Review OS

Monday, January 8, 2024 9:44 AM

Semaphore

1. **Critial Section Problem**
- **Race condition**: Nhiều thread chia sẽ một biến dùng chung, đồng thời cùng tranh đoạt điều khiển gây ra kết quả không mong đợi.
- **Critical Section:** Là một đoạn code mà nhiều tiến trình hoặc tiểu trình có thể cùng cập nhật, thao tác dùng chung.

2. **Synchronization**
- **Tiêu chí Synchronization:**
  + Độc quyền truy xuất: chỉ một process/thread trong CS được thực thi tại 1 thời điểm
  + Progress: có sự tiến triển, một process/thread ở ngoài CS không thể block other để vào CS
  + Bounded waiting: không có tiến trình nào phải đợi vô hạn để vào miền găng
- **Phương pháp:**
  + Busy waiting: Software solution (Lock variable, Strict Alternation, Peterson's solution), Hardware solutions(Interrupt disabling, TSL)
  + Sleep and wakeup: Semaphore (binary and counting), Monitor (bị phụ thuộc vào NNLT)
- Lock variable: dùng 1 biến lock để kiểm tra tiến trình có đc vào CS ko => two process maybe insid CS at same time
- Strict Alternation: luân phiên => a thread outside CS may prevent another thread from entering CS
- Peterson's solution: CPU waiting, priority inversion

3. **Sleep and wakeup**

a. **Semaphore**

```
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);
  }
}
```

- Có 1 biến số nguyên được sử dụng để kiểm tra quyền truy cập vào CS qua 2 atomic operations:

+ Down: trước khi vào CS, process sẽ gọi hàm down để kiểm tra xem có quyền vào CS hay không

+ Up: Khi ra khỏi CS, process sẽ gọi hàm Up để báo hiệu việc ra khỏi CS và đánh thức thread còn lại

b. **Monitor:**
- Một cấu trúc lập trình được đóng gói gồm 3 thành phần chính:Biến dùng chung, Biến điều kiện (wait và signal) cho đồng bộ hóa, Procedures: tổ chức trên các biến dùng chung

Deadlock

1. Khái niệm:
- Nhiều tiến trình, trong đó mỗi tiến trình chờ tài nguyên đang được cấp phát cho tiến trình khác trong khi đang sở hữu tài nguyên nhất định =>ngưng hoạt động => deadlock
2. Điều kiện xảy ra deadlock: (phải có cả 4)
- Độc quyền truy xuất, chiếm giữ và chờ đợi, không chiếm đoạt(độc quyền), Tạo thành một chu trình ít nhất 2 process (circular wait)
3. Phương pháp:

a. Ignorant (bỏ qua)

b. Prevention (phòng ngừa 1 trong 4 điều kiện)

c. Avoidance (phòng tránh bằng thuật toán)

d. Detection and recovery (phát hiện và khôi phục): cho phép deadlock xảy ra và khôi phục lại hệ thống

Ví dụ về Semaphore

Ví dụ 1:

```
#define N 50
semaphore mutex, empty, full;
init (mutex, 1), init (empty, 0), init (full, N);
```

```
producer() {
    down(full);
    down(mutex);
    insertItem(item);
    up(mutex);
    up(empty);
}
```

```
consumer() {
    down(empty);
    down(mutex);
    item = removeItem()
    up(mutex);
    up(full);
}
```

Ví dụ 2:

✓ Several readers can read from database at the same time
✓ If a writer is writing to database, other threads cannot access database
✓ If there are any readers in database, a writer cannot access database

```
semaphore db, mutex;
init (db, 1), init (mutex, 1);
int nReaders = 0;
```

```
Reader () {
    down(mutex);
    if (nReaders==0) down(db);
    nReaders+=1;
    up(mutex);
    readFromDB();
    down(mutex);
    nReaders-=1;
    if (nReaders==0) up(db);
    up(mutex);
}
```

```
Writer () {
    down(db);
    writeToDB();
    up(db);
}
```

```
monitor DiningPhilosophers
{
  enum { THINKING, HUNGRY, EATING };
  state [5];
  condition self [5];
  void pickup (int i) {
    state[i] = HUNGRY;
    test(i);
    if (state[i] != EATING) self[i].wait;
  }
  void putdown (int i) {
```

```
void test (int i) {
  if (state[(i + 4) % 5] != EATING &&
      state[i] == HUNGRY &&
      state[(i + 1) % 5] != EATING ) {
    state[i] = EATING ;
    self[i].signal () ;
  }
}
initialization_code () {
  for (int i = 0; i < 5; i++)
    state[i] = THINKING;
```

