1. 운영체제 기초지식
    1) 레지스터
        (1) cpu에 속해있음.
        (2) 32, 64비트가 있음.
        (3) 레지스터는 임시적으로 데이터를 저장하는 굉장히 빠른 저장장치.
        (4) 레지스터의 크기 = 표현할 수 있는 크기
    2) ALU(산술연산장치)
        (1) 레지스터의 데이터들이 ALU에 들어가서 계산이 됨.
    3) 단위의 기초
        (1) 1바이트 = 8비트
        (2) 모든 데이터를 비트로 표현
        (3) 숫자 0, 1은 1비트로 표현 가능.
        (4) 2^x (x는 비트)만큼의 개수를 표현 가능.
    4) cashe
        (1) 저장장치는 HDD, RAM, CASHE, REGISTER가 있음, 뒤로 갈수록 빠름.
        (2) cashe에는 L1, L2, L3가 있음
            L1 일수록 더 빠르고 비쌈. (CPU에 붙어있는 i3, i5, i7<- 차이 점은 CPU 클럭속도, cashe사이즈.(클럭이란, CPU같은 전기회로에서 보내는 신호, 0,1등을 보냄.)

2. 운영체제 유형
    1) 시스템 구성
        (1) 하드웨어
        (2) 운영체제
        (3) 응용프로그램
        (4) 유저
    2) 일괄 처리 시스템
        (1) 유휴 상태의 시간을 없애기 위하여 작업 순서의 자동화 개념.
            (즉, 다음 할 일을 대신 해줌) (이점 모든 자원을 프로그램 하나가 독식 가능.)

        (2) 상주모니터를 수행 (일 처리 상황 확인 가능)
        (3) 작업의 준비 및 실행 순서를 자동화함으로써 시스템의 성능 증진

    3) 다중 프로그래밍 시스템
        (1) CPU가 항상 수행되도록 하는 개념.

(2) 주기억장치에 여러 프로그램들이 존재하도록 함 (램에 여러 프로그램 올려놓고 한꺼번에 돌리는 개념) 모르겠으면, 운영체제 소개 PPT 그림 참조. (I/O상황이 발생하거나 해서 유휴시간이 어쩔 수 없이 발생하면 곤란하기 때문에, 만들어진 것 같음)(단점으론 독식 불가능.)

4) 시분할 시스템
   (1) 여러 사용자들이 컴퓨터 자원에 대한 짧은 시간단위 공유.(이해가 안가면 PPT 보자. 주로 프린트에 많이 사용되는 것 같음)

5) 실시간 시스템
   (1) 엄격하게 데드라인을 지켜야하는 상황일 경우 사용.
   (2) 사전에 정의된 제약 내에서 사용.
   (3) 의료기기 등에서 사용.

6) 다중 처리 시스템
   (1) 밀착된 결합 시스템 혹은 강결합 시스템이라고도 부름.
   (2) 병렬시스템이라고도 함.(별로 중요한 것 같진 않음.)
   (3) 프로세서(코어)가 여러개.
   (4) 코어들끼리 기억장치, 클럭을 공유 -> 밀착된 결합.
   (5) 느슨한 결합도 있음.
   (6) N개의 프로세서를 사용한다고 해서 N배가 되는 건 아님, (N개의 프로세서가 하나의 일을 분담해서 할 수도 있기 때문.)

7) 개인용 컴퓨터 시스템
   (1) 주변장치의 응답성을 중시.

8) 분산처리시스템.(뭔가 중요해보인다잉)
   (1) 느슨한 결합시스텝이라고도 함.
   (2) 프로세서들이 기억장치와 클럭을 공유하지 않음.
   (3) 고속 버스나, 전화선 같은 다양한 통신 라인을 통해 서로 통신.
   (4) 자원을 가지고 있는 사이트는 서버가 되며, 반면 다른 사이트에서의 클라이언트나 사용자는 그 자원을 사용. (자원이란, 프린트드라이버 같은 소프트웨어를 말함.)(예 클라우드)
   (5) 두 가지 기법이 있음.
      ① 네트워크 운영체제
        네트워크 운영체제는 노드 간 기종차이가 심하고 대규모 네트워크 시스템에 사용.
        각 노드들은 독자적인 운영체제를 가짐.(클라우드)
      ② 분산 운영체제
        각 노드들은 하나의 운영체제로 운영. (프린트)
   (6) 장점

① 자원 공유(소프트웨어를 말하는 듯.)
② 연산 속도 향상
③ 신뢰성 향상
④ 통신 가능

9) 멀티미디어 시스템
(1) 다양한 미디어를 이용하여 멀티미디어 콘텐츠를 제작하기 위해 필요한 하드웨어와 소프트웨어로 구성.
(2) 콘텐츠 제작을 위한 저작도구가 필요. ppt그림 참조.
① (참고 : CODEC : CODE - DECODER, 디지털 신호 -> 아날로그 신호. 이런 작업이 멀티미디어 시스템에서 굉장히 중요.)

10) 임베디드 시스템
(1) 마이크로프로세서 또는 마이크로컨트롤러를 내장하여 시스템 제작자가 의도한 몇 가지 기능 혹은 특수한 기능만을 수행하도록 제작된 시스템.
① (참고 - 작은 부품에 들어가는 시스템, 최적화 되어 몇가지 기능들을 위한 시스템, 예로 스마트 워치 등등.)
(2) 강점
① 한정된 자원에 맞게 최적화.

3. 운영체제에 대한 관점
1) 자원관리자 관점

(1) 각 자원에 대한 수행과정
① 자원의 상태를 추적 저장
② 어떤 프로세스가 언제 어떤 자원을 얼마나 사용할 것인지를 결정하기 위한 방법 수립.
③ 자원의 할당
④ 자원의 회수 ( 프로그램 코딩 중 블루스크린이 뜨는 이유가 회수가 안되어서...)

(2) 프로세스 관리 기능

(3) 중앙처리장치와 프로세스의 상태를 추적 및 저장 = 트래픽 컨트롤러라고 부름.

(4) 프로세스 스케줄러
① 다음 실행 될 프로세스들의 스케줄을 정리해 주는 역할.
(5) 기억장치 관리 기능
① 기억장치의 상태를 추적 저장. 주로 main memory

(6) 장치 관리 기능

① 채널 등의 제어장치 및 입출력장치와 같은 각종 장치의 상태를 추적, 저장.(채널이란 - 버퍼를 할 때 자주 쓰인다고 함, 간단하게 쓰레드가 아닐까? *시간날 때 찾아볼 필요성이 있음.)

② 입출력 트래픽 컨트롤러 - (입출력에 관한 프로세스 상태를 추적 및 저장.)

③ 입출력 스케줄링 - 장치를 할당하는데 어떤 방법이 효율적인지 결정. 만약 장치가 공유되는 것이라면 어떤 프로세스가 이 자원을 얼마나 사용할지를 결정.(큐에 저장해주는 스케줄링이라고 생각하면 편함.)

(7) 정보관리기능

① 정보의 위치, 사용 여부 및 상태 등을 추적, 관리

② 파일시스템이라고도 함.

2) 프로세스 관점

(1) 하나의 작업이 제시되어 완료될 때까지 하나의 프로세스에 대하여 그 상태를 변환시키고 관리 - (다중 프로그래밍 이상의 운영체제에서 봤을 때)

3) 계층 구조 관점

(1) 자원 관리 루틴이 어떻게 수행되고, 이 루틴이 상호간에 어디에 위치하는가,

(2) 모듈화된 자원관리.

(3) 레벨에 따른 분류 (레벨이 낮을수록 중요하다요)

① 레벨 1 - 프로세서 관리 하위 모듈(프로세스 스케줄러)

② 레벨 2 - 기억장치 관리(메모리 할당 및 회수)

③ 레벨 3 - 프로세서 관리 상위 모듈(메세지, 프로세스의 생성 및 제거)

④ 레벨 4 - 장치 관리(입출력 트래픽 제어기)

⑤ 레벨 5 - 정보 관리(파일시스템)

4. 입출력 프로그래밍

1) 중앙처리장치와 비동기적으로 수행 (동기적이면 중앙처리장치 아무것도 못해요 프린트 끝날 때 까지.) (참고로 동기화라는건 단위 시간마다(예 5초) 수행을 맞추는 것)

2) 중앙처리장치와 입출력장치에 대한 작동이 독립적이며, 수행상의 시간관계 규정이 없이 병행적으로 수행.

3) BIOS (basic input/output system)

(1) ROM에서 얻어낸 정보로 실행.(ROM = 읽기만 하는 전용 저장장치)

(2) 부트 프로세스를 실행시키는 명령어들을 포함.
(3) 컴퓨터 칩에 저장된 명령어들을 펌웨어라고 함. (칩에 있는 명령어를 바로 실행 시킬 수 있는 것 = 확장성 펌웨어 인터페이스, 내생각엔 이거 펌웨어 같은데 ?)
(4) BIOS를 뛰어넘는 확장성 펌웨어 인터페이스가 개발됨,.

4) 부트 진행 과정(외울필요는 없다요)
(1) 컴퓨터 가동을 위해 파워버튼을 누르면 부터로더가 준비동작에 들어감.
(2) POST(Power-On Self-Test)수행 시작 - 성공적인 부팅 및 적정 수행의 확인을 위해 필요하드웨어에 대해 행하는 테스트
① 초기 BIOS의 완벽한 보전 확인
② 주기억장치의 할당, 확인 및 그 크기의 결정
③ 시스템 버스(system buses, computer bus, cpu가 처리한 데이터들은 모니터에 출력되거나 메모리에 저장되어 진다. 이러한 행위가 이루어지기 위해서는 위의 데이터들이 각 각 컴포넌트끼리 통신이 가능해야 한다, 컴포넌트란, 통신규정이라고 생각하면 됨.).와 시스템디바이스 할당 및 시작
④ 다른 BIOS들의 시작 허용(비디오 또는 그래픽카드 등)
⑤ 사용자에게 BIOS시스템 구성 페이지에 접근할 수 있는 권한 부여
⑥ 부트 디바이스 할당 및 부트 파일을 가진 디바이스 찾기
⑦ 운영체제에 의해 요구되는 그 외 준비관련 태스크들의 마무리 작업 수행.
(3) 유저가 원할 경우 BIOS 세팅들에 접근을 위해 즉시 '키'조작 시도함.
(4) 잠깐 동안의 메모리 테스트가 수행되고 여러 파라메타들이 세트됨.
(5) 플러그 & 플레이 디바이스들이 준비동작에 들어감
(6) DMA(Direct Memory Access)채널을 위한 자원들과 IRQ(Interrupt Request)할당됨.
(7) 부트 디바이스들이 정해지고 준비동작에 들어감.
(8) OS가 준비동작에 들어감.

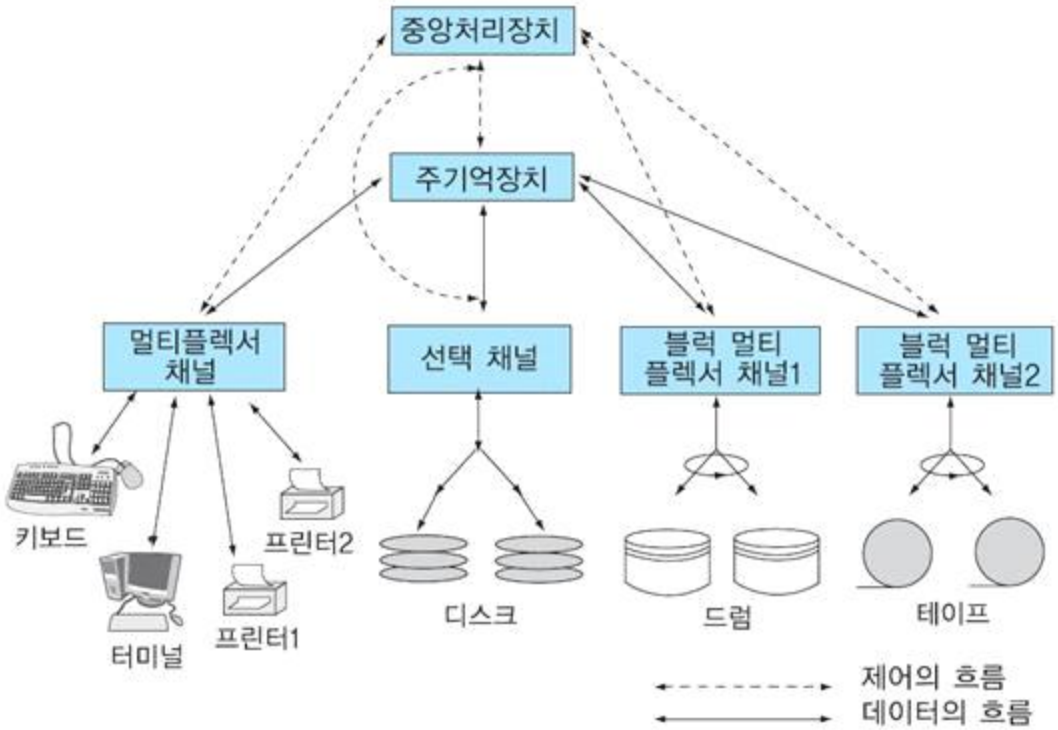
5) 버퍼링
(1) 입출력장치의 느린 속도를 보완하는 한 가지 방법, 중앙처리장치와 입출력장치 간의 시간적 불균형을 극복, 미리 읽혀진 레코드들이 존재하는 곳은 주기억장치의 일부.
(2) 방법
① 중앙처리장치에서 미리 읽어 주기억장치에 저장, ->채널이 저장된 것을 읽고 -> 프린트(이 사이에 주기억장치는 다른 일 가능)
(3) 이중버퍼링의 예
① 중앙처리장치가 버퍼A를 채운다.
② 다 채우면, 채널에게 버퍼A의 출력을 지시.

