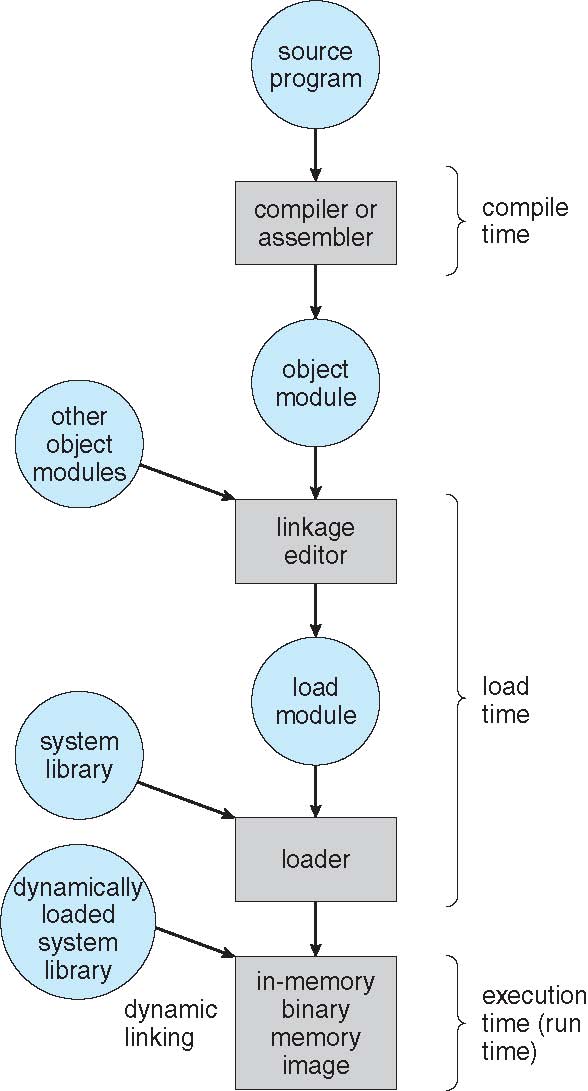
* **Multistep Processing of a User Program**



* **Binding of Instruction and Data to Memory**
  + Compile time

프로세스가 메모리 내에 들어갈 위치를 컴파일 시간에 미리 알 수 있으면 컴파일러는 절대 코드를 생성할 수 있다. 시작 주소가 변경되어야 한다면, 이 코드는 다시 컴파일되어야 한다.

* + Load time

프로세스가 메모리 내의 어디로 올라오게 될지를 컴파일 시점에 알지 못하면 컴파일러는 일단 이진 코드를 재배치 가능 코드(relocatable code)로 만들어야 한다.

* + Execution time

프로세스가 실행하는 중간에 메모리 내의 한 세그먼트로부터 다른 세그먼트로 옮겨질 수 있다. 특별한 하드웨어가 필요하다.

* **Input Queue**

디스크에서 주 메모리로 들어오기를 기다리고 있는 프로세스들의 집합.

* **Logical Address**

CPU가 생성하는 주소.

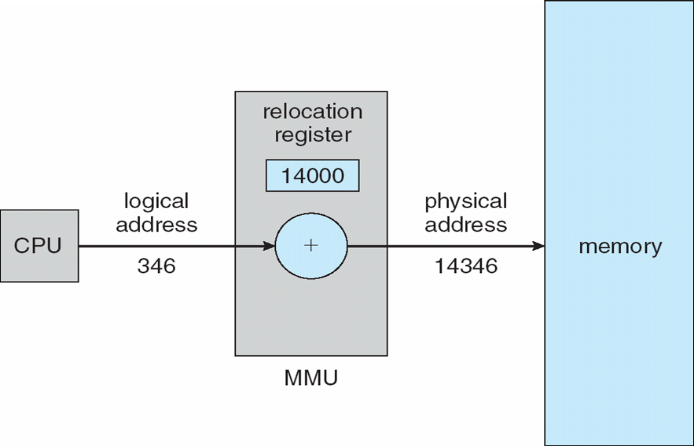
* **Physical Address**

메모리가 취급하는 주소.

