Ôn tập thi cuối kỳ (Chương 5 → Chương 8)
Sinh viên tự luyện tập, không cần nộp bài

Ôn tập:
Đồng bộ hoá (Synchronization)

```
Int availTicket = 1;
```

```
Client A ( ) {
    if (availTicket > 0) {
        availTicket -= 1;
    }
}
```

```
Client B ( ) {
    if (availTicket > 0) {
        availTicket -= 1;
    }
}
```

Critical Section (CS) (Miền găng):
Code segment in which multiple processes/threads access shared resources concurrently

(e.g., availTicket)

→ Solution: synchronization (Đồng bộ hoá)

Ôn tập:
Đồng bộ hoá (Synchronization)

- Mutual Exclusion (Độc quyền truy xuất)
  - ✓ Only one process/thread can be executing inside a critical section at a time
- Progress (Có sự tiến triển)
  - ✓ A process/thread outside a critical section cannot block others to enter this critical section
- Bounded Waiting (Không có tiến trình nào phải đợi vô hạn để vào miền găng)
  - ✓ Processes/Threads should not wait indefinitely for entering a critical section

Ôn tập:
Đồng bộ hoá (Synchronization)

Busy Waiting

while (*no permission to enter CS*);
critical_section;

- **Software solutions**
  - ✓ Lock variable
  - ✓ Strict Alternation
  - ✓ Peterson's solution
- **Hardware solutions**
  - ✓ Interrupt disabling
  - ✓ TSL

Sleep and Wakeup

if (*no permission to enter CS*) **sleep**();
critical_section;
wake_up(another);

- ***Semaphore***
  - ✓ *Binary semaphore (mutex)*
  - ✓ *Counting semaphore*
- **Monitor**

Ôn tập:
Đồng bộ hoá (Synchronization)

```
semaphore S = value;
down(S)
critical_section
up(S)
```

```
typedef struct semaphore{
    int value;
    struct thread *list; //blocking threads
} semaphore;
```

```
down(semaphore *S) {
    S→value--;
    if (S→value < 0) {
        add calling thread to S→list;
        block();
    }
}
```

```
up(semaphore *S) {
    S→value++;
    if (S→value <= 0) {
        remove a thread A from S→list;
        wakeup(A);
    }
}
```

Ôn tập:
Đồng bộ hoá (Synchronization)

```
semaphore mutex = 1;
processA (){
        down(mutex);
        critical_section
        up(mutex);
}
```

```
semaphore count = 5;
processX (){
        down(count);
        critical_section
        up(count);
}
```

```
semaphore mutex = 0; //synchronize execution order: threadB ➔ threadA
processA (){
        down(mutex);
        ...
}
```

```
processB (){
        ...
        up(mutex);
}
```

CÂU 1

Cho đoạn code bên dưới trong đó biến X được chia sẻ bởi hai tiến trình P1, P2.

a. Giả sử X được khởi tạo ban đầu là 0. X có khả năng vượt quá 20 không? Giải thích.

b. Sử dụng giải pháp đồng bộ hoá đảm bảo X không thể vượt quá 20.

```
Semaphore mutex = 1;
do
    {
        down(mutex);
        X = X +1;
        if ( X == 20) X = 0;
        up(mutex);
    }while ( TRUE );
```

a. Xét tình huống sau:
   - P1 tăng X lên 19 ➔ P1 chạy lệnh If ➔ Hệ thống thu hồi CPU cho P2
   - P2 tăng X lên 20 ➔ Hệ thống thu hồi CPU trả cho P1 ➔ P1 có thể tăng X???

b. Dùng semaphore mutex lock biến X trong lúc cập nhật.