③ 버퍼A에 대한 출력과는 무관하게 버퍼B를 채운다.
④ 채널이 버퍼A를 다 비우게 되면, 중앙처리장치에게 알린다.
⑤ 중앙처리장치는 다시 버퍼A를 채우고 채널은 버퍼B를 비운다.(이 때, 1과 6, 3과 4는 동시 발생 가능.)

6) 스풀링.
(1) 다중프로그래밍 환경에서, 다수의 프로세서들이 입출력장치를 서로 요구하지만, 실제로 그 장치의 수가 제한되어 있을 때, 이들에 대한 공유를 가능하게 하기 위해 가상장치를 각 프로세스에 제공해주는 개념. (즉 디스크 = 프린터기로 가정하여 일단 정보를 디스크에 저장하고 실제 입출력은 천천히 진행되는 것 같음..)

7) 채널
(1) 출력장치와 중앙처리장치 사이에 입출력을 담당.
(2) 선택채널
① 여러 개의 입출력장치가 연결되어 있다 하더라도 한 번에 하나의 입출력장치만 선택적으로 지원
② 비교적 전송 속도가 빠른 입출력장치인 디스크나 CD-ROM등의 입출력을 제어
(3) 멀티플렉서 채널
① 다수의 저속도 입출력장치가 채널의 단일한 데이터 경로를 공유하면서 데이터를 전송.
② 여러 개의 저속도 입출력장치가 멀티플렉서 채널에 연결되어 시분할 형태로 제어. (1,2번을 종합해 보면, 즉 너무 느리니까, 여러 선로를 깔아놓고, 제어는 하나의 채널이 하는 거 아님 ?)
(4) 채널과 중앙처리장치는 일반적으로 인터럽트로 통신 (이말은 엄마가 아들에게 두부사와줘 ! 하는 것과 같은 문맥, 심부름은 인터럽트로 통신됨.)
(5) 블록 멀티플렉서 채널
① 멀티플렉서 채널의 장점과 선택 채널의 장점만을 골라 만든 채널로, 동시에 복수 개의 고속 입출력 장치를 공유하여 데이터를 받아들일 수 있는 채널. (선택채널은 고속입출력 장치 1개만을 관리, 멀티플렉서 채널은 저속입출력 장치 여러개를 관리, 블록 멀티플렉서 채널은 고속 입출력장치 여러개를 관리.)

8) 인터럽트
(1) 시스템에 예기치 않은 상황이 발생 하였을 때, 그것을 운영체제에 알리기 위한 메커니즘.
① 입출력 인터럽트(해당 입출력 하드웨어가 주어진 입출력 동작을 완료하였거나 입출력의 오류가 발생하였을 때 발생.)
② 외부 인터럽트(시스템 타이머에서 일정한 시간이 만료된 경우나 오퍼레이터가




콘솔 상의 인터럽트 키를 입력한 경우, 즉 응답없음의 상태.)

③ SVC 인터럽트(프로그램이 수행되는 과정에서 입출력 수행, 기억장치의 할당, 오퍼레이터의 개입을 요구할 때.(알림느낌?))

④ 기계 검사 인터럽트 (컴퓨터 자체 내의 기계적인 장애나 오류로 인한 인터럽트)

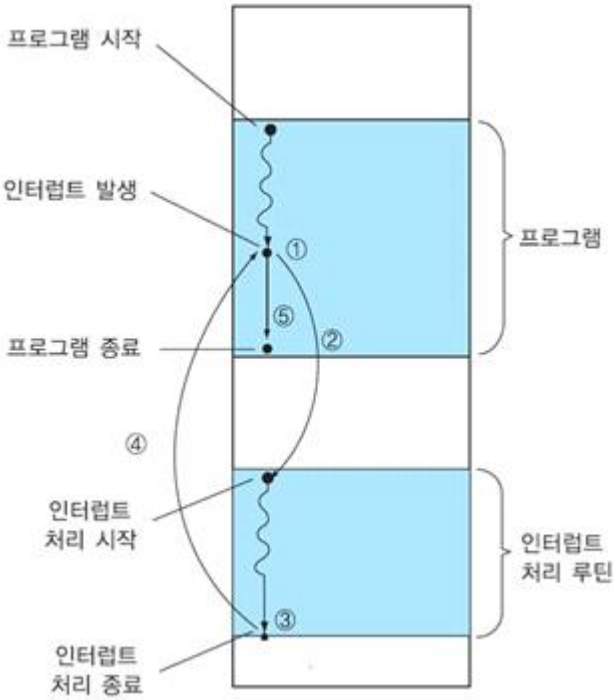
⑤ 프로그램 에러 인터럽트 (주로 프로그램 실행 오류로 인해 발생, 0으로 나누는 연산, 보호되어 있는 기억장소에 대한 접근, 허용되지 않는 명령어의 수행)

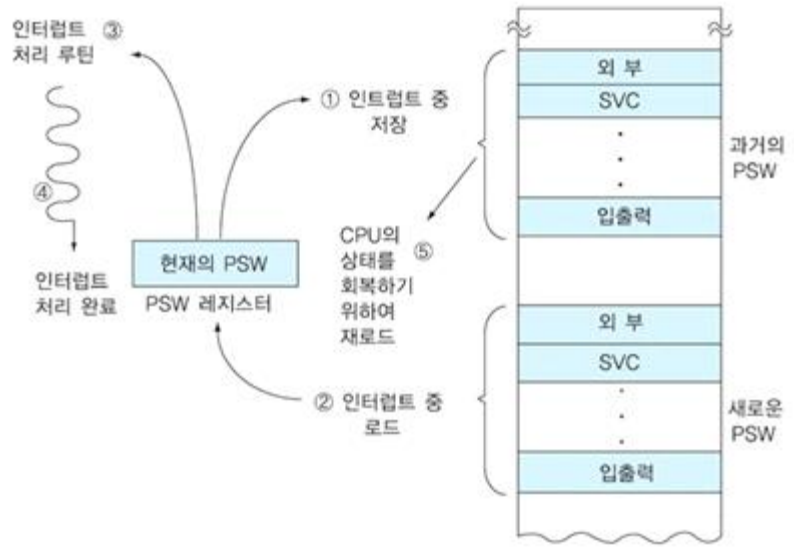
⑥ 재시작 인터럽트 (오퍼레이터가 콘솔상에 재시작 키를 누를 때 발생.)

(여기서부터 다시 프린트.)

5. 프로세스

1) CPU




(1) CPU는 컴퓨터 자원 중 가장 중요한 자원

2) 중앙처리장치 스케줄링 = 프로세스 스케줄링

(1) 준비 완료 상태에 있는 프로세스들 중 어느 것을 중앙처리장치에 할당시킬 것인가를 위한 정책.

(2) 중앙처리장치 효율 및 처리율의 최대화와 반환시간의 최소화

3) 프로세스 정의

(1) 실행 중인 프로그램

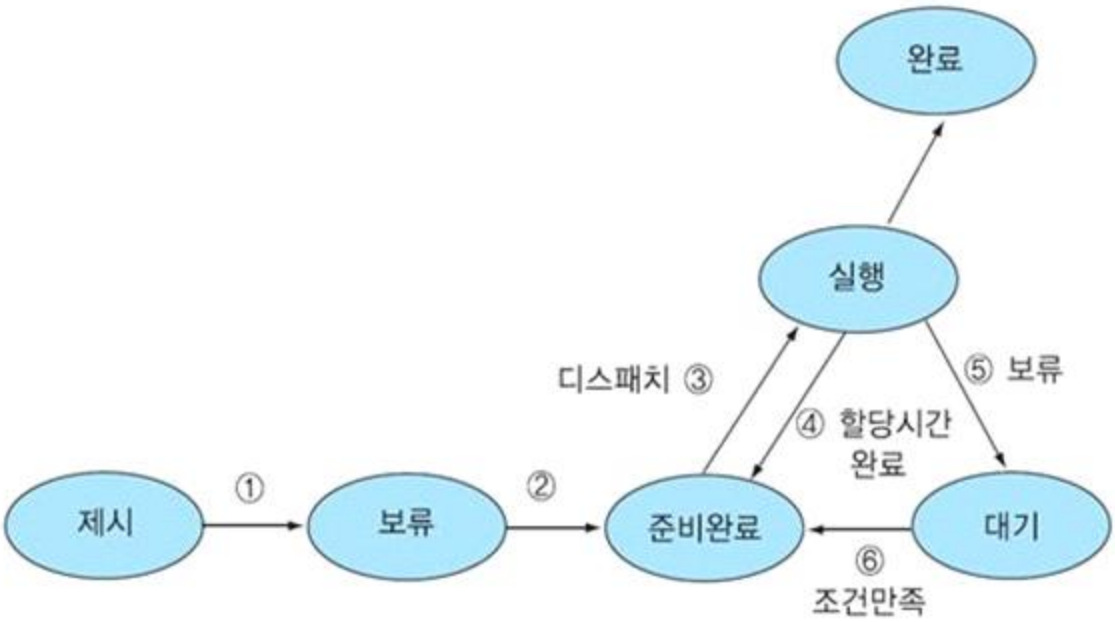
(2) PCP를 지닌 프로그램

(3) 프로그램 카운터를 지닌 프로그램

(4) 능동적 개체로 순차적으로 수행하는 프로그램

4) 프로세스 관리
   (1) 사용자 프로세스와 시스템 프로세스의 생성과 삭제 (시스템  프로세스는 뭐 os프로그램 같은거겠지 ?)
   (2) 프로세스의 일시 중지와 재 수행
   (3) 프로세스 스케줄링
   (4) 프로세스의 동기화
   (5) 프로세스 간 통신
   (6) 교착상태 처리
5) 프로세스 구성 요소
   (1) 임의의 프로그램이 실행되기 위해서는 반드시 실행되기 전에 주기억장치에 저장되어야 함. (기본이잖아 이건.)
   (2) 프로세스 구성요소 4가지
      ① 코드영역(프로그램 코드 자체, 프로그램은 실행되기 전에 주기억장치에 cpu가 해석할 수 있는 바이너리 코드 상태로 주기억장치에 올라가게 됨.)
      ② 데이터 영역(프로그램의 전역변수나 정적 변수의 할당.(전역변수 = 처음부터 메모리를 차지하고 들어가는 변수, 정적 변수 = 기본적으로 전역변수랑 같은 뜻을 내포하고 있으나, 접근할 수 있는 범위를 하나의 함수로 제한하고 있다. 즉 여러 클레스에서 맘대로 접근 불가느낌 ?)
      ③ 스택영역 (지역변수 할당과 함수 호출 시 전달되는 인수의 값.)
      ④ 힙 영역 (동적 할당)

6) 프로세스의 상태
   (1) 실행상태
      ① 프로세스가 중앙처리장치를 차지하고 있는 상태.
   (2) 준비완료 상태
      ① 중앙처리장치가 사용 가능하게 될 때 그것을 할당 받을 수 있는 상태
   (3) 대기 상태
      ① 프로세스가 중앙처리장치를 차지하고 처리하다가 입출력 처리 등을 하게 되면 중앙처리장치를 양도하고 입출력 처리가 완료될 때까지 대기하고 있는 상태.
   (4) 각 과정에 대한 변화과정 용어 설명
      ① 디스패치 : 준비완료 상태에서 실행 상태로 변환 되는 과정
      ② timer runout : 실행상태에서 준비완료 상태로 돌아가는 과정(할당시간 완료일 경우.)
      ③ block : 실행 상태에서 대기 상태로 가는 과정
      ④ wakeup : 대기 상태에서 준비완료 상태로 가는 과정.