* **Memory Management Unit (MMU)**

가상 주소를 물리 주소로 변환 작업을 하는 하드웨어 장치.

* **Relocation Register**



* **Ready Queue**

실행할 준비가 된 프로세스들의 집합.

* **Swapping**

자주 접근하지 않는 메모리는 일시적으로 보조 기억 장치(Backing Store)에 저장된다. 해당 메모리가 필요하면 가져오고, 자주 접근하지 않는 메모리를 저장하는 방법이다.

* **Context-Switch Time Including Swapping**

100MB process swapping to hard disk with transfer rate of 50MB/sec. Swap out time of 2sec. Swap in same sized process. Assume that disk head delay 8ms

head delay + Swap out + head delay + Swap in = (100MB / 50MB/sec + 8ms) \* 2 = 4016ms

* **Dynamic Storage-Allocation Problem**
  + First-Fit

사용 가능한 첫번째 자유 공간에 할당.

* + Best-Fit

사용 가능한 자유 공간 중 가장 작은 자유 공간에 할당.

* + Worst-Fit

사용 가능한 가장 큰 자유 공간에 할당.

* **External Fragmentation**

공간을 모두 합하면 사용가능하지만, 작은 공간으로 분산되어 있어 사용불가능한 공간.

* **50% Rule**

First-Fit의 통계적인 분석에 따라, 메모리의 1/3이 외부 단편화 문제로 사용 불가능한 현상.

* **Internal Fragmentation**

메모리가 고정된 크기로 분할되어 있고, 요청되는 메모리를 분할된 크기의 정수 배로만 할당해주어, 할당된 공간이 요구된 공간보다 더 클 경우 이들 둘 사이에 남는 공간.

* **Compaction**

외부 단편화를 해결하는 방법으로, 메모리를 한 곳으로 몰아 모든 자유 공간을 큰 블록을 만드는 방법.

* **Paging**

가상 주소 공간을 모두 같은 크기의 블록으로 편성하여 관리하는 방법. 논리 주소 공간이 한 연속적인 공간에 다 모여 있어야 한다는 제약을 없앤다. 즉, 외부 단편화가 발생되지 않는다.

* **Translation Look-aside Buffer (TLB)**

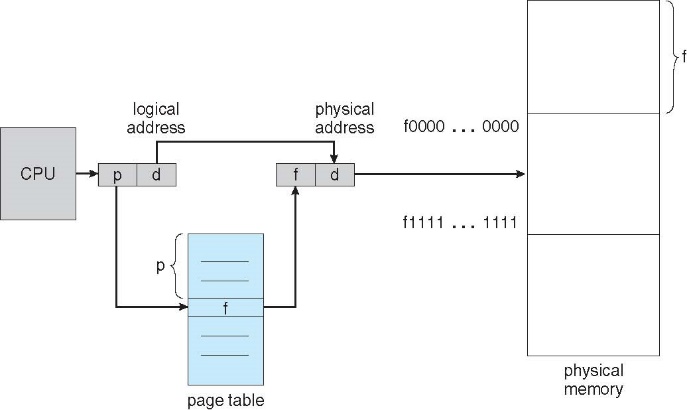
페이지 테이블 접근 시간을 절약하기 위한 소형 하드웨어 캐시.

* **Effective Memory Access Time**

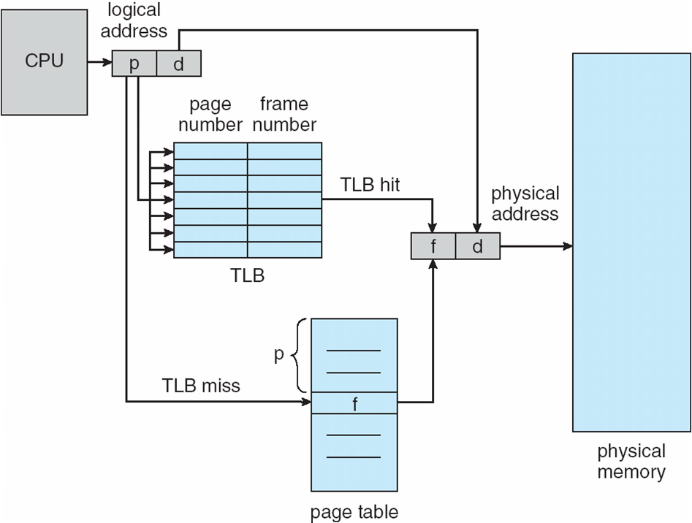
Hit ratio 80%, TLB search time 20ns, Memory Access time 100ns