Main memory

1. Định nghĩa
- Virtual/ logical address: Địa chỉ được tạo và quản lý bởi cpu. Địa chỉ mà các không gian tiến trình và cpu làm việc trên đó.
- Virtual address space: là tập hợp các địa chỉ ảo (program address space).
- Physical address: là địa chỉ thật trên bộ nhớ chính.
- Physical address space: khi tiến trình được thực sự nạp vào bộ nhớ, được đánh địa chỉ trên RAM
- Address binding: Thực hiện mapping 1 địa chỉ vào address space, có thể thực hiện tại Compile time, Load Time (linking và load vào bộ nhớ) và hầu hết ở giai đoạn Execution time (chỉ khi nào đang được thực thi mới xác định địa chỉ vật lý).
- Memory Management Unit (MMU): phần mạch tích hợp trong cpu, chuyển đổi địa chỉ cpu đang thao tác đến địa chỉ vật lý.

2. Contiguous memory allocation:
- Đặc điểm: Áp dụng cho hệ thống Main Frame. Toàn bộ tiến trình không thể được phân nhỏ, và phải được nạp vào một không gian liên tục của bộ nhớ chính. Physical memory được chia sẵn thành các partition hoặc các phân vùng có thể được hình thành trong quá trình nạp/đẩy
- Fixed-partition: Được tạo ra trước quá trình nạp tiến tình, kích thước có thể không phù hợp với tiến trình
- Variable-partition: Được tạo ra trong quá trình tiến trình được load, Kích thước có thể phù hợp với tiến trình

=> Không phù hợp với hệ thống hiện tại Nếu kích thước lớn.
- Chiến lược cấp phát:

a. First-fit:
- Ưu điểm: Đơn giản hiệu quả. Giảm sự phân mảnh Memory. Tốc độ nhanh
- Nhược điểm: Hiệu suất kém trong bộ nhớ bị phân mảnh nội vi. First - Fit có thể chọn phân bổ những vùng nhớ lớn hơn yêu cầu của tiến trình, dẫn đến lãng phí bộ nhớ và giảm khả năng đáp ứng các yêu cầu bộ nhớ khác

b. Best-fit:
- Ưu điểm: Tiết kiệm bộ nhớ. Giảm tình trạng external fragmentation.
- Nhược điểm: Tốc độ phân bổ chậm. Tăng chi phí tính toán. Gia tăng khả năng gây internal fragmentation.

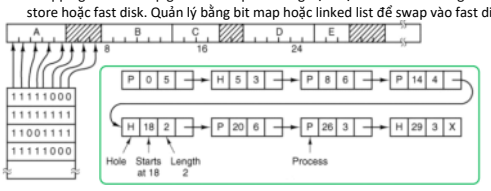
c. Worst-fit:
- Ưu điểm: Bằng cách chọn vùng nhớ trống lớn nhất, worst – fit tạo ra phân mảnh bên trong lớn, giúp tận dụng vùng nhớ để đặt các tiến trình nhỏ.
- Nhược điểm: Hiệu suất kém trong bộ nhớ bị phân mảnh nhiều. Phân bổ chậm do phải duyệt qua toàn bộ vùng nhớ để tìm vùng trống lớn nhất

- Address protection:được đánh từ 0 đến limit – 1, nếu hợp lý thì cộng với giá trị Relocation registe

+ Relocation register (thanh ghi tái định vị): lưu trữ địa chỉ physic đầu tiên của tiến trình trong bộ nhớ

+ Limit register: lưu trữ địa chỉ max

3. Swapping:Có thể swap giữa ram và phân vùng đặc biệt của ổ đĩa backing store hoặc fast disk. Quản lý bằng bit map hoặc linked list để swap vào fast disk.