7) 프로세스 제어 블록(PCB)

(1) 프로세스에 관한 모든 정보를 가지고 있는 데이터베이스. 모든 프로세스는 각기 고유의 PCB를 가짐.

(2) PCB내용

① 프로세스의 현재 상태(실행, 준비완료, 대기 등)

② 프로세스의 고유 이름(identifier)
③ 프로세스의 우선순위
④ 프로세스가 적제된 기억장치의 주소를 가지는 포인터
⑤ 할당된 자원을 가리키는 포인터


⑥ 중앙처리장치의 각종 레지스터 상태를 저장하기 위한 공간

8) 프로세스 스케줄링

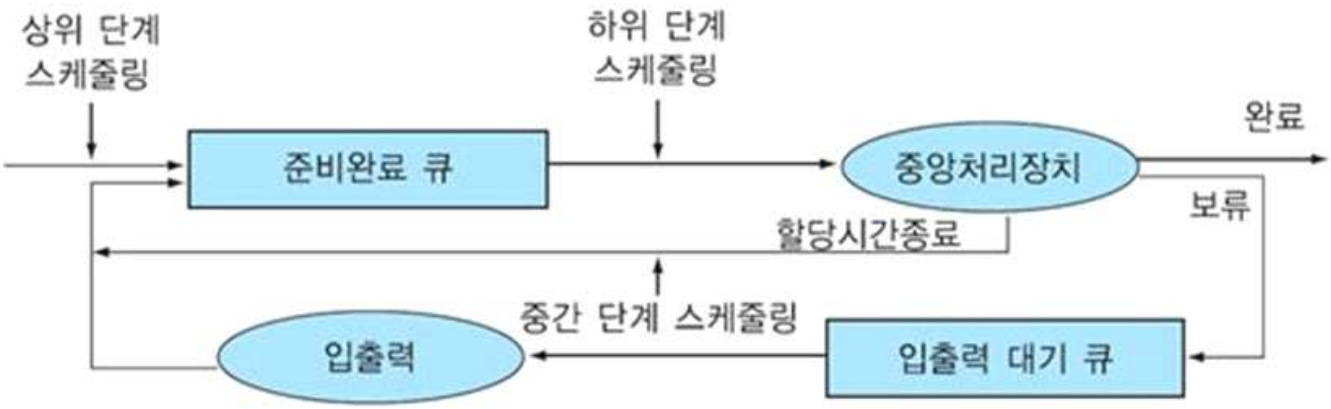
```
#include <unistd.h>
 main(){
    pid_t pid;
        printf("Before fork ... ₩n");
  /* fork( ) 명령어 실행 */
    pid=fork();
          if(pid == 0) /* 자식 프로세스 */
             printf("I'm the child ... pid is %d ₩n", pid);

          else if(pid> 0){ /*부모 프로세스 */
             wait(NULL);
             printf("I'm the parent... pid is %d ₩n", pid);
            }
          else
             printf("fork error");
       }
```

(1) 목적

① 공정성
② 응답 시간의 최소화
③ 예측가능
④ 오버헤드의 최소화
⑤ 자원 사용의 균형유지
⑥ 응답과 이용 간의 균형 유지
⑦ 실행의 무한한 지연을 피할 것
⑧ 우선순위제의 실시
⑨ 주요 자원들을 차지하고 있는 프로세스에게 우선권을 부여
⑩ 좀 더 바람직한 동작을 보이는 프로세스에게 더 좋은 서비스를 제공.
⑪ 과다한 부하를 감소

9) 기준 (여기서부터 다시 프린트.) (다시 돌아와야함.)
   (1) 입출력을 위한 프로세스인가
   (2) 연산 위주의 프로세스인가
   (3) 프로세스가 일괄처리 형인가 대화형인가
   (4) 긴급한 응답이 요구 되는가
   (5) 프로세스의 우선순위
   (6) 프로세스가 페이지 부재를 얼마나 자주 발생 시키는가
   (7) 높은 우선순위를 지니는 프로세스에 의해서 얼마나 자주 프로세슥 선점 되는가
   (8) 프로세스가 받은 실행 시간은 얼마나 되는가
   (9) 프로세스가 완전히 처리되는 데 필요한 시간은 얼마나 더 요구되는가.
10) 단계별 분류
   (1) 상위 단계 스케줄링
      ① 어떤 작업에게 시스템의 자운을 차지할 수 있도록 할 것인가 결정
   (2) 중간 단계 스케줄링
      ① 짧은 순간에 프로세스들에 대한 일시적 활동의 중단 및 재개를 수행.
   (3) 하위 단계 스케줄링
      ① 어떤 준비완료 프로세스에게 중앙처리장치를 할당할 것인가를 결정




11) 방법 및 환경별 분류.
   (1) 비선점 스케줄링
      ① 하나의 프로세스에 중앙처리장치가 할당되면 끝날 때까지 중앙처리장치를 계속 차지하는 것.
   (2) 선점 스케줄링
      ① 교대로 중앙처리장치를 차지하는 것. (시분할 시스템)
   (3) 우선순위 스케줄링
      ① 각 프로세스에게 우선순위를 부여하여 우선순위가 높은 순서대로 처리하는 방법.

② 정적 우선순위 기법 (상대적으로 오버헤드는 적으나, 주위 여건의 변화에 적응하지 않고 우선순위를 바꾸지 않는다.

③ 동적 우선순위 기법 (필요에 따라 우선순위 재구성

(4) 기한부 스케줄링 (실시간 시스템에서 주로 사용)

① 작업이 명시된 시간이나 기한 내에 완료되도록 계획

② 실시간 시스템에는 두가지 종류

경성실시간시스템 (정해진 시간 내에 완료할 수 있도록 해주는 강한형태의 실시간 시스템.) (시스템에 의해 실행될 테스크집합이 미리 정의되어 있는 경우, 주기적인 연성 실시간 태스크 집합에 유용 (아 이거 안외워도 될 것 같음.)

연성실시간시스템 (시간 제한이 다소 약한 형태의 실시간시스템) (동적 스케줄링 방식, 태스크의 발생 시간이나 특성을 미리 예측할 수 없을 경우에 유용 보장된 주기의 시간 내에 서비스를 보장받기 위해서는 경성실시간스케줄링을 사용해야 함.)

12) FCFS스케줄링

(1) 가장 간단한 스케줄링 방식

(2) 비 선점 스케줄링 방법

(3) 도착한 순서에 따라 중앙처리장치를 할당

(4) 호위 효과

① 첫 번째 프로세스가 끝날 때까지 매우 긴 시간을 기다리게 됨.

(5) 평균 대기시간 = 기다린 시간들을 합해 평균으로 나누는 것.

13) SJF스케줄링

(1) 비 선점 스케줄링 방법

① 프로세스 중에서 수행시간이 가장 짧은 것을 먼저 수행하는 비 선점 스케줄링 방식.

(2) 문제점

① 수행될 프로세스나 프로세스가 얼마나 긴 것인가를 정확히 알아야 하는데 이 정보를 얻기가 어려움.

(3) 평균 반환시간 = 즉 평균 프로세스 끝나는 시간. 한 개당 프로세스가 끝나는 시간을 다 더해서 프로세스 개수로 나눔.

14) 우선순위 스케줄링

(1) 무한 대기 또는 기아 현상.

① 낮은 우선순위의 프로세스들이 중앙처리장치를 무한히 대기하게 되는 경우

(2) 에이징

① 낮은 우선순위의 프로세스들의 무한대기 문제에 대한 해결책, 오랫동안 시스템에서 대기하는 프로세스들의 우선순위를 점진적으로 증가시키는 방법.

(3) 라운드 로빈 스케줄링

① 시분할 시스템을 위하여 고안된 선점 스케줄링 방식
② 각 프로세스는 같은 크기의 중앙처리장치 시간을 할당 받음.
③ 할당시간의 크기는 10에서 100ms사이 (할당시간이 너무 크면, FCFS방식과 같은 형식이 된다.) (할당시간이 너무 적으면 문맥 교환을 위한 오버헤드가 무시못할 요소가 되어 결과적으로 대부분의 시간이 중앙처리장치를 분배하는 데 소모.)
④ 작업 큐에 저장을 하고 CPU를 차지하는 시간을 할당하여 할당시간이 다 되면 다음 작업이 오게끔 함.

15) SRT스케줄링
(1) SJF 기법에서 선점 방식을 도입한 기법.
(2) 시분할 시스템에서 유용
(3) 새로 도착한 프로세스를 포함하여 처리가 완료되는데 가장 짧은 시간이 소요된다고 판단되는 프로세스를 먼저 수행. PPT참조

16) 다단계 큐 스케줄링(다단계 피드백 큐랑은 다르다잉)
(1) 작업들을 여러 그룹으로 나누어 여러개의 큐를 이용하는 기법
(2) 전면작업 프로세스들은 후면작업 프로세스들보다도 높은 우선순위
(3) 후면작업 큐가 선입선출 알고리즘에 의해 스케줄 되는 반면 전면작업 큐는 라운드로빈 알고리즘에 의해 스케줄
(4) 큐별로 스케줄링
(5) 전면작업 큐(라운드 로빈을 사용하는 우선순위가 높은 큐)에는 자신의 프로세스들 사이에서 라운드 로빈스케줄링을 위해 중앙처리장치 시간의 80%
(6) 후면작업(라운드 로빈을 사용하지 않는 우선순위가 낮은 큐) 큐는 자신의 프로세스들을 선입 선 처리 방식으로 중앙처리장치 시간의 20% (즉 우선순위가 낮으면 할당시간도 작아지는게 다단계 큐)
(7) 각 큐마다 이동이 불가능.

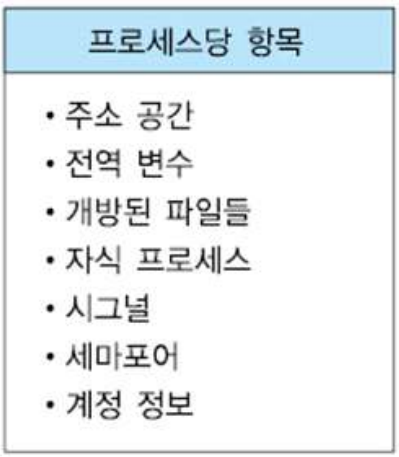
17) 다단계 피드백 큐 스케줄링
(1) 다단계 피드백 큐는 우선순위가 낮으면 할당시간이 커짐.
(2) 아래 단계 큐로 이동해 가면서, 할당시간이 점점 높아짐.
(3) 입출력 인터럽트같은 경우는 라운드 로빈 형식으로 재실행.
(4) 반드시는 아니지만 맨 마지막은 라운드로빈 형식.

18) HRN스케줄링
(1) SJF의 약점인 버스트 시간에 따른 프로세스 불평등을 보완해주는 스케줄링.
(2) 비선점 방식으로 하나의 작업이 CPU를 차지하면 그 작업은 완성될 때까지 실행.
(3) 대기시간이 고려되어 긴 작업과 짧은 작업간의 불평등을 없앰. (시스템 응답시간의 값이 커질수록 우선순위가 높아짐)
(4) 우선순위 = 시스템 응답시간 : (대기시간 + 서비스를 받을 시간) / (서비스를 받을

시간) = 시스템 응답시간.