EMAT = 0.8 \* (20+100) + (1–0.8) \* (20+100+100)

* **Paging Hardware**



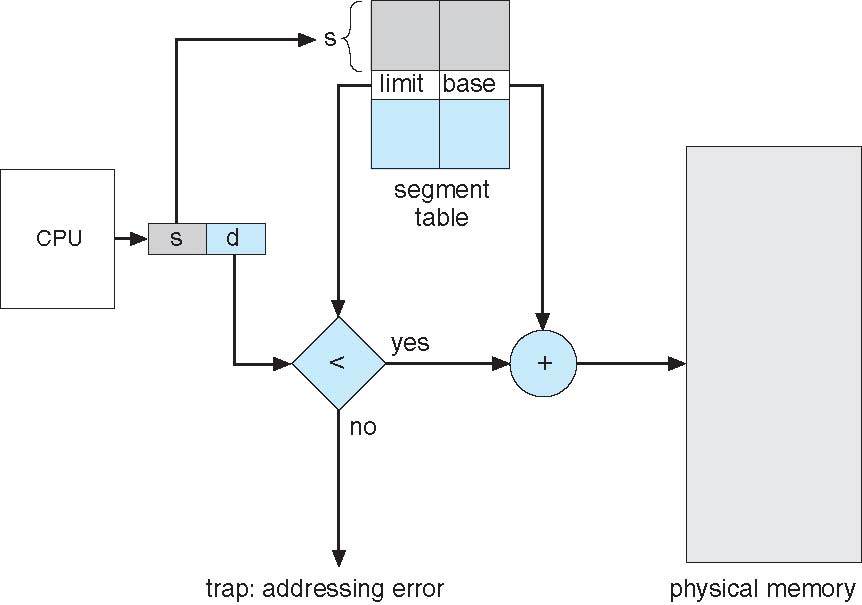
* **Paging Hardware With TLB**



* **Segmentation**

사용자의 메모리 관점을 그대로 지원하는 메모리 관리 기법.