- Total swaptime = 2* Transfer rate between memory and backing store

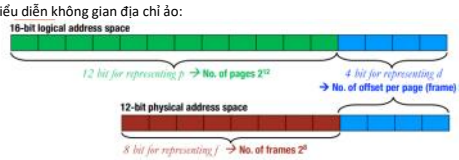
4. Fragmentation:
- External (ngoại vi): Khi các ô trống không liên tục, bị rời rạc (Tổng không gian trống đủ nhưng không liên tục để chứa tiến trình). Có thể dùng compaction để các khoảng trống lại về một phân vùng to hơn.
- Internal (nội vi): khi vùng nhớ được chia sẵn thành các partition kích thước cố định. Và kích thước không phù hợp với tiến trình => tạo ra không gian trống không sử dụng. Giải pháp: Best-fit allocation

Virtual memory

1. Định nghĩa:
- Dynamic linking: Linking được delay cho tới quá trình thực thi
- Dynamic loading: Khi CPU thuc thi tới đoạn nào thì routine đó mới được load
- Overlay: chỉ 1 phần tiến trình được thực thi sẽ được load vào bộ nhớ
- Noncontiguous memory allocation: chia tiến trình thành nhiều phần khác nhau, mỗi phần có thể load vào bộ nhớ chính và không cần liên tục.

2. Segmentation
- Mỗi tiến trình được chia thành các segment (đoạn) khác nhau rời rạc được nạp vào bộ nhớ chính không cần liên tục. Cần thanh ghi Base(địa chỉ đầu tiên) và Limit(độ dài)
- Địa chỉ logic: <s: segment number, d: offset (0>Limit -1)>

3. Paging
- Tiến trình sẽ được chia thành các page có kích thước bằng nhau, còn bộ nhớ vật lý chia thành các frame, và các page có thể được nạp không liên tục. Trang phải được nạp trọn vẹn vào một khung trang.
- Địa chỉ tương đối logic <p: page: d: offset >
  Địa chỉ tương đối vật lý <f: frame, d: offset>
- Địa chỉ tuyệt đối = p*No.offset perpage + d
- Biểu diễn không gian địa chỉ ảo:



Each address points to a byte-addressable in the memory

(Note: If memory is word-addressable, each address, generally, points to a 4-byte word)

→Size of a page (and frame): 2^4 bytes = 16 bytes
→Size of logical address space: 2^{16} bytes = 64KB
→Size of physical memory: 2^{12} bytes = 4KB

- Page table: dùng để mapping địa chỉ ảo sang địa chỉ vật lý.

An example of Page Table Entry Structure

Frame Number	Valid/Invalid	Protection	Caching	Referenced	Modified

+ Valid/Invalid: 1/0 cho biết page đã được load vào bộ nhớ hay chưa

File system

File interface

1. File: Tập tin là một khái niệm trừu tượng, đơn vị lưu trữ ở mức logic, trừu tượng hoá byte dữ liệu trong ổ đĩa.
- Các loại file:
  + Ordinary/Regular File: ASCII or bin file
  + Directory file: chứa nội dung là danh sách các file
  + Shortcut file: ko chứa nội dung và liên kết đến 1 file khác
  + Special file: dành riêng cho thiết bị nội vi
- Thuộc tính: Name, size, location, creator, date, type, permission/ protection, etc.
- Cách truy cập: Sequential tuần tự từng block theo đúng thứ tự, Direct random truy cập trực tiếp, indexed sequential dựa trên index để truy cập.
- Cấu trúc file: là chuỗi các byte, chuỗi các record hoặc cây record

2. Thư mục:
- Thư mục: 1 tập hợp các file/thư mục con khác
- Có 2 loại đường dẫn:
  + Absolute path: C:\user\home\myfile.doc
  + Relative path:.\myfile.doc
- Tổ chức thư mục:
  + Tiêu chí tổ chức: hiệu quả, naming, grouping cability according to user, sharing
  + Single level dễ cài đặt và tìm kiếm nhưng gặp vấn đề naming và grouping.
  + Two-level: giải quyết vấn đề naming và tìm kiếm nhưng vẫn gặp vấn đề gouping (tầng1 cho người dùng).
  + Tree-Structure: chia thành nhiều tầng hỗ trợ naming, grouping, searching nhưng truy cập phức tạp, vấn đề chia sẽ bộ nhớ.
  + Graph: hỗ trợ link thư mục này sang thư mục khác, cho phép chia sẽ dữ liệu nhưng tốn chi phí.
- Protection: dùng Access Control List, gắn từng file để định quyền Read, Write, Execute. 3 class user: Owner, group, universe (public). Dãy gồm 9 bit theo thứ tự class và theo RWX. Thêm 1 bit trước 9 bit để xác định thư mực(d) hay file(-)