6. 스레드 (가능하다면 나중에 따로 공부하는 걸 추천)
   1) 서로 독립적인 일들을 순차적으로 수행하는 것을 개선하기 위해, 프로세스보다 작고 독립적으로 스케줄링이 가능한 스레드라는 개념이 도입.
   2) 각 스레드는 서로 독립적
   3) 스레드의 실행 / 종료 순서는 예측 불가능.
   4) 스레드들은 수행을 위해 스케줄 되고 결과는 프로세스에게 전달
   5) 프로그램에 있는 스레드의 수는 다른 스레드에게 알려지지 않음
   6) 스레드는 프로그램의 외부에서 보이지 않음
   7) 스레드는 프로세스의 일부분이기 때문에 프로세스의 자원을 공유하지만 처리시간과 스택 레지스터는 별도할당
   8) 프로세스가 끝나면 모든 스레드들도 종료

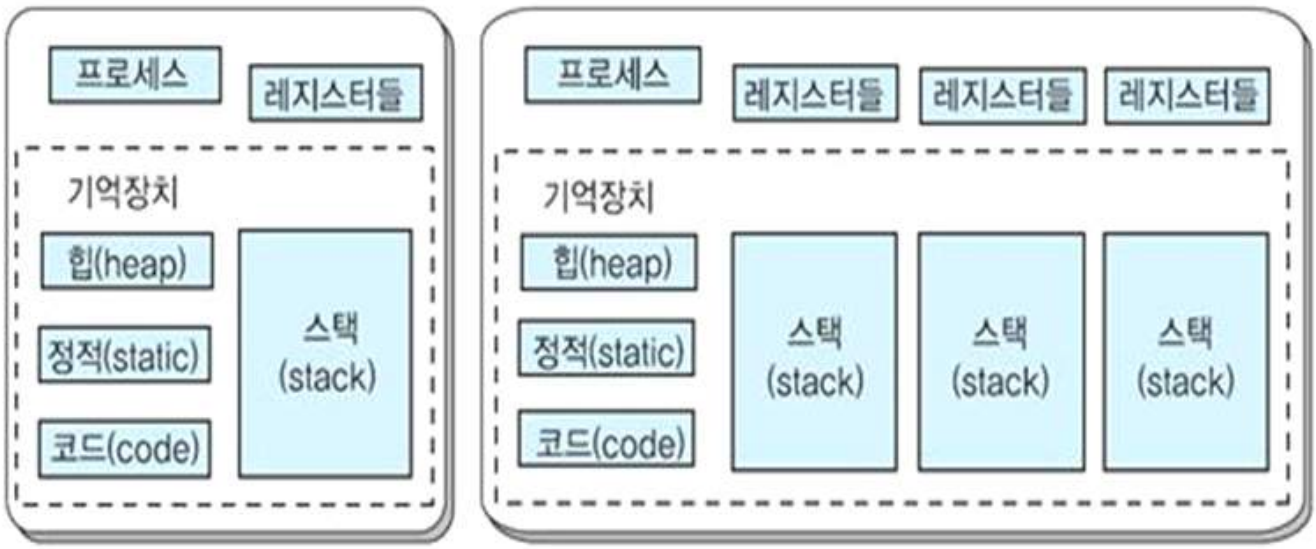

(a) 스레드당 항목




(b) 프로세스당 항목

   9) 프로세스와 다중스레드

      (1) 다중 스레딩
         ① 다수의 스레드를 이용하여 하나의 프로그램을 동시에 처리하는 것.
         ② 하나의 프로세스 자체에 다수의 실행 단위들이 존재하여 작업의 수행에 필요한 자원들을 공유하기 때문에 자원의 생성 및 관리가 중복되는 것을 최소화.
         ③ 중량 프로세스는 하나의 스레드를 가진 프로세스를 말함
         ④ 경량 프로세스는 프로세스 내에 두 개 이상의 스레드를 포함하고 있을 경우.
         ⑤ KLT방법 (스레드들이 커널에 의하여 지원)
         ⑥ ULT방법 (커널 상위 수준에서 지원)




(a) 단일 쓰레드형 프로세스 (b) 다중 쓰레드형 프로세스

7. 기억장치 관리

1) 프로그램과 데이터는 직접 실행되거나 참조되기 위하여 주기억장치에 있어야함

2) 주기억장치는 용량이 제한되어 있고 값이 비싸요..

밑에 그림은 참고만 하자

<table>
<tr><td></td><td colspan="2">실(real)기억장치</td><td colspan="3">실(real)기억장치</td><td colspan="3">가상(virtual)기억장치</td></tr>
<tr><td>단일 사용자<br>전용 시스템</td><td colspan="5">실기억장치<br>다중 프로그래밍</td><td colspan="3">가상메모리<br>다중 프로그래밍</td></tr>
<tr><td></td><td colspan="2">고정 분할<br>다중 프로그래밍</td><td rowspan="2">가변 분할<br>다중 프로그래밍</td><td rowspan="2">순수<br>페이징</td><td rowspan="2">순수<br>세그먼테이션</td><td rowspan="2">페이징/<br>세그먼테이션</td></tr>
<tr><td></td><td>절대</td><td>재배치<br>가능</td></tr>
</table>

3) 주소 바인딩 (나중에 나오거나 그냥 외우자 ..)

(1) 주소에는 논리적 주소와 물리적 주소가 있는데, 실제로 가져오려면, 논리적 주소에서 물리적 주소로 주소를 바꿔야함 이걸 주소바인딩이라고 함. Mapping한다고 하기도 함.

(2) 컴파일 시간 바인딩

① 실행 시 위치가 바뀌면 다시 컴파일 해야 함.

(3) 적재 시간 바인딩

① 적재될 위치를 컴파일러가 알지 못하면 컴파일러는 일단 이진코드를 재배치 가

능 코드로 생성.

② 주소 바인딩은 프로그램이 기억장치에 적재되는 시간에 적재기에 의해 이루어짐

(4) 실행 시간 바인딩

① 실행되는 동안에 기억장치의 한 세그먼트에서 다른 세그먼트로 옮겨질 경우.

4) 논리적 주소와 물리적 주소 그리고 기억장치 관리기.

(1) 논리적 주소

① 중앙처리장치가 생성하는 주소

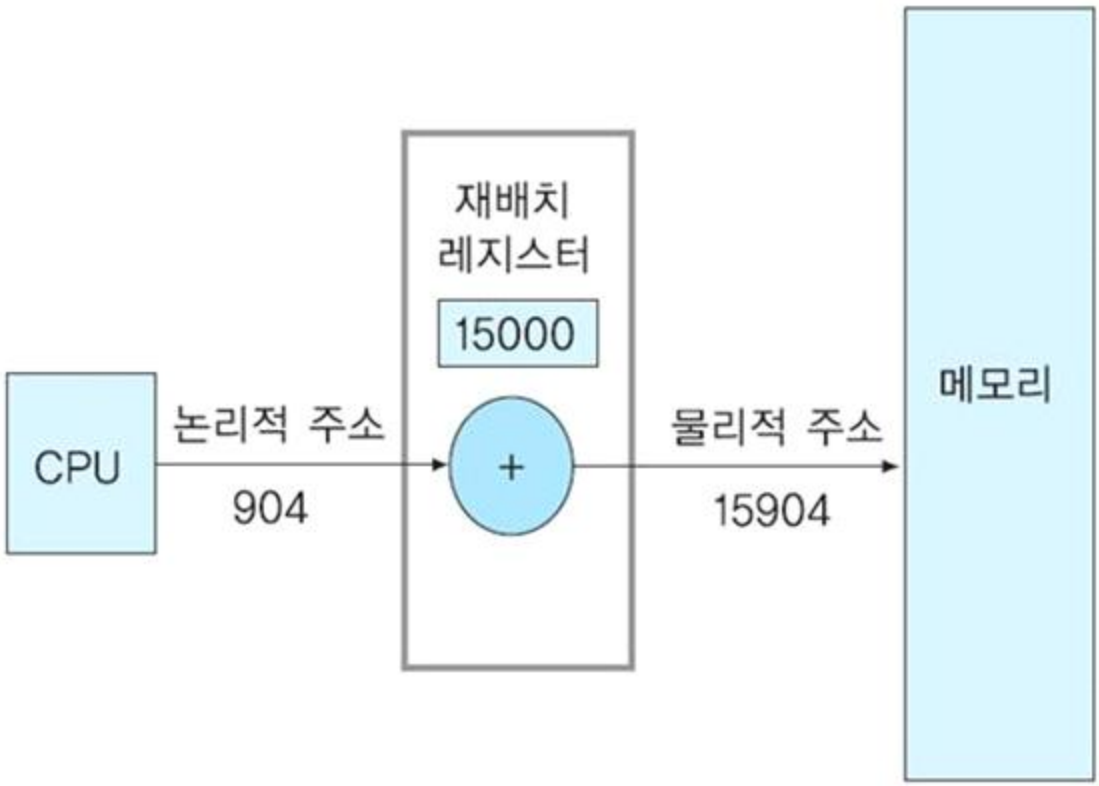
(2) 물리적 주소

① 메모리의 실제 주소 즉 기억장치가 취급하는 주소

(3) 기억장치 관리기

① 프로그램 실행 중에 논리적 주소를 물리적 주소로 변환하는 것.

아래 사진은 재배치 레지스터를 이용한 주소변환 사진.




5) 기억장치 관리 기법

(1) 인출 기법

① 주기억장치에 적재할 다음 프로그램이나 데이터를 언제 가져올 것인가를 결정

② 요구인출 기법

③ 예상 인출 기법

(2) 배치 기법

① 새로 인출된 데이터나 프로그램을 주기억장치의 어디에 위치시킬 것인가를 결정

하는 기법

② 최초 적합, 최적 적합 및 최악 적합

(3) 교체 기법

① 새로 들어온 프로그램이 들어갈 장소를 마련하기 위해 어떤 프로그램 및 어떤 데이터를 제거할 것인가를 결정

6) 단일 사용자 연속 기억장치 할당

(1) 초기 컴퓨터 시스템에서는 임의의 시간에 한 사용자만이 시스템 사용이 가능했음.

(2) 오버레이 기법

① 프로그램의 일 부분이 실행에 필요 없어지면 없애고 다른 프로그램을 보조기억장치로부터 옮겨오는 것.

(3) 시스템 보호

① OS같은 곳을 들어가면 안되기에... 경계 레지스터를 이용해 격리

(4) SVC(슈퍼바이저 호출) 명령

① 사용자가 운영체제 영역에 들어가려 할 때.

7) 고정 분할 기억장치 할당

(1) 효율적인 주기억장치 사용.

(2) 다중프로그래밍을 위해 여러 작업이 동시에 컴퓨터 내 주기억장치에 존재 가능

(3) 절대 번역 및 로딩 (사용자가 최대 기억장치 요구량을 명시)

(4) 재배치 가능 번역 및 로딩

(5) 시스템 보호 (여러 개의 경계 레지스터를 이용)

8) 가변 분할 기억장치 할당

(1) 작업들이 필요로 하는 만큼의 공간을 동적으로 할당.