* **Segmentation Hardware**

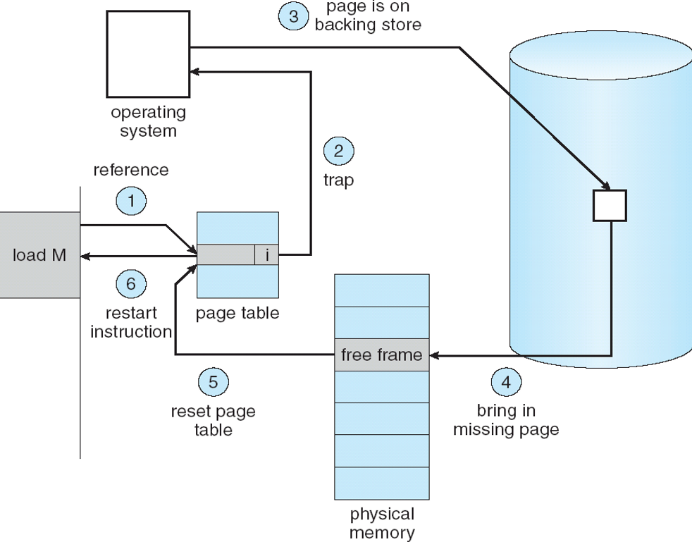


* **Effective Access Time (EAT)**

Page fault rate 0.01, Memory access time 300ns, Average fault service time 5ms

EAT = 0.01 \* 300 + (1-0.01) \* 5000

* **Processing of Page Fault**



* **FIFO Page Replacement**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| 7 | 7 | 7 | 2 |  | 2 | 2 | 4 | 4 | 4 | 0 |  |  | 0 | 0 |  |  | 7 | 7 | 7 |
|  | 0 | 0 | 0 |  | 3 | 3 | 3 | 2 | 2 | 2 |  |  | 1 | 1 |  |  | 1 | 0 | 0 |
|  | 1 | 1 | 1 |  | 1 | 0 | 0 | 0 | 3 | 3 |  |  | 3 | 2 |  |  | 2 | 2 | 1 |

* **Belady’s Anomaly**

프로세스에게 프레임을 더 주었지만, 오히려 페이지 부재율이 증가하는 현상.

* **Optimal Page Replacement**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| 7 | 7 | 7 | 2 |  | 2 |  | 2 |  |  | 2 |  |  | 2 |  |  |  | 7 |  |  |
|  | 0 | 0 | 0 |  | 0 |  | 4 |  |  | 0 |  |  | 0 |  |  |  | 0 |  |  |
|  |  | 1 | 1 |  | 3 |  | 3 |  |  | 3 |  |  | 1 |  |  |  | 1 |  |  |

* **Least Recently Used (LRU) Algorithm**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| 7 | 7 | 7 | 2 |  | 2 |  | 4 | 4 | 4 | 0 |  |  | 1 |  | 1 |  | 1 |  |  |
|  | 0 | 0 | 0 |  | 0 |  | 0 | 0 | 3 | 3 |  |  | 3 |  | 0 |  | 0 |  |  |
|  |  | 1 | 1 |  | 3 |  | 3 | 2 | 2 | 2 |  |  | 2 |  | 2 |  | 7 |  |  |

* **Fixed Allocation**
  + **Equal Allocation**

모든 프로세스에 같은 수의 프레임을 할당.

* + **Proportional Allocation**

프로세스 크기에 따라 프레임을 할당.

* **Global Allocation**

경쟁 프레임의 대상을 다른 프로세스에 속한 프레임을 포함한 모든 프레임을 대상으로 하는 방법.

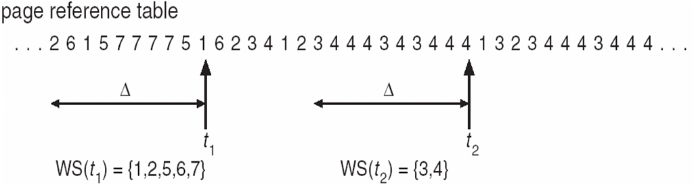
* **Local Allocation**

경쟁 프레임의 대상을 자신에게 할당된 프레임들만 대상으로 하는 방법.

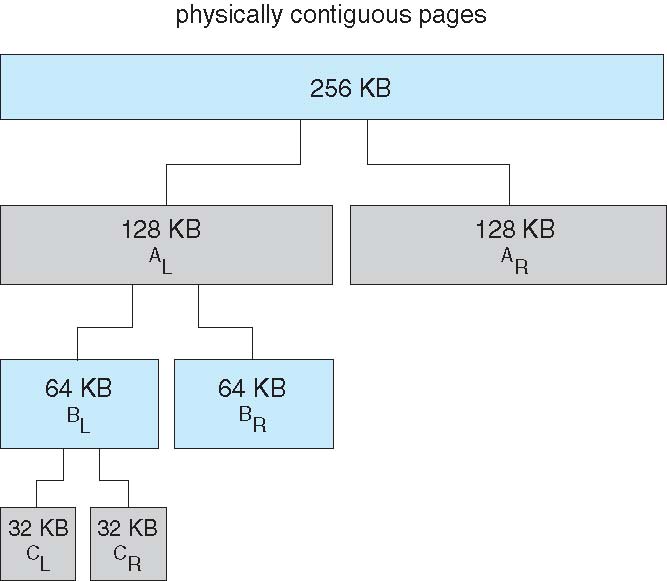
* **Thrashing**

어떤 프로세스가 실제 실행시간보다 더 많은 시간을 Paging에 사용하고 있을 경우, 이런 과도한 Paging 작업을 Thrashing이라 한다.