File System Implementation

1. File system: cung cấp cho người dùng cách truy cập file dễ dàng mà không cần quan tâm đến cấu trúc vật lý của thiết bị lưu trữ. Quản lý và map file vào disk drives(cấp phát,..)
2. Disk drive: có thể chia thành nhiều partitions. Mỗi vùng phải cài đặt file system có thể khác nhau.
3. Windows File System:
- FAT (flash disk, removable device): file allocation table, có tính tương thích cao với nhiều thiết bị do cũ chứa được tối đa 4GB/ file. exFat là phiên bản tối ưu hơn có thể lưu trữ 16EB/file nhưng tương thích kém hơn. Không bảo mật, backup restore, lưu nhật ký
- NTFS (window): Là hệ thống tập tin mới, bảo mật, ghi nhật ký, nén file, … hỗ trợ tối đa 16EB/file.
4. Linux File System:
- Hỗ trợ journaling (check sum), 16GB -> 16TB /file. Một thư mục tối đa có 6400 thư mục con HFS, HFS+, APES (MacOS).
5. File and Directory Implemenntation:

a. Contigious Allocation:
- Contiguous Allocation: cấp liên tục, Tìm các block đĩa còn trống liên tục nhau để lưu file. Lưu vị trí Start và chiều dài block tính luôn block ban đầu
- Ưu: dễ thực hiện, hiệu suất cao
- Nhược: gây phân mảnh, file lớn thì không đủ không gian, và phải biết kích thước file để cấp phát

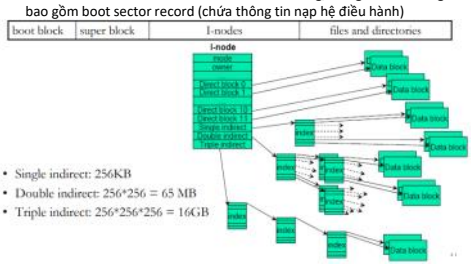
b. Linked List Allocation:
- Dùng linked list, lưu block start và end, block n lưu địa chỉ của block n + 1.
- Ưu: Giải quyết vấn đề phân mảnh, linh hoạt
- Nhược: Mở rộng nhưng không hỗ trợ random access, cần một không gian trong block để lưu pointer, mất 1 block sẽ mất liên kết.

c. Index Allocation:
- Index block, block chứa tất cả các block chứa dữ liệu trong file theo thứ tự.
- Ưu: Cho phép random access, xử lý chuyện 1 block bị mất.
- Nhược: Không phù hợp file nhỏ do phải dùng 1 block để lưu index gây lãng phí

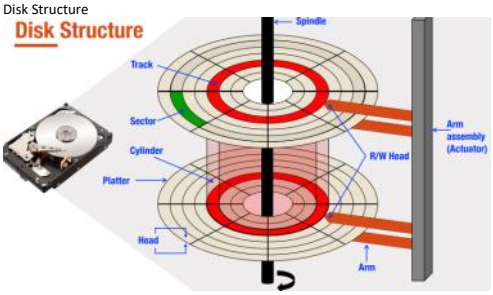
-----Window: Linked list + File Allcation table để lưu truy vết của tất cả các block.

-----UNIX: dùng: Indexed Allocation (I nodes) multilevel indexed allocation. Khớp từng loại kích thước file.