CÂU 2

Cho đoạn code bên dưới được chia sẻ bởi hai tiến trình P1, P2. Giải pháp này có đảm bảo độc quyền truy xuất không. Giải thích.

```
while (TRUE) {
int j = 1-i;
flag[i]= TRUE; turn = i;
while (turn == j && flag[j]==TRUE);
critical-section ();
flag[i] = FALSE;
   Noncritical-section ();
}
```

Xét tính huống (giả sử: P1, P2 lần lượt có i = 0, 1):

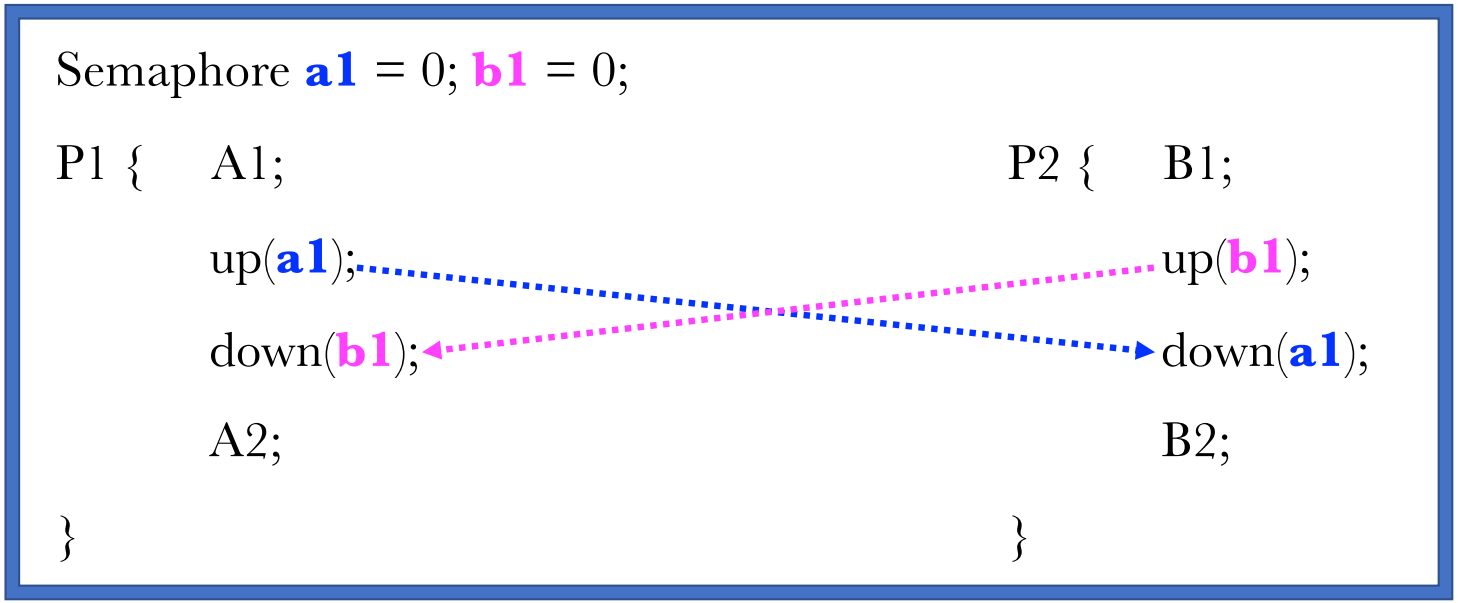
- P1 thực thi gán flag[0] = 1
- Hệ thống lấy lại CPU cho P2
- P2 thực thi gán flag[1] = 1, turn = 1
- P2 vào miền găng
- Hệ thống lấy lại CPU cho P1
- P1 gán turn = 0
- P1 vào miền găng

➔ Cả P1 và P2 đều ở trong miền găng.

CÂU 3

Cho hai tiến trình P1 và P2 có hoạt động như bên dưới:
Đồng bộ hoá hoạt động đảm bảo cả **A1 và B1** đều phải hoàn thành thì **A2 hay B2** mới được phép bắt đầu.

```
P1 { A1; A2; }
P2 { B1; B2; }
```