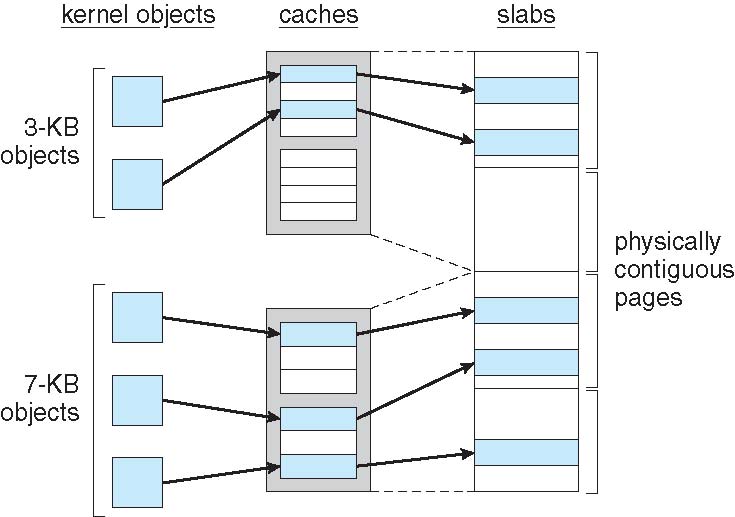
* **Working-Set Model**



* **Buddy System Allocator**



* **Slab Allocator**



* **Prepaging**

관련된 모든 페이지를 사전에 한꺼번에 메모리로 가져오는 기법.

* **Page Size**

메모리 사용효율이나 내부 단편화를 최소화 하기 위해서는 작은 페이지 크기가 좋다.

디스크 I/O시간의 효율이나 페이지 부재 횟수를 줄이기 위해서는 큰 페이지가 좋다.

* **File Attributes**

Name / Identifier / Type / Location / Size / Protection / Time, date and user identification

* **File Operations**

Create / Write / Read / Reposition within file / Delete / Truncate / Open / Close