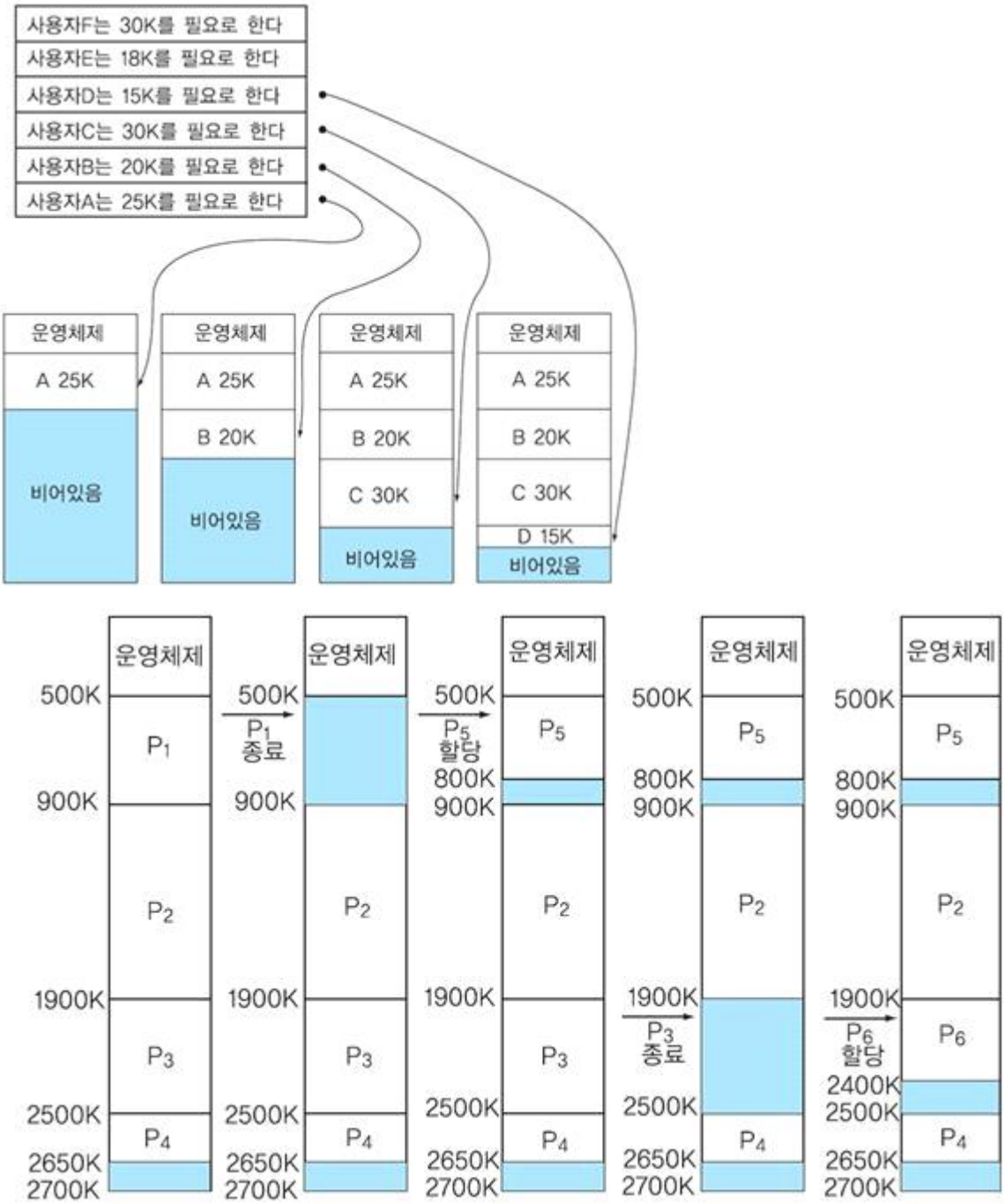
(2) 그러나 가변분할 다중 프로그래밍 조차도 단편화 현상이 생긴다. 초반에는 단편화 현상이 없지만, 프로세스가 종료됨에 따라, 단편화 현상이 생기는게 다음 그림이다.

(3) 공백의 합병

① 인접한 공백들을 결합하여 다른 프로세스가 들어올 수 있도록 함.(인접한 공백만 가능한 듯)

(4) 기억장소의 집약

① 여러 곳에 존재하는 작은 공백들을 하나의 커다란 공간으로 통합. 가비지컬렉션이라고 부르기도 함. (이 행위를 할 때에는 반드시 모든 프로세스가 멈춰야 함.) (메모리 이동은 복사라고 생각하면 됨.)



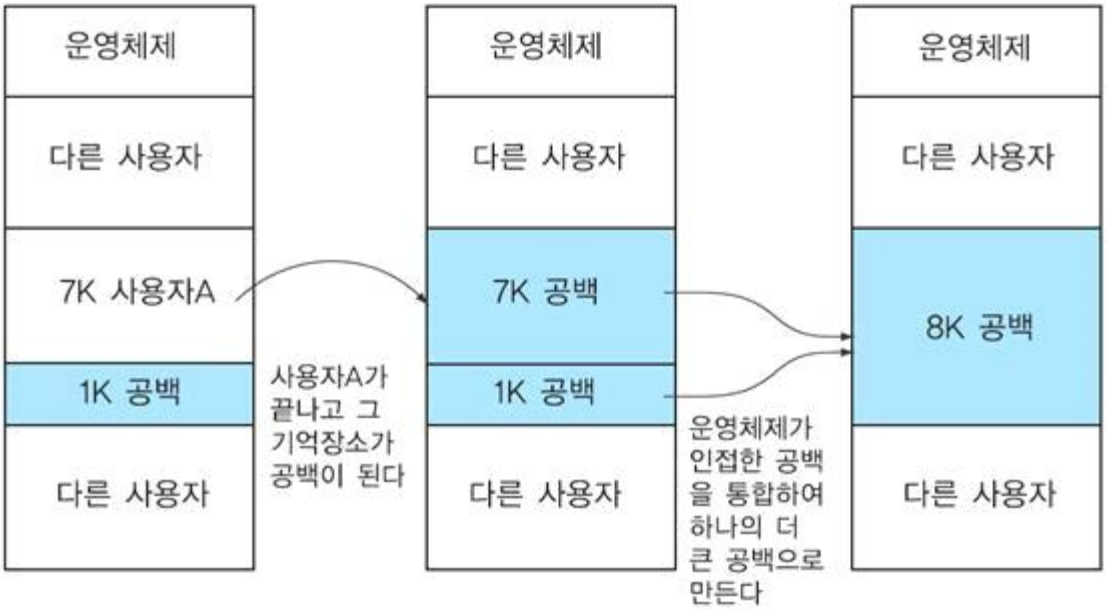

기타 여러 방법의 가비지 컬렉션.

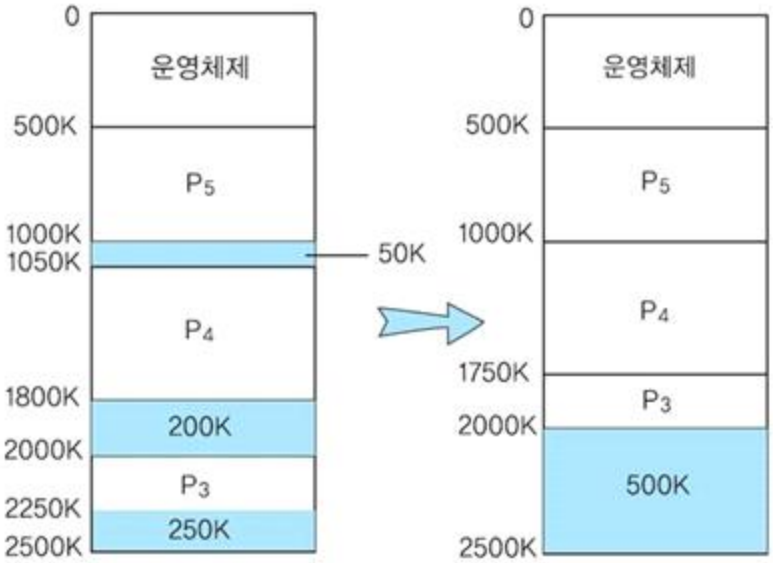
② 기억장소 집약의 단점.

생산적으로 사용될 수 있는 시스템 자원을 낭비할 수도 있음.

집약이 실행되는 동안 모든 일을 중지함

집약시 기억장치 내에 있는 작업들이 재배치 되어야함




정상적인 경우 기억장치 내의 작업 적재 환경은 급격히 변화되기 때문에 자주 집약해야 하는데, 이는 시스템 자원들의 소모가 집약으로부터 얻는 이익보다 더 커질 수도 있음.

(5) 기억장치 배치 전략.

① 입력되는 프로그램과 대이터를 주기억장치의 어느 곳에 넣을 것인가를 결정

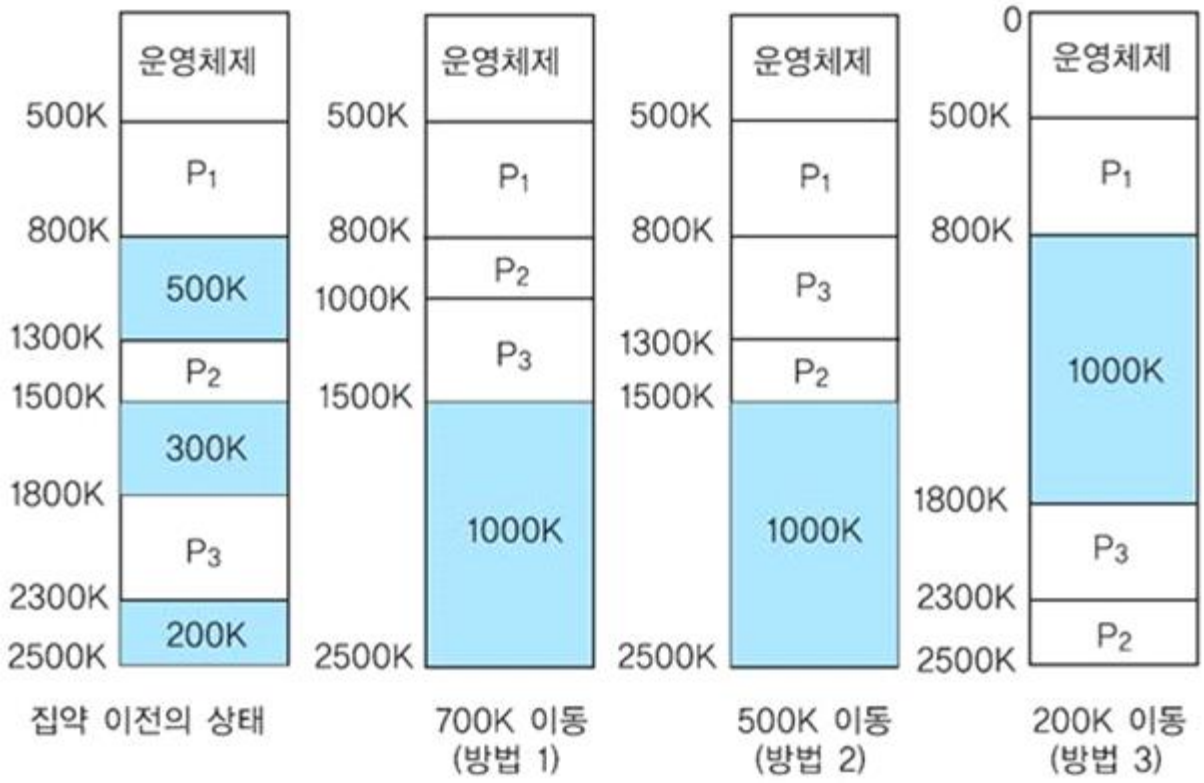
② 최초 적합 기법 (주기억장치의 첫 번째 유용한 공백을 우선적으로 선택)

③ 최적 적합 기법 ( 가장 적합한 공간을 선택, 기억장치의 단편화를 최소로 하는 방법)

④ 최악 적합 기법(가장 큰 공백에 배치

(6) 기억장치 교체(swaping)

① 하나의 작업이 전체 기억장치를 사용한 후, 필요에 따라 그 작업은 제거되고 다




시 다음 작업이 적재.

② 오늘날 일반적으로 사용되는 페이징 시스템의 기초.

8. 가상 메모리 관리

1) 하나의 프로세스가 한 번에 주기억장치 내에 존재하지 않고 일부만 있어도 수행하게 하는 방법을 제공
2) 실제 주소 공간의 크기에 구애받지 않고 보다 큰 가상주소 공간상에서 프로그래밍 가능
3) 주기억장치보다 크기가 큰 프로세스를 수행 가능
4) 가상 주소 공간을 구성하는 것을 가상메모리라 하고 실제 주소공간을 구성하는 것을 주기억장치라고 함.
5) 동적 주소 변환
   (1) 수행중인 프로세스가 참조하는 주소를 가상주소라 하며, 주기억장치상에서 이용할 수 있는 주소를 실제주소라고 함.
   (2) 프로세스가 수행될 때 가상주소를 실제 주소로 변환하는 대표적인 메커니즘
   (3) 인위적 연속성
      ① 가상 주소 공간상의 연속된 주소들은 실기억 공간에서도 반드시 연속적일 필요가 없는 특성.
6) 블록 사상(블럭 맵핑)
   (1) 가상 메모리에 블록 단위로 분할
   (2) 블록은 가상 메모리에 대한 분할 단위

(3) 페이지 / 페이징

① 블록이 모두 같은 크기 -> 페이지

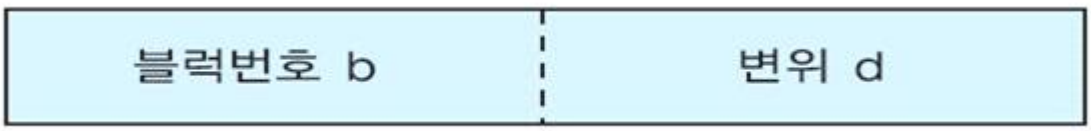
② 블록을 모두 같은 크기로 분할하는 것을 페이징이라 함.