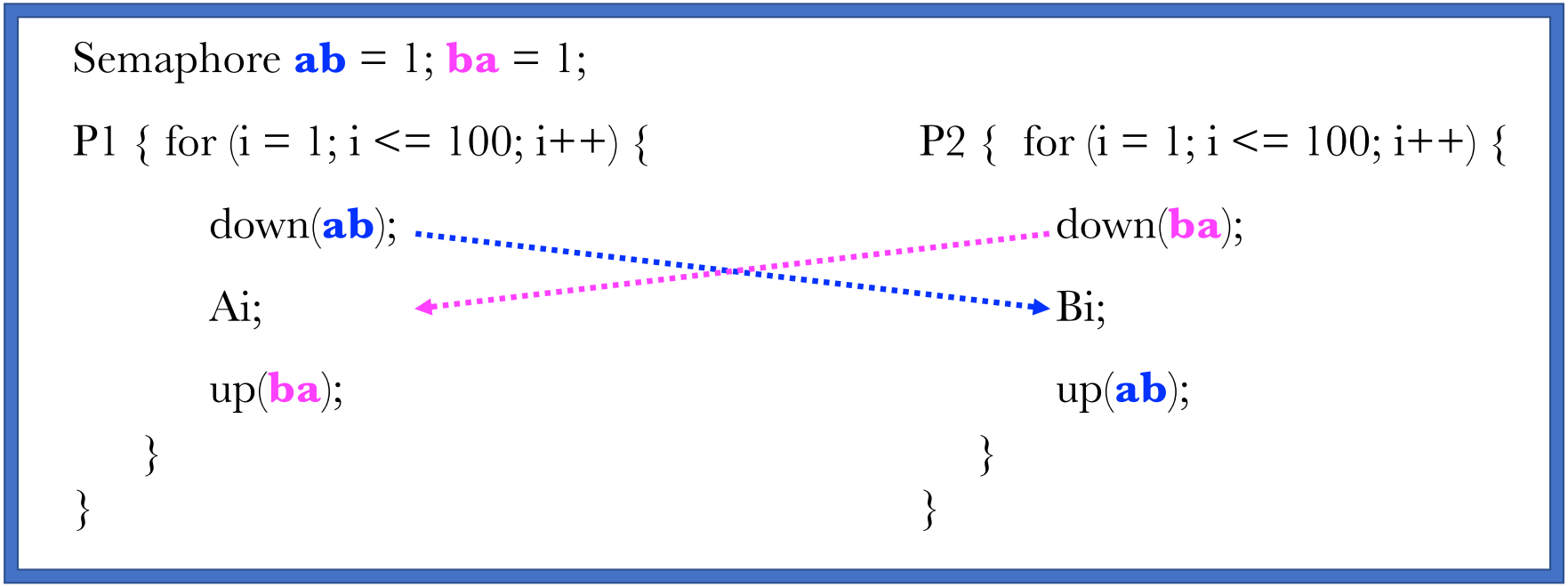


CÂU 4

Đồng bộ hoá hoạt động của P1 và P2 sao cho với k bất kỳ (2<=k<=100), Ak chỉ được bắt đầu khi B(k-1) kết thúc, và Bk chỉ bắt đầu khi A(k-1) kết thúc.

P1 { for (i = 1; i <= 100; i ++) Ai }

P2 { for (j = 1; j <= 100; j ++) Bj }



CÂU 5

Cho mảng sau: int x[20];

Dùng semaphore đồng bộ hoá cho 3 threads B, C, D thực hiện các nhiệm vụ sau trên tiêu chí khai thác tối đa khả năng xử lý song song, chia sẻ tài nguyên dùng chung giữa các threads.

a. B tính tổng giá trị các phần tử của mảng x có chỉ số chẵn
b. C tính tổng giá trị các phần tử của mảng x có chỉ số lẽ
c. D tính tổng giá trị các phần tử của mảng x dựa trên kết quả từ B và C

Giả sử: B, C, D cùng đến hệ thống tại 1 thời điểm, có thể kết thúc công việc mà không cần chờ đợi nhau.

```
Semaphore bd = 0; cd = 0;
sumB = 0; sumC = 0;
B {
    for (i = 0; i < 9; i++)
        sumB += x[i];
    up(bd);
}
```

```
C {
    for (i = 1; i < 9; i++)
        sumC += x[2*i+1];
    up(cd);
}
```

```
D {
    down(bd);
    down(cd);
    sum = sumB + sumC;
}
```

CÂU 6

Một hãng sản xuất xe ôtô có các bộ phận hoạt động song song:

+ Bộ phận sản xuất khung xe:

```
MakeChassis() { //Sản xuất ra một khung xe
        Produce_chassis();
}
```