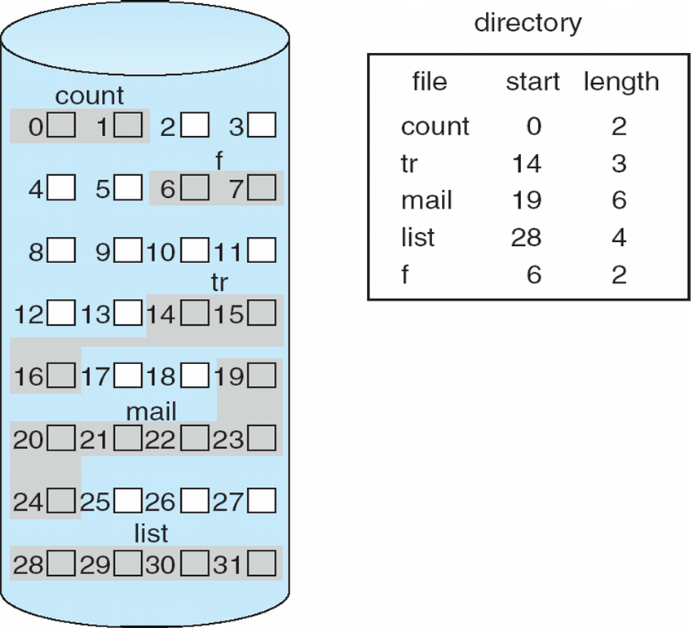
* **Operations Performed on Directory**

Search / Create / Delete / List / Rename / Traverse

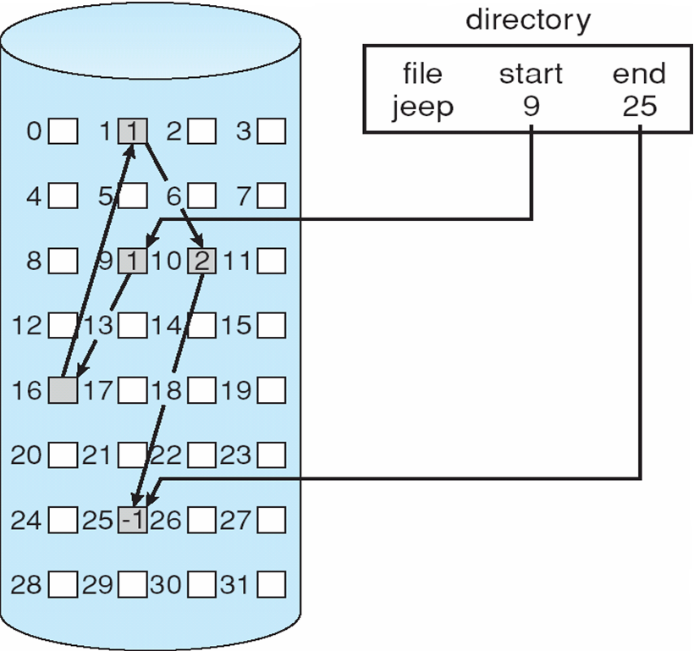
* **Layered File System**

Application Programs  
Logical file System  
Basic file System  
I/O Control  
Devices

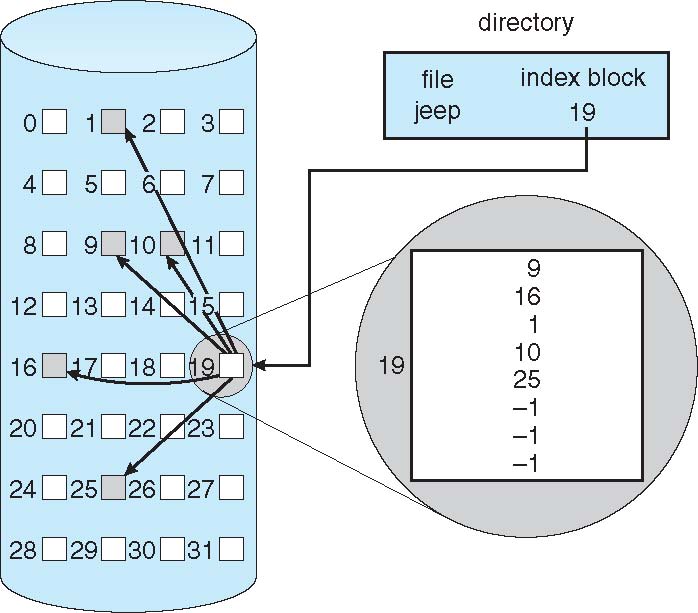
* **Contiguous Allocation**

****

* **Linked Allocation**



* **Indexed Allocation**



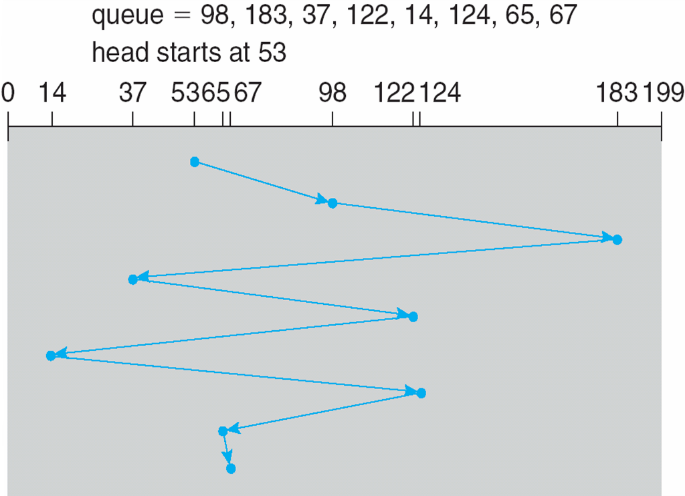
* **File Allocation Table (FAT)**

FAT는 각 디스크 블록마다 한 개의 항목을 가지고 있고, 이 하옥은 디스크 블록 번호를 색인으로 찾는다. 디렉토리의 항목은 각 파일의 첫 번째 블록 번호를 가리킨다. 그 번호로 FAT를 참조하면 다음 블록의 번호를 알 수 있다.