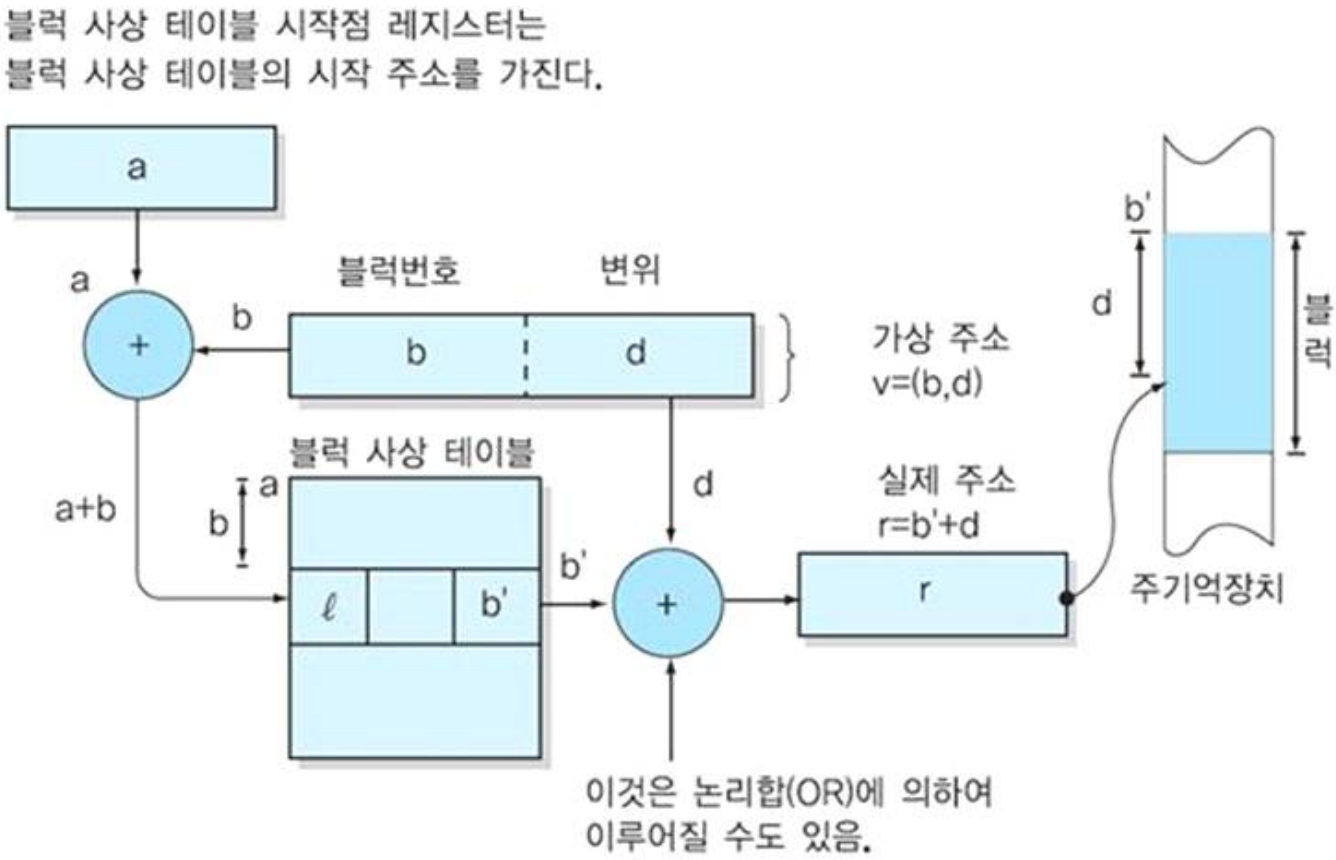
(4) 세그먼트 / 세그먼테이션

① 블록이 전부 다른 크기 -> 세그먼트

② 세그먼트로 가상 메모리 구성 -> 세그먼테이션

(5) 가상 주소는 V=(b,d)로 표시




9. 페이징

1) 일정한  크기의 블록

2) 페이징 시스템에서의 가상주소는 순서쌍 V = (p, d)로 표현

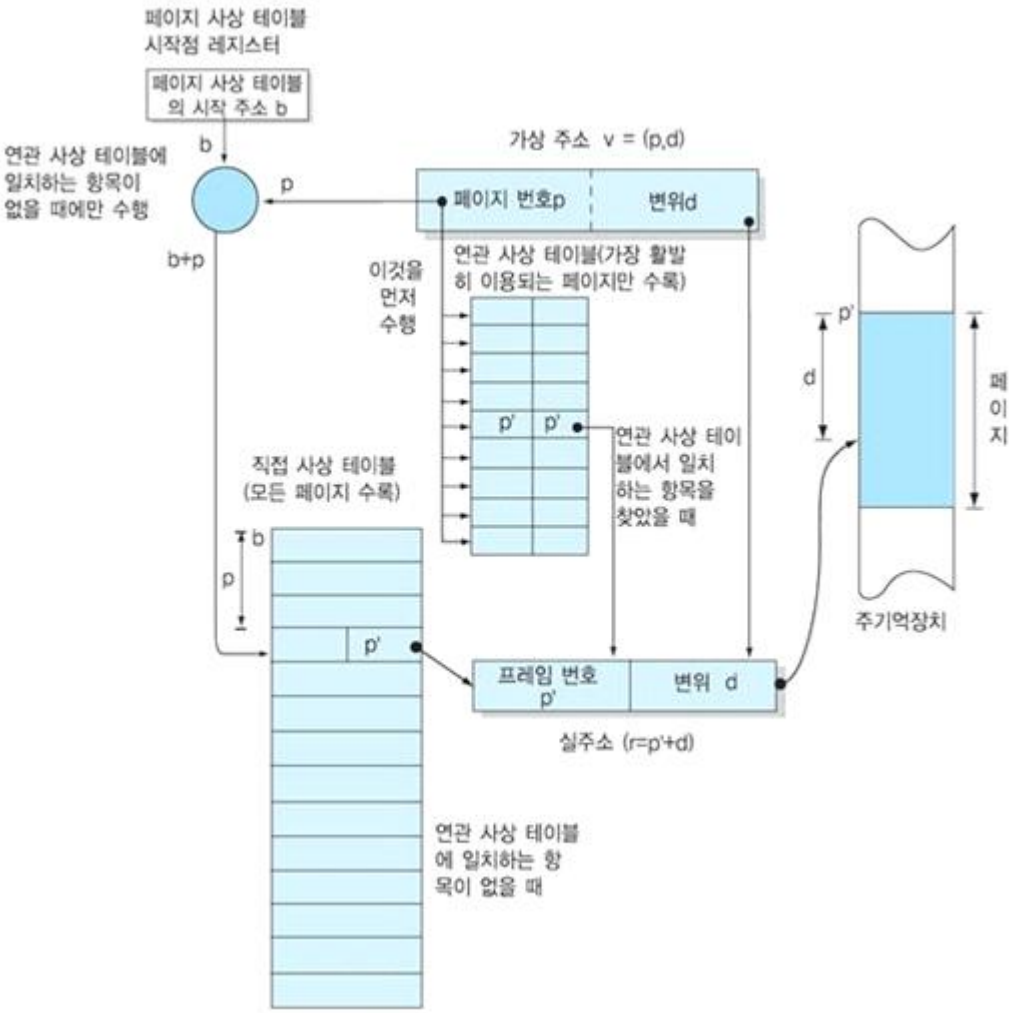
3) 동적 주소 변환

(1) 가상 주소는 페이지 사상 테이블을 통해 실제 주소를 구함

(2) Ex. 가상주소 v = (p,d) 일 경우, 실제 주소 r = p'(프라임은 실제 기억장치 상의 주소를 나타냄 ) + d

4) 페이지 존재 비트
   (1) 주기억장치 내에 존재할 때 1로 표시, 존재하지 않을 때 0으로 표시.
5) 직접사상
   (1) 아주 큰 페이지 사상 테이블은 보통 주기억장치에서 유지 관리.
   (2) 보다 빠른 주소 변환을 위하여, 속도가 매우 빠른 고속 캐시 기억장치를 이용하여 직접 사상의 페이지 사상 테이블을 구현
6) 연관사상 = 캐쉬메모리, 주소기반이 아닌 내용기반.
   (1) 저장된 값을 이용하여 데이터를 접근
   (2) 내용 주소화 메모리
   (3) 연관 기억장치 즉 캐쉬는 고가.
7) 연관 / 직접 사상
   (1) 가장 최근에 참조된 페이지는 조만간 다시 사용되기 쉽다는 사실을 이용하여 연관 기억장치에는 가장 최근에 참조된 일부 페이지 항목들만 수용.
   (2) 나머지는 직접 사상테이블에서 찾아야함.

8) 페이징 시스템의 공유.
   (1) 공유가 가능한 페이지를 가능한 한 공유
   (2) 순수 프로시듀어 <-(함수라고 생각하자) (프로시저 같은데 )



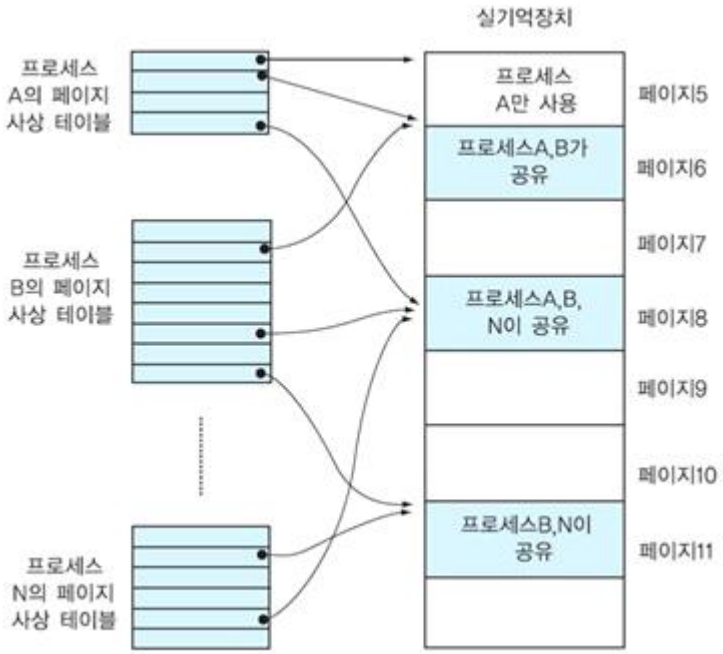

① 수정이 불가능한 프로시듀어

② 재진입가능 프로시듀어라고도 함.

9) 페이지 크기

(1) 고려내용

① 페이지 크기가 작으면 작을수록 페이지 테이블의 크기가 증가하여 기억 공간이 낭비. -> 테이블 단편화




② 페이지 크기가 작을수록 프로세스가 작업세트를 확보하는데 도움.

③ 페이지 단편화 현상을 초래. - 평균적으로 1/2페이지의 단편화.

④ 페이지가 크게 되면 참조되지 않을 많은 정보들까지 주기억장치로 옮겨지게 되어 기억공간의 낭비

⑤ 프로그램 실행 중 입출력 전송의 횟수를 줄이기 위해서는 페이지 크기가 클수록 효과적.

10) 세그먼테이션

(1) 세그먼트

① 논리적 단위가 되는 프로그램 모듈이나 자료구조 등.