+ Bộ phận sản xuất bánh xe:

```
MakeTire() {            //Sản xuất ra một bánh xe
        Produce_tire();
}
```

+ Bộ phận lắp ráp: Sau khi có được 1 khung xe và 4 bánh xe thì tiến hành lắp ráp 4 bánh xe này vào khung xe:

```
Assemble(){             //Gắn 4 bánh xe vào khung xe
Put_4_tires_to_chassis();
}
```

Hãy đồng bộ hoạt động của các bộ phận trên thoả các nguyên tắc sau:

Tại mỗi thời điểm chỉ cho phép sản xuất ra 1 khung xe. Cần chờ có đủ 4 bánh xe để gắn vào khung xe hiện tại này trước khi sản xuất ra một khung xe mới.

CÂU 6

```
semaphore sC = 0, sT = 0, sA = 1
```

MakeChassis {

```
    down(sA);
    produceOneChassis();
    up(sC);
}
```

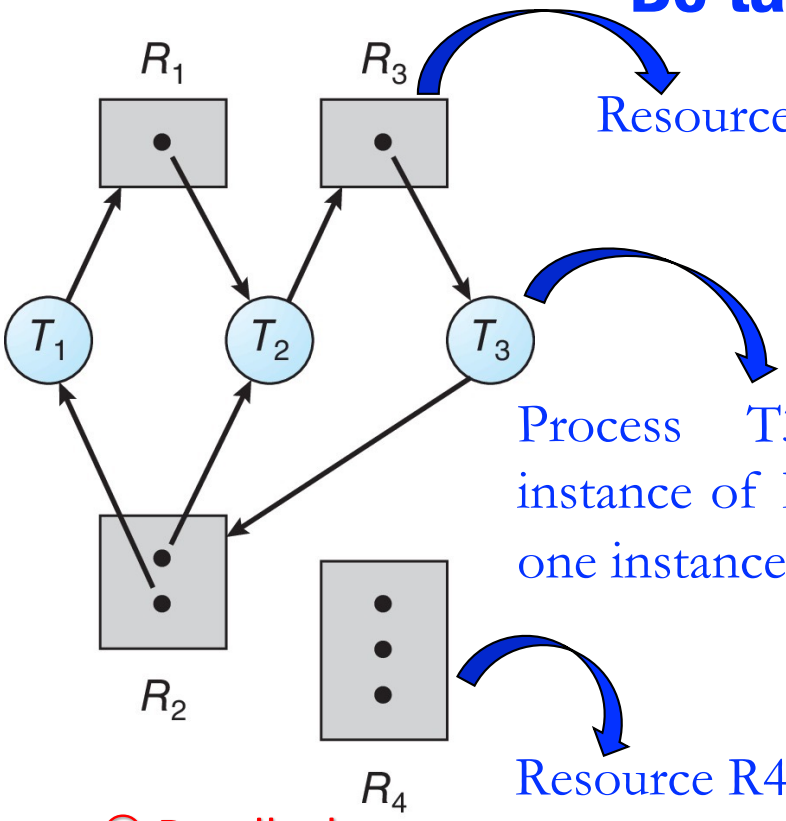
MakeTire {

```
    produceOneTire();
    up(sT);
}
```

Assemble {

```
    down(sC);
    down(sT);
    down(sT);
    down(sT);
    down(sT);
    put4TiresToChassis();
    up(sA);
}
```

Ôn tập:
Bế tắc (Deadlock)