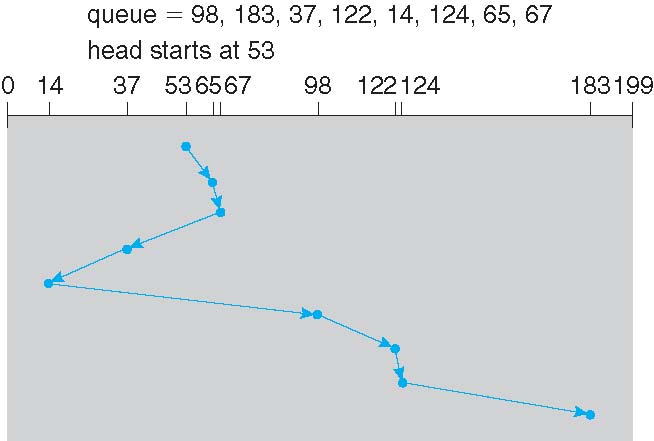
* **Difference Between UNIX FS and Andrew FS on Consistency Semantic**

열린 파일에 대하여 UNIX FS는 한 사용자의 파일 변경이, 동일 파일을 연 다른 사용자에게 즉시 보일 수 있지만, Andrew에서는 변경된 후에 시작되는 세션에만 보인다.

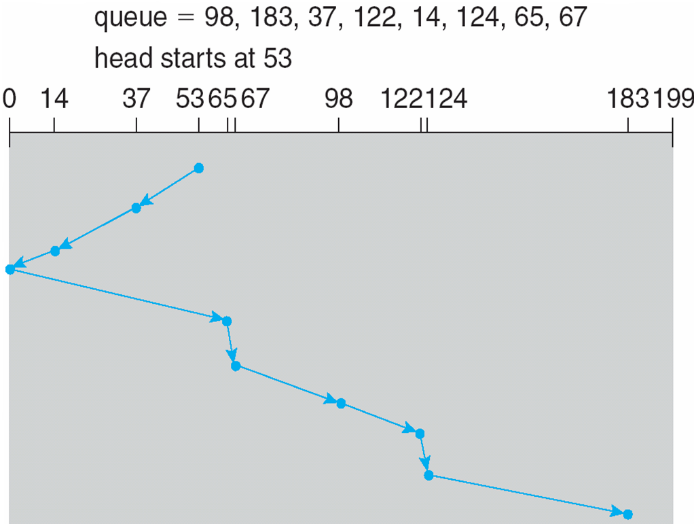
* **First Come First Served (FCFS) Scheduling**