(2) 직접 사상

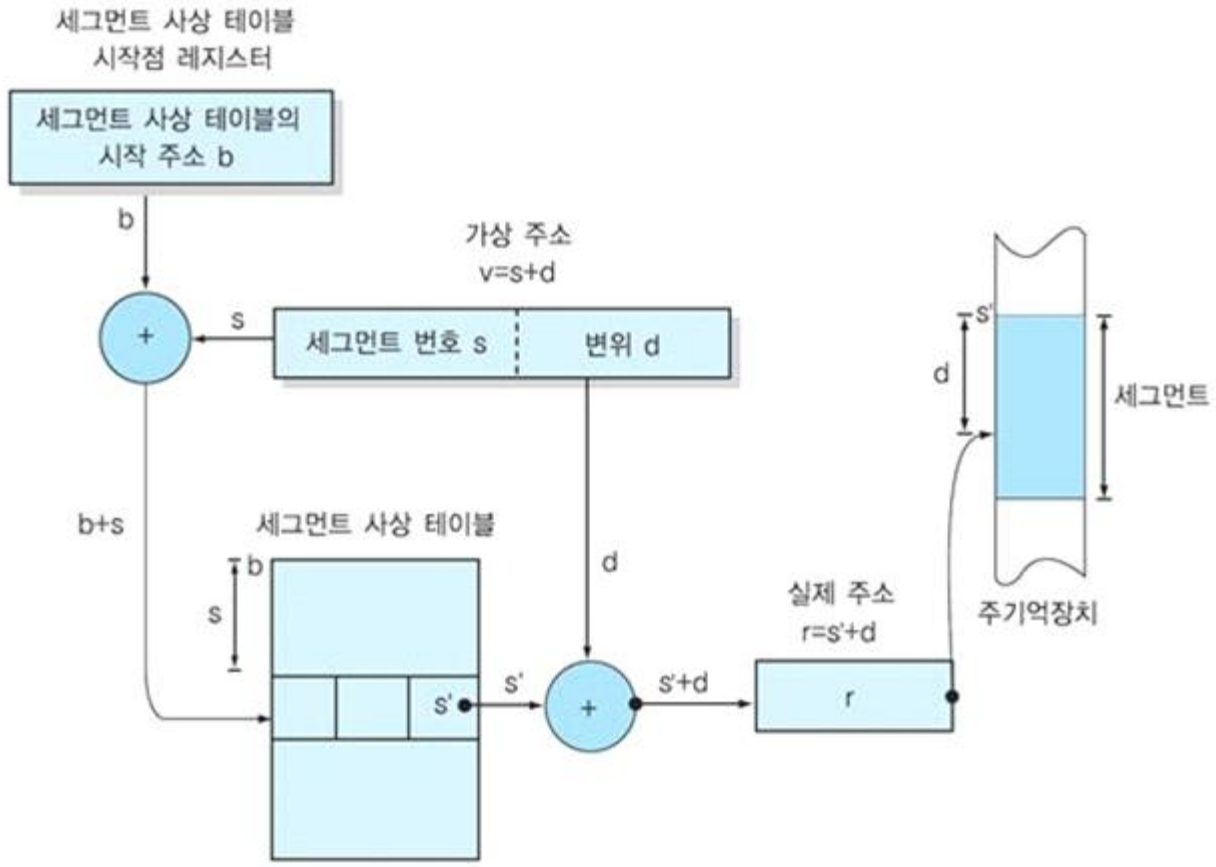
① 가상주소는 V=(s, d)로 표현

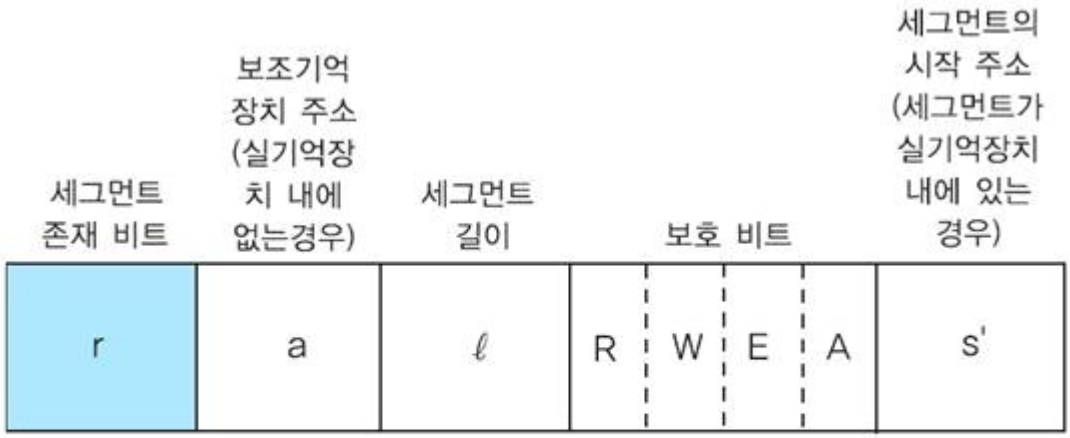

② 주기억장치에 옮겨지는 세그먼트는 그것이 들어갈 만큼의 충분한 주기억장치가 연속적으로 확보되어야 함.

(3) 존재 비트

① 해당 세그먼트가 주기억장치에 존재하는지 여부

(4) 세그먼테이션 공유 및 보호




r=0 : 세그먼트가 주기억장치 내에 없는 경우 R : 판독 접근
r=1 : 세그먼트가 주기억장치 내에 있는 경우 W : 기록 접근
보호 비트 : (1-예, 0-아니오) E : 수행 접근
A : 첨가 접근

① 페이징과 다르게 세그먼테이션은 주기억장치의 시작좌표만 알면 됨.

(5) 세그먼테이션의 장점 중 하나

① 세심한 접근 제어가 가능함.

② R = 읽기

③ W = 수정

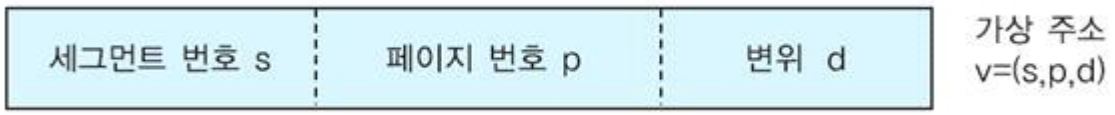
④ E = 수행

⑤ A = 첨가

11) 세그먼트 / 페이징 혼용 기법

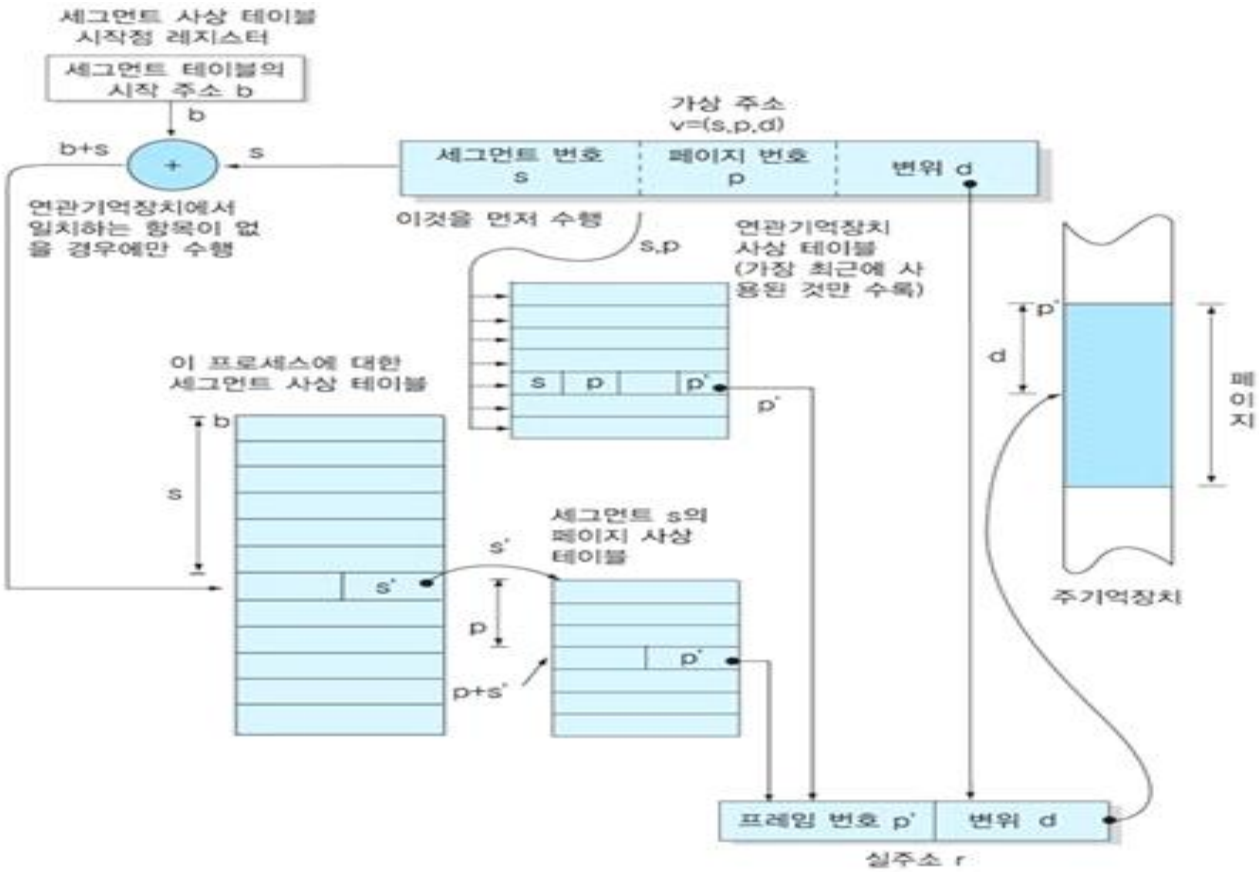
(1) 하나의 세그먼트를 정수 배의 페이지로 다시 분할하는 세그먼트 / 페이징 혼용기법

(2) 가상 주소 V는 3차원 요소로 구성 = V(s,p,d)



