.

3. Roll back (Quay lại trạng thái trước đó)

- đưa tiến trình và trạng thái trở lại một checkpoint đã lưu trước đó.

- Tối bộ nhớ để lưu checkpoint và tối thiểu thời gian khôi phục trạng thái.

6. Chiến lược né tránh deadlock

(Deadlock Avoidance Strategy)

1) Khái niệm cơ bản

- Chiến lược né tránh deadlock tập trung vào việc duy trì và ngăn chặn các tình huống dẫn

dùn' dead lock trước khi chúng xảy ra.

- Khi hệ thống cần phải tài nguyên, cần kiểm tra xem việc cấp phát này có đưa hệ thống vào trạng thái không an toàn hay không. Nếu có, tài nguyên không được cấp phát, và process yêu cầu tài nguyên sẽ bị đưa vào trạng thái chờ.

2) Trạng thái an toàn và không an toàn

- Trạng thái an toàn: Là trạng thái mà hệ thống không bị deadlock vì có ít nhất một chuỗi các process có thể thực thi dùn' bì hoàn tất.

• Ví dụ: Với các process và tài nguyên còn lại nếu có thể sắp xếp các process hoàn tất mà ko dẫn đến deadlock, thì hệ thống đang ở trạng thái an toàn.

- Trạng thái không an toàn: Là trạng thái mà hệ thống có nguy cơ bị deadlock nếu tiếp tục cấp phát tài nguyên.

3) Quy trình chuyển trạng thái:

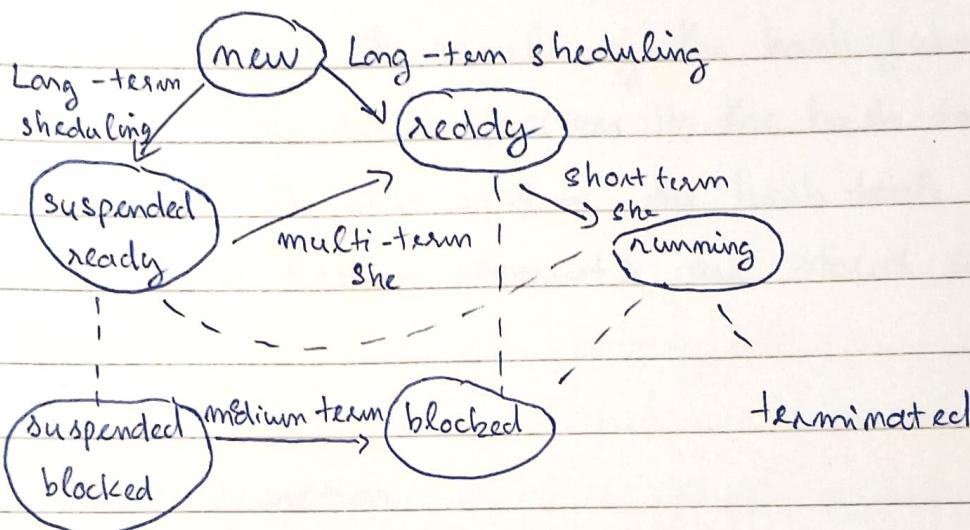
TUẦN 8.

Các khái niệm cơ bản:

Các giải thuật định thời

- First Come, First Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time (SRTF)
- Round Robin (RR)
- Priority Scheduling

Các bộ định thời



Hai yếu tố của giải thuật định thời

→ Không trung dung (Non-preemptive)

Không có trạng thái running, process sẽ chờ

thì cho đến khi kết thúc hoặc block do I/O

→ Trung dung (Preemptive)

Process đang chạy có thể bị ngắt

Chi phí cao hơn non-preemptive

TITLE:

Date: _____

Note: SJF & ché ôtô Preemptive chính (a)
SRTF