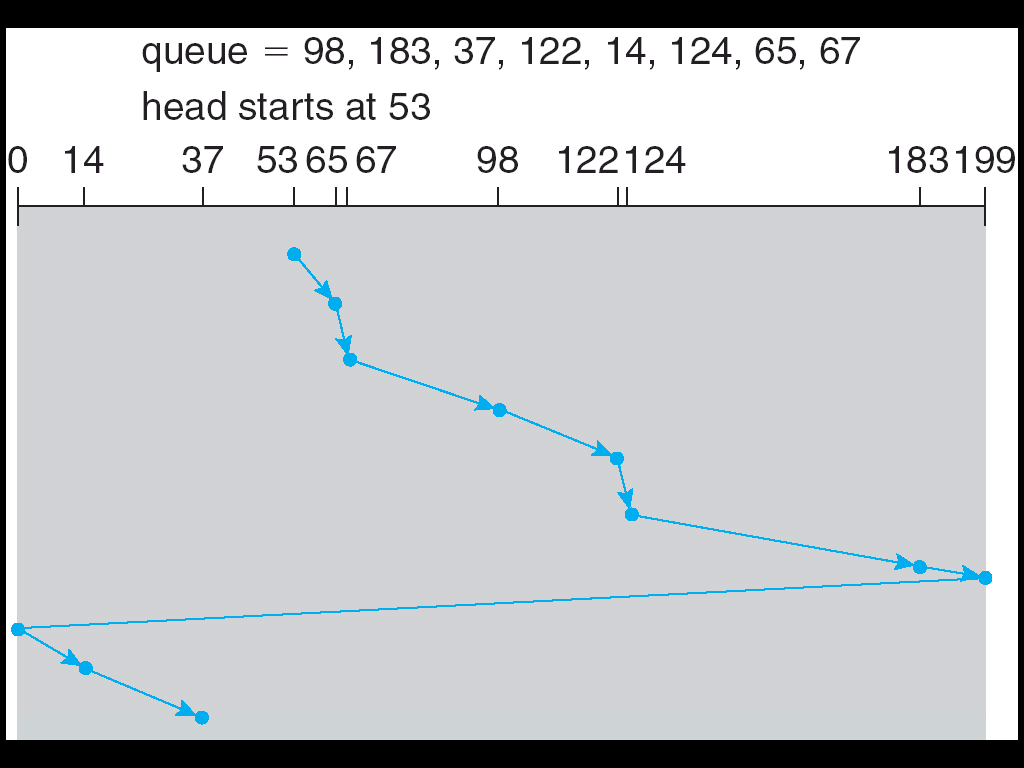
* **Shortest Seek Time First (SSTF) Scheduling**

****

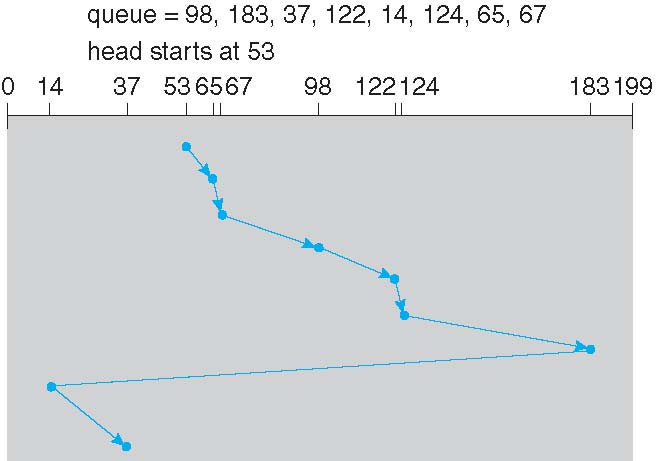
* **SCAN**

****

* **C-SCAN**



* **C-LOOK**

****

* **RAID**

**10_11.pdf**

* **Access Matrix**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | F1 | F2 | F3 | printer |
| D1 | r |  | r |  |
| D2 |  |  |  | print |
| D3 |  | r | e |  |
| D4 | rw |  | rw |  |

어떤 도메인에서 어떤 오브젝트에 대해서 어떤 권한을 가졌는지 바로 알 수 있다.

* **What Policy, How Mechanism**
* **Implementation of Access Matrix**
  + **Global table**

<domain, object, rights-set>

간단하지만, 테이블이 크다. 오브젝트를 그룹화하기 힘들다.

* + **Access lists for objects (column)**

<domain, rights-set>

각각의 오브젝트가 access list를 가진다. 사용자에 따라서 관리된다. 도메인에 대해서 관리하기 어렵다.

회수는 간단하게, access list를 탐색하고 지운다. 즉각적이고, 범용적으로 또는 선택적으로 전부를 또는 일부를 영구적으로 또는 일시적으로 회수가능하다.

* + **Capability list for domains (row)**

권한을 가진 오브젝트들의 리스트다. 오브젝트는 capability라고 하는 이름 또는 주소로 대표된다.

* + - Reacquisition

주기적으로 삭제.

* + - Back-pointers (Multics)
    - Indirection
    - Keys
  + **Lock-Key**

각각의 오브젝트와 도메인은 Lock과 Key이라는 고유한 비트 패턴의 리스트를 가지고 있다. Lock과 Key가 매칭이 될 때 권한이 승인된다.