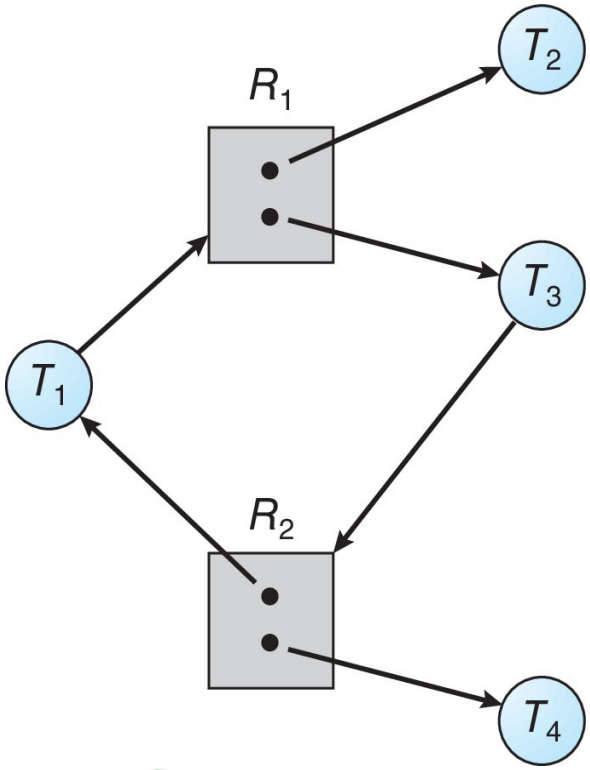


Resource R3 has one instance

Process T3 holds one instance of R3 and requests one instance of R2

Resource R4 has three instances

☹ Deadlock



☺ No Deadlock

Ôn tập:
Bế tắc (Deadlock)

A set of processes is in a **deadlock** situation when each of them is waiting for resources allocated to other waiting processes in the set, therefore none of them can get all necessary resources to continue executing.

Ôn tập:
Bế tắc (Deadlock)

All four following conditions must take place to cause deadlock

- **Mutual Exclusion**
  - ✓ Only one process can hold the resource at a time
- **Hold and Wait**
  - ✓ Process holds some resources while waiting for others, which are held by other waiting processes
- **No preemption**
  - ✓ A resource can be released only by the process holding it
- **Circular Wait**
  - ✓ There is a wait-cycle between at least two processes

Ôn tập:
Bế tắc (Deadlock)

- **Deadlock Handling**
  - *Ignorance (Bỏ qua)*
  - *Prevention (Phòng ngừa một trong 4 điều kiện)*
  - *Avoidance (Phòng tránh bằng thuật toán)*
  - *Detection and Recovery (Phát hiện và khôi phục)*

Ôn tập:
Bế tắc (Deadlock)

Example of Banker's algorithm

	Allocation			Max			Need		
	C	D	E	C	D	E	C	D	E
P_0	0	1	0	7	5	3	7	4	3
P_1	2	0	0	3	2	2	1	2	2
P_2	3	0	2	9	0	2	6	0	0
P_3	2	1	1	2	2	2	0	1	1
P_4	0	0	2	4	3	3	4	3	1

Available (Work)		
C	D	E
3	3	2

The system is safe?

One possible safe sequence: (P1, P3, P0, P2, P4)

Ôn tập:
Bế tắc (Deadlock)

Example of Banker's algorithm

	Allocation			Max			Need		
	C	D	E	C	D	E	C	D	E
P_0	0	1	0	7	5	3	7	4	3
P_1	3	0	2	3	2	2	0	2	0
P_2	3	0	2	9	0	2	6	0	0
P_3	2	1	1	2	2	2	0	1	1
P_4	0	0	2	4	3	3	4	3	1

Available (Work)		
C	D	E
2	3	0

One possible safe sequence: (P1, P3, P0, P2, P4)

$Request_{P1}$ = (1, 0, 2) will be granted safely?

$Request_{P1} < Need_{P1} < Available$

➔ $Request_{P1}$ granted ➔ Update $Allocation_{P1}$, $Need_{P1}$, Available

CÂU 8

- Xét trạng thái hệ thống với các loại tài nguyên A, B, C, và D

	Max				Allocation			
	A	B	C	D	A	B	C	D
P0	4	4	1	3	2	0	1	2
P1	1	6	5	0	1	0	4	0
P2	5	4	5	6	1	3	5	2
P3	0	6	5	2	0	6	3	2
P4	0	6	6	6	0	0	1	2

Available			
A	B	C	D
2	6	2	1

- Xác định nội dung bảng Need
- Hệ thống có ở trạng thái an toàn không?
- Nếu tiến trình P2 có yêu cầu thêm tài nguyên (4,0,0,4), yêu cầu này có được đáp ứng ngay lập tức hay không?