6. Free-space management: dùng bit map, linked list, grouping (block đầu tiên chứa địa chỉ các block free), counting
7. Directory implementation: linear linked list hoặc hashed table
8. Disk Layout:
- Master boot record: sector đầu tiên của đĩa cứng, bao gồm các thông tin về OS booting, bao gồm boot sector record (chứa thông tin nạp hệ điều hành)



Disk Structure



CHS to LBA address

6. Semaphore chassis=1, tire=4, sem_c=0, sem_t=0 (trong trường hợp đủ 4 mới tạo bánh mới)

MakeChassis(){	MakeTire(){	Assemble(){

down(chassis)
Produce_chassis();
up(sem_c);
}
	down(tire)
	Produce_tire();
	up(sem_t);
	}
		down(sem_c);
		down(sem_t);
		down(sem_t);
		down(sem_t);
		down(sem_t);
		
		Put_4_tire_to_chassis();
		up(chassis);
		up(tire); up(tire);
		up(tire); up(tire);
		}

Ví dụ về bế tắc

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

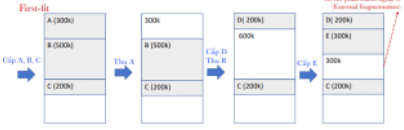
Một Request được granted khi Request < Need <Available => Nếu thỏa thì kiểm tra xem dãy sau request có an toàn hay ko => Thêm request thì nhớ cập nhật lại Allocation, need, available

a. Consider a system consisting of 4 resources of the same type that are shared by 2 processes, each of which needs at most 3 resources. Can deadlock occur? Explain your answer.
   - Có thể xảy ra trong trường hợp: mỗi tiến trình giữ 2 R, và chờ 1 R
b. Consider a system consisting of m resources of the same type that are shared by 4 processes, each of which needs at most 3 resources. What is the minimum value of m that ensures no deadlock? M=4*2+1=4
c.

Ví dụ về Paging

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 mảu như ví dụ bên dưới].



- Với page size =1024

P1. 1024	P1.2048	P2.2050	P1.5120	P1.1090	P2.1030	P1.6000	P2.7050	P1.5120	P2.0
P1-1	P1-2	P2-2	P1-5	P1-2	P2-1	P1-5	P2-6	P1-5	P2-0

- Page table: dùng để mapping địa chỉ ảo sang địa chỉ vật lý.

An example of Page Table Entry Structure

Frame Number	Valid/Invalid	Protection	Caching	Referenced	Modified

+ Valid/Invalid: 1/0 cho biết page đã được load vào bộ nhớ hay chưa

4. Page fault handling: nếu page được truy cập chưa được nạp vào (invalid) => báo lỗi CPU => OS đi ra tìm page cần tìm load vào

- **Effective memory-accestime (EAT or EMAT):**

$$EAT = (1 - p) * t_m + p * t_p$$

p: page fault ratio (probability of occurring a page fault)
t_m: memory-access time
t_p: page-fault service time (swap-in, swap-out, restart instruction, ...)

- Translate Lookaside Buffer: Là bộ nhớ high-speed cache chứa một vài page table entry mà thường xuyên được sử dung (thường từ 64-1024 entrys)

$$EAT = h * (t_c + t_m) + (1 - h) * (t_c + 2 * t_m)$$

h: TLB hit ratio (probability of finding a desired page in TLB)
t_m: memory-access time
t_c: TLB lookup time

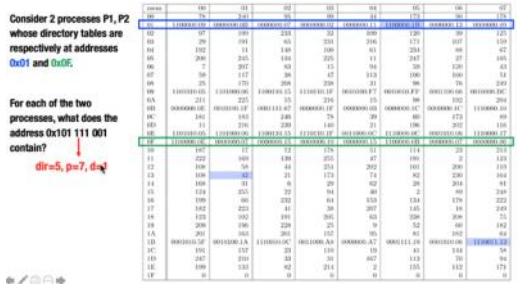
1 for Page Table access
1 for memory access

4. Page replacement: trong trường hợp không có frame trống, chọn 1 trang làm victim page để thay thế

- FIFO: page nào vào trước thì ra trước. Ưu: đơn giản. Nhược: belady's anomaly
- Optimal:chọn trang có ít refer nhất trong tương lai. Ưu: nhanh, hiệu quả. Nhược: ko biết được tương lai
- LRU: chọn trang ít refer nhất trong quá khứ.
- Second change: mỗi page sẽ có 1 cơ hội thứ 2 nếu reference bit là 1

	7	0	1	2	0	3	0	4	2	3	0
Frame 1	7	7	7	2	2	2	2	4	4	4	4
Frame 2		0	0	0	0	0	0	0	2	2	2
Frame 3			1	1	1	3	3	3	3	3	0
Fifo	7	70	701	012	012	203	203	034	342	342	420
Referenced bit	7	70	701	2	20	23	230	4	42	423	0
	*	*	*	*		*		*	*		*