Ôn tập:
Bộ nhớ và bộ nhớ ảo (Memory & Virtual Memory)

- Cơ chế cấp phát:
  - **Liên tục (Contiguous Allocation)**
    - ✓ Phân vùng động (Dynamic/Variable partitions)
    - ✓ Phân vùng cố định (Fixed partitions)
  - **Không liên tục (Non-contiguous Allication)**
    - ✓ Phân đoạn (Segmentation)
      - <s, d> --Bảng phân đoạn (segment table)-- <base, limit>
    - ✓ Phân trang (Paging)
      - <p, d> --Bảng phân trang (page table)-- <f, d>

Ôn tập:
Bộ nhớ và bộ nhớ ảo (Memory & Virtual Memory)

- Các kỹ thuật xử lý bảng trang lớn?
- Chuyển đổi địa chỉ tuyệt đối <==> tương đối?
- Tính EMAT khi có/không có lỗi trang? Khi sử dụng/không sử dụng TLB?
- Các kỹ thuật thay thế trang: FIFO, Optimal, LRU, FIFO Cơ hội 2, …?

CÂU 7

Hãy dùng chiến lược thay thế trang FIFO, LRU, FIFO Cơ hội 2 để thực hiện thay thế trang cho các tiến trình sau có nhu cầu dùng bộ nhớ trong quá trình hoạt động. Chuỗi truy xuất trang như bên dưới. Tính tỉ lệ lỗi trang phát sinh cho từng chiến lược. Giả sử ban đầu hệ thống có 4 khung trang và tất cả đều còn trống.

8, 2, 1, 2, 3, 0, 2, 1, 8, 4, 3, 0, 8, 1, 0, 2, 3

Minh hoạ: Dùng LRU

	8	2	1	2	3	0	2	1	8	...
f1	8	8	8	8	8	0	0	0	0	
f2		2	2	2	2	2	2	2	2	
f3			1	1	1	1	1	1	1	
f4					3	3	3	3	8	
Page fault (Lỗi trang)	*	*	*		*	*			*	

CÂU 7 (tt)

Dùng FIFO Cơ hội 2: Chọn trang đầu tiên trong FIFO làm để xét nạn nhân, nếu bit referenrence=1, cho cơ hội 2 bằng cách gán bit reference = 0 và cho xuống cuối hàng đợi FIFO. Nếu trang bị xét có bit reference = 0 thì thay thế luôn.

	8	2	1	2	3	0	2	1	8	...
f1	8	8	8	8	8	0	0	0	0	
f2		2	2	2	2	2	2	2	2	
f3			1	1	1	1	1	1	1	
f4					3	3	3	3	8	
FIFO	8	82	821	821	8213	2130	2130	2130	0218	
Reference Bit	8	82	821	821	8213	0	02	021	08	
Page fault (Lỗi trang)	*	*	*		*	*			*	

CÂU 9

Một máy tính có 48 bit địa chỉ ảo, và 32 bit địa chỉ vật lý. Kích thước một trang là 8K. Có bao nhiêu phần tử trong một bảng trang (thông thường)? Trong bảng trang nghịch đảo?

8K = 2^13 ➔ dùng 13 bit lưu d ➔ dùng 48-13 = 35 bits lưu số trang p ➔ có tối đa 2^35 phần tử trong bảng trang (thông thường).

Phần tử trong bảng trang nghịch đảo (inverted page table) = số khung trang trong bộ nhớ vật lý = 2^19 (do cần 32-13=19 bits để lưu số hiệu khung f ➔ bộ nhớ vật lý chia thành 2^19 khung trang)

CÂU 10

Một máy tính sử dụng bộ nhớ ảo phân trang, biết rằng bộ nhớ ảo sử dụng 10bit địa chỉ, RAM 1KB, kích thước mỗi trang là 256 bytes.

a. Số lượng trang tối đa cho một tiến trình thực thi trên hệ thống này là bao nhiêu?
b. Giả sử sử dụng bảng trang nghịch đảo thì số lượng entry trong bảng trang là bao nhiêu.
c. Thử đề xuất một giải pháp phân trang đa cấp để quản lý địa chỉ ảo của tiến trình.

a. Kích thước trang 256 = 2^8 (bytes) ➔ dùng 8 bit lưu d ➔ 2 bits lưu số trang p ➔ có tối đa 2^2 = 4 trang cho một tiến trình thực thi trên hệ thống này.

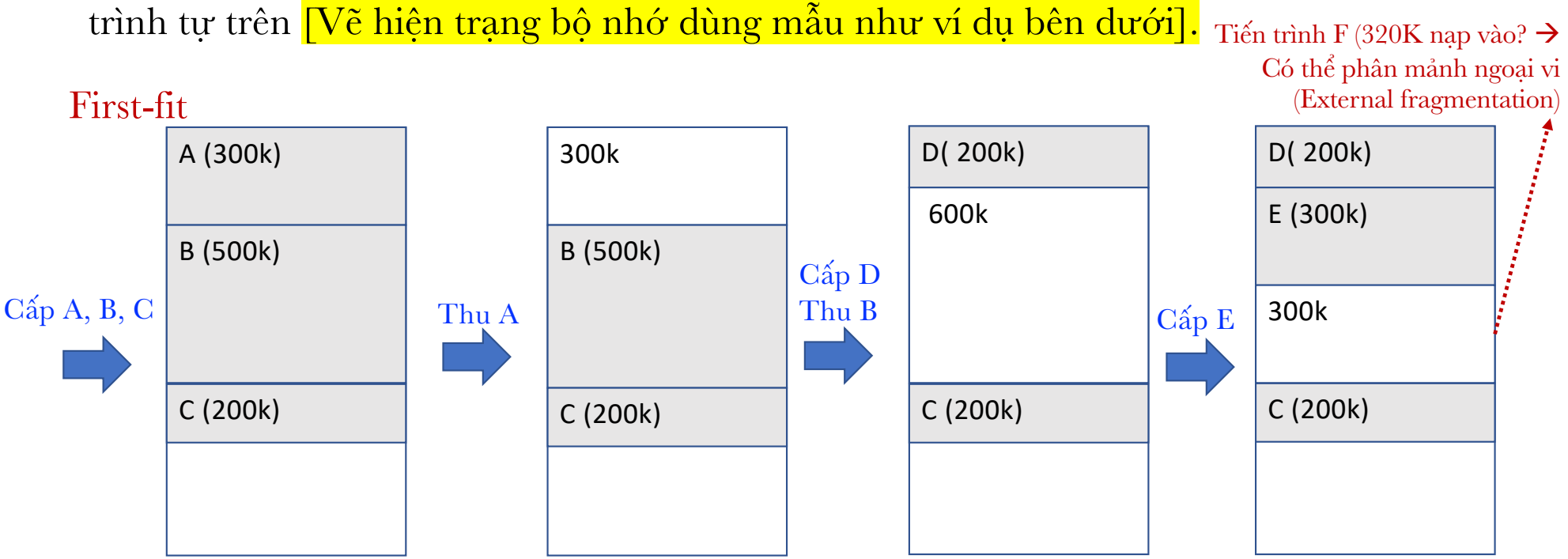
b. 1KB RAM = 2^10 (bytes)/2^8 (bytes) = 4 khung trang ==> Phần tử trong bảng trang nghịch đảo (inverted page table) = số khung trang trong bộ nhớ vật lý = 4

c. (p1 = 2, p2 = 2, d=8)

CÂU 11

Cho các tiến trình có bộ nhớ tương ứng A(300K), B(500K), C(200K), D(200K), E(300K). Sử dụng giải thuật First-fit, Best-fit, Worst-fit (trong kỹ thuật phân vùng động) cấp phát bộ nhớ theo trình tự : A→B→C→thu hồi A →D→thu hồi B→E với dung lượng bộ nhớ dùng để cấp phát là 2000k.

a. Cho biết hiện trạng bộ nhớ và danh sách quản lý bộ nhớ ở các thời điểm cấp phát theo trình tự trên [Vẽ hiện trạng bộ nhớ dùng mẫu như ví dụ bên dưới].



Ôn tập:

Quản lý tập tin và ổ đĩa (File & Disk Management)

Sinh viên tự xem lại file video và bài tập demo ☺.