5. Paging with large address space:
   - Hierarchical (Multilevel) Paging: tổ chức page table thành nhiều cấp
   - Inverted Page Table <=> Page table size = Frame size
   - Hash Page Table



6. Consider a system that implements a two-level page table with a 32-bit address: 9-bit used for the top-level page table; 11-bit used for the second-level page table. The system has a word-addressable memory of 10GB.
   a) Give the size of a page
   b) Give the number of frames in the memory
   c) What is the maximum size of process space supported in this system?
   d) If loading the process P1 of 2.8GB into this system, may we suffer from the fragmentation problem? Justify your answer.
   e) Which type of fragmentation can we have in a paging system?

a. Page size: $2^{32}.4=2^{34}$ byte=16KB

b. No.frames: 10GB/2^{34} bytes=655360

c. Page size: $2^{32}.4=2^{34}$ byte

Second-level page table has 2^{11} pages => size: $2^{11}.2^{34}$

Top level page table has 2^9 second-level page table => size: $2^9.2^{11}.2^{34}=2^{34}$ byte=16GB

d. 2.8GB = 183500.8pages => internal fragment

e. Internal Fragment

CHS to LBA address

$$LBA = (C * N_{heads} + H) * N_{sectors} + (S - 1)$$

LBA to CHS address

$$C = LBA / (N_{heads} * N_{sectors})$$
$$H = (LBA / N_{sectors}) \% N_{heads}$$
$$S = (LBA \% N_{sectors}) + 1$$

Where:
- C: cylinder | track number
- H: head number
- S: sector number
- N_{heads}: no. of heads | tracks per cylinder
- $N_{sectors}$: no. of sectors per track

1. Disk Access Time = Seek Time + Rotational Time + Data Transfer Time
   - Seek Time: thời gian để chuyển đầu đọc/viết đến track/cylinder tương ứng
   - Rotational Time: thời gian chuyển đầu đọc/viết đến sector tương ứng
   - Data Transfer Time: tời gian transfer data từ đĩa
2. Disk Scheduling Algorithms:
   - FCFS, SSTF(chọn track gần nhất để move tới)
   - SCAN: Đi đến đỉnh quay về gặp track nào xử lý track đóC-SCAN Lên đỉnh về đáy rồi lên lại.
   - LOOK: Lên track trên xa nhất Xuống duyệt theo như scan. C-LOOK lên track xa nhất về cũng thg xa nhất rồi lên lại.

Ví dụ về File

A 255-GB disk has 65,536 cylinders with 255 sectors per track and 512 bytes per sector. How many platters and heads does this disk have? Assuming an average cylinder seek time of 11 msec, average rotational delay of 7 msec and reading rate of 100 MB/sec, calculate the average time it will take to read 400 KB from one sector

- Một platter có 2 heads
- 225GB=512.255.65536(cylinders).Nhead
  ⇨ Nheads=32
  ⇨ N_platters=16
- Seektime:11msec
- Rotation: 7msec
- Transfer time:400Kb/100MB=3,9msec
  ⇨ Disk access time: 5ms

Consider the Unix I-node which uses 12 direct data block addresses, 1 single indirect, 1 double indirect, and 1 triple indirect. The disk block address requires 32 bits and disk block size is 1 KB. What is the maximum size?

- 12 direct data block => 12KB
- No.pointers per disk block: 1KB/32bit=256
- 1 single indirect: 256*1KB=256Kb
- 1 double indirect: 256*256*1KB=2^{16}KB
- 1 triple indirect: 2^{24}KB
  ⇨ Maximum: 16GB

Question#5: FCFS

Consider a disk request for disk blocks on cylinders as follows: 93, 182, 37, 20, 122, 185, 14, 120, 65, 67. The cylinders are numbered from 0 to 199. Assume the current position of R/W head is at 53. For each of FCFS, SSTF, SCAN, C-SCAN, LOOK, and C-LOOK disk scheduling algorithms:

- Give the total number of R/W head movement while servicing these requests? 790
- Assume a seek takes 6 msec per cylinder. How much seek time is needed for each algorithm? 6.74