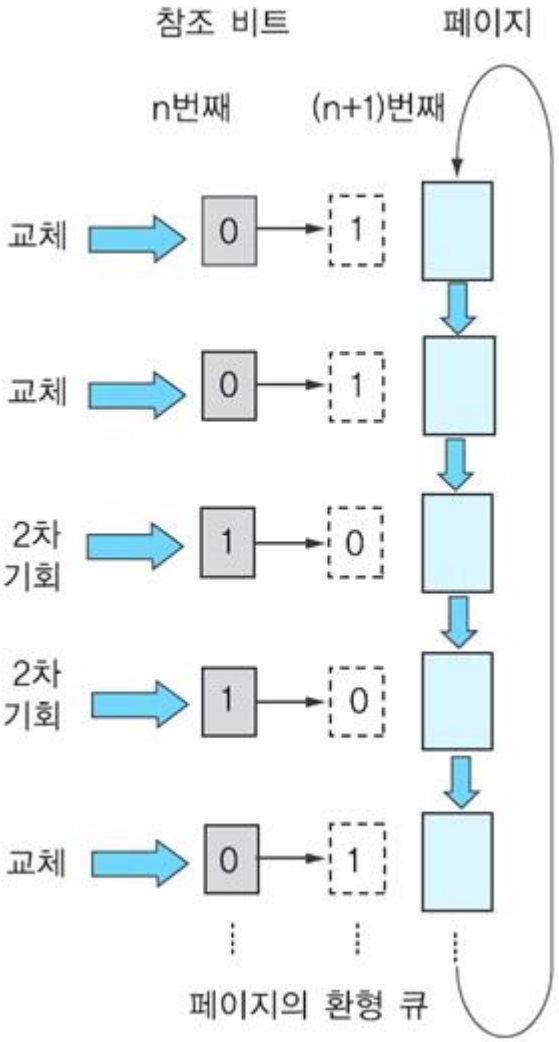

(3) 시스템 공유

① 세그먼트의 공유는 서로 다른 프로세스의 세그먼트 사상테이블의 항목들이 동일 페이지 사상 테이블을 공유



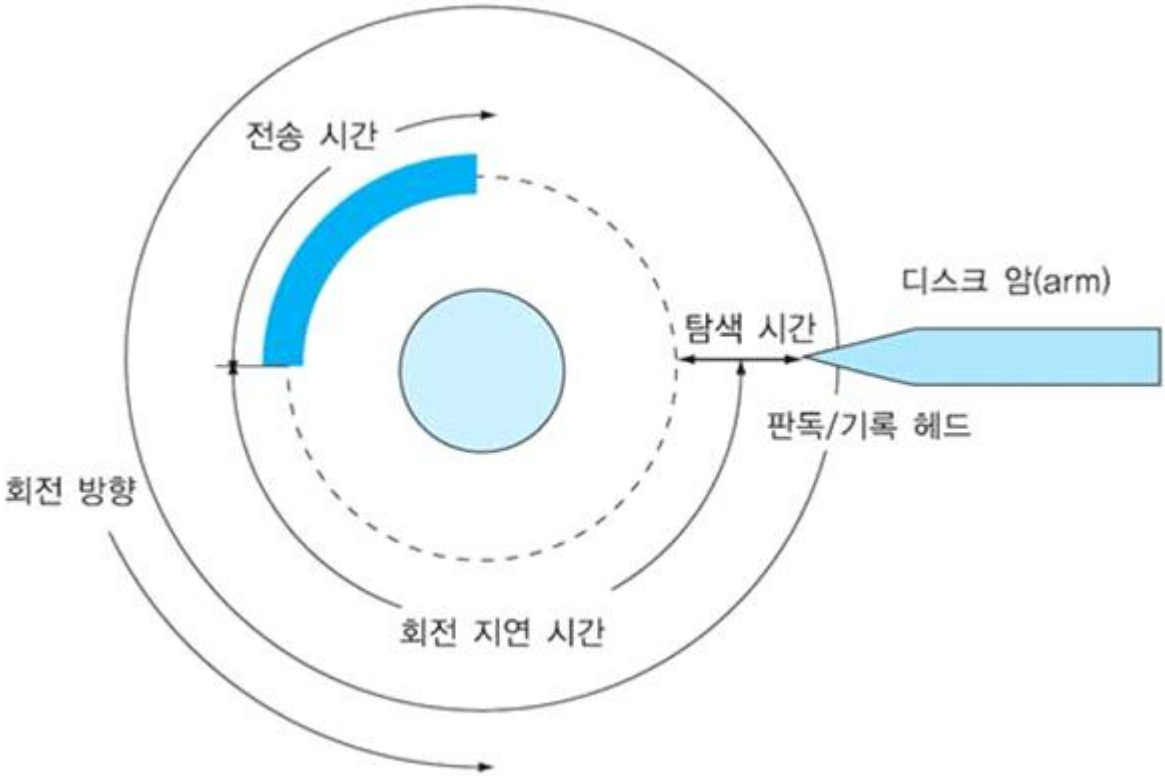
10. 페이지 교체 알고리즘
    1) 주기억장치 공간을 확보하기 위하여, 현재 주기억장치를 차지하고 있는 페이지들 중에서 어떤 페이지를 선택하는 기법.
    2) FIFO알고리즘
        (1) 가장 먼저 주기억장치에 들어와 있는 페이지를 교체시키는 방법.
        (2) FIFO 모순 또는 Belady의 이변
            ① 주기억장치 내에 페이지 프레임 할당이 많아졌을 때, 페이지 부재가 적어질 거라 생각 하지만, 실제로는 부재가 더 증가되는 경우도 있음.
    3) 최적 교체 알고리즘
        (1) 앞으로 가장 오랫동안 사용되지 않을 페이지를 교체
        (2) 최소의 페이지 부재율을 가짐
        (3) 사전에 미리 파악하고 있어야 하기 때문에 다루기가 어렵.
    4) LRU알고리즘
        (1) 한 프로세스에서 사용되는 각 페이지마다 타임-스탬프용 카운터나 스택을 두어 현 시점에서 가장 오래전에 사용된 페이지를 제거.
        (2) 카운터에 의한 방법
            ① 어떤 페이지에 대한 참조가 있을 때마다 그 시간 레지스터의 내용은 페이지 테이블 내에 있는 사용 시간 필드에 복사.
        (3) 스택에 의한 기법
            ① 스택의 꼭대기에 있는 페이지가 가장 최근에 사용된 것임을 나타내고 바닥에 있는 것이 가장 오랫동안 사용되지 않은 페이지
        (4) 이중연결리스트로 구현되는 것이 일반적.
    5) 2차 기회 알고리즘
        (1) 하드웨어의 지원이 반드시 요구되는 LRU알고리즘에 대한 접근방안의 하나로서 참조비트를 이용하는 방법
    6) LFU 알고리즘
        (1) 가장 적게 사용되거나 집중적이 아닌 페이지가 대체

11. 쓰래싱




1) 페이지 부재가 계속적으로 발생되어 프로세스가 수행되는 시간보다 페이지 교체에 소비되는 시간이 더 많아지는 경우.
2) 구역성
   (1) 국부적인 부분만을 집중적으로 참조를 의미
3) 시간 구역성
   (1) 최근 참조된 기억장소가 그 후에도 참조될 가능성이 높음.
      ① 예로 순환, 서브루틴, 스택, 카운팅
4) 공간 구역성
   (1) 하나의 기억장소가 참조되면 그 근처의 기억장소가 계속 참조
      ① 예로 배열 순차코드 등등
5) 작업세트
   (1) 프로세스에 의해 자주 참조되는 페이지들의 집합체

12. 디스크 스케줄링 및 파일 시스템 (여기서부터 기말고사)
    1) 파일 시스템
        (1) 전체적인 정보 관리 시스템의 한 부분
    2) 현재의 컴퓨터와 시스템은 주로 디스크 시스템 중심의 처리
    3) 운영체제는 디스크나 CD-ROM같은 기억용량이 큰 기억장치를 관리 및 운영함.
    4) 디스크 구조
        (1) 섹터
            ① 디스크 구조에서 부채꼴 모양으로 자른 것처럼 나누어진 구역
            ② 보통 512바이트
        (2) 트랙
            ① 중심축에 대해 동심원으로 나누어진 것
        (3) 블록
            ① 섹터와 트랙의 교차점으로 둘러싸인 각각의 구역
            ② 보통 몇 개의 섹터를 모아 블록이라고 함
            ③ 입출력의 기본 단위
        (4) 실린더
            ① 동일한 인덱스 번호를 가진 트랙의 모임 (하드디스크엔 여러 디스크가 있는데 각 층의 같은 트랙들을 부르는 말)
        (5) 탐색시간
            ① 실제 원하는 실린더를 찾는데 소요되는 시간. (암이 앞 뒤로 움직이는 시간)
        (6) 회전지연시간
            ① 원판이 회전하면서 원하는 섹터가 있는 곳 까지 가는 시간.
        (7) 전송시간
            ① 읽은 데이터를 주기억장치에 전달하는데 소요되는 시간
        (8) 디스크 접근 시간
            ① 위의 모든 시간을 다 더한 시간.




5) CD-ROM
   (1) 적은 비용으로 많은 데이터를 저장할 수 있음. (650MB 이상)
   (2) 레이저를 이용한 재생방식으로 수명이 김.
   (3) 레이저로 깍기 때문에 read only memory임
6) 디스크 저장 매체의 두가지 저장방식
   (1) CAV
      ① 등각 속도 방식 ( 안쪽과 바같쪽이 동일한 각도 크기만큼 이동, 일정한 속도로 회전하기 때문에, 읽어드리는 데이터 양이 달라진다. 하드디스크와 플로피디스크가 이 방식.)
   (2) CLV(Constant일정 Angular각 Velocity속도)
      ① 등선 속도 방식
      ② 회전 속도를 조절하여, 데이터를 읽고 쓰는 속도가 일정
      ③ 데이터가 디스크의 중심에 얼마나 가까이 있느냐에 따라, 디스크 회전 속도 변화.
      ④ 데이터는 연속된 나선형 트랙을 따라 저장.
      ⑤ 연속적인 오디오 또는 비디오 트랙에 적합.
      ⑥ 단점 : CLV는 저장용량이 큰 반면, 데이터 접근시간이 느림.
7) 디스크 스케줄링
   (1) 탐색시간을 최적화하는 방법 (일반적인 방법임)

① 탐색시간은 회전지연시간에 비해 시간소요가 크기 때문에.

(2) 회전 지연 시간을 최적화하는 방법

8) FCFS 스케줄링

(1) 먼저 도착한 요청을 우선적으로 서비스.

(2) 공평성 보장

(3) 우선순위에 따라 실행순서가 변경되지 않음.

(4) 대기 중인 요청들 간의 위치에 대한 관련성을 무시하기 때문에 좋은 방법이 아님.

(5) 탐색 패턴을 최적화하려는 시도가 없는 스케줄링기법 즉 효율이 낮음.

9) SSTF(Shortest seek time first)스케줄링

(1) 현재 헤드의 위치에 가장 가까운 요청을 먼저 서비스하는 기법

(2) SJF알고리즘 형태

(3) 기근현상 발생 가능성 있음.

(4) 가운데 트랙이 더 많은 서비스를 받을 수 있기 때문에 응답시간에 편차가 생길 수 있음.

(5) FCFS보다 처리율이 높고 평균 응답시간이 짧음.

10) SCAN 및 LOOK 스케줄링

(1) SCAN알고리즘은 진행 방향 상 가장 짧은 거리에 있는 요청을 서비스하는 기법.

(2) 헤드가 디스크의 한쪽 끝에서 반대편 끝까지 이동

(3) 끝에서 다시 되돌려 돌아오는 방식.

(4) 엘리베이터 방식과 유사하기 때문에 엘리베이터 알고리즘이라고도 부름

(5) 진행도중 새롭게 도착하는 요청도 함께 서비스

(6) SSTF방법의 헤드가 높은 편차를 갖고 움직이는 단점이 보완

(7) 실제로 구현되는 디스크 스케줄링의 기본 전략

(8) LOOK알고리즘은 헤드가 끝까지 가지 않고 마지막 요청 트랙까지만 이동.

11) C-SCAN 및 C-LOOK 스케줄링

(1) 한쪽 방향으로 헤드를 이동하면서 진행 방향 상 가장 짧은 거리에 있는 요청을 처리.

(2) 끝까지 가면 처음부터 다시, 끝에서 다시 같은 방향으로 처리를 진행하는 방식.

(3) 안쪽과 바깥 쪽 트랙의 차별대우를 개선, 대기시간의 편차가 매우 짧음.

(4) C-LOOK 알고리즘을 적용하면 헤드가 각 방향의 트랙 끝까지 이동하지 않고 마지막 요청 트랙까지만 이동.

12) 알고리즘 선택

(1) 디스크 서비스에 대한 마지막 요청은 파일 할당 방법에 따라 많은 영향

(2) 연속적으로 할당되어 있는 파일의 관리를 위한 것은 결과적으로 헤드의 움직임은 큰 문제가 되지 않음. (붙어있는 파일들)

(3) 링크된 파일이나 색인화 된 파일은 헤드의 이동에 관심을 가지고 더 좋은 디스크

이용방안 모색해야함( 디스크 상에 산재해 있는 레코드를 취급해야 하기 때문)

(4) 디렉터리와 색인 블록의 위치또한 중요.(디렉터리를 디스크 끝 부분에 두는 것 보다 중간 부분에 두는 것이 디스크 헤드의 이동을 상대적으로 감소)

13. 파일 시스템

1) 관련된 정보를 포함하는 실제적인 파일들의 집합체

2) 시스템 내의 모든 파일에 대한 정보를 제공하는 디렉터리 구조

3) 파일이란

(1) 일반적으로 작성자와 사용자에 의해 그 의미가 정의된 비트, 바이트, 행(line), 또는 레코드들의 연속체

(2) 하나의 파일에는 그 파일의 이름, 형태, 작성 시기, 작성자, 길이 등의 여러 속성이 있음.

4) 데이터의 계층 구조

(1) 필드 - 상호 관련있는 문자들의 집합.

(2) 레코드 - 서로 관련이 있는 필드들의 집합

① 레코드 키 - 어떤 레코드를 다른 레코드로부터 식별하는 데 사용되는 제어필드

(3) 파일 - 상호 관련 있는 레코드들의 집합

(4) 데이터베이스 - 상호 관련 있는 파일들로 구성


5) 블록킹

(1) 물리적 레코드나 블록

① 기억매체에 출력되거나 기억매체로부터 입력되는 실제 정보의 단위

(2) 논리적 레코드

① 사용자 관점에서 취급되는 자료 집단의 단위

(3) 블록킹되지 않은 레코드

① 물리적 레코드가 단 하나의 논리적 레코드로 구성 (메인메모리에 올라가는 문장이 단어 한 개라고 생각하면 됨.) (비디오로 생각하면, 논리적 레코드는 1초, 하

지만 물리적 레코드는 실제로 1초단위가 아니라 비트단위)

(4) 블록킹된 레코드

① 여러 개의 논리적 레코드가 하나의 물리적 레코드를 구성(여러개의 논리적데이터가 하나의 블록을 만들고 그 블록이 모여 파일이 되는데, 즉 한글(논리적 레코드)이 모여 단어(블록)를 만들고 단어가 문장(파일)이 되는 것. 즉 올라가는 건 하나의 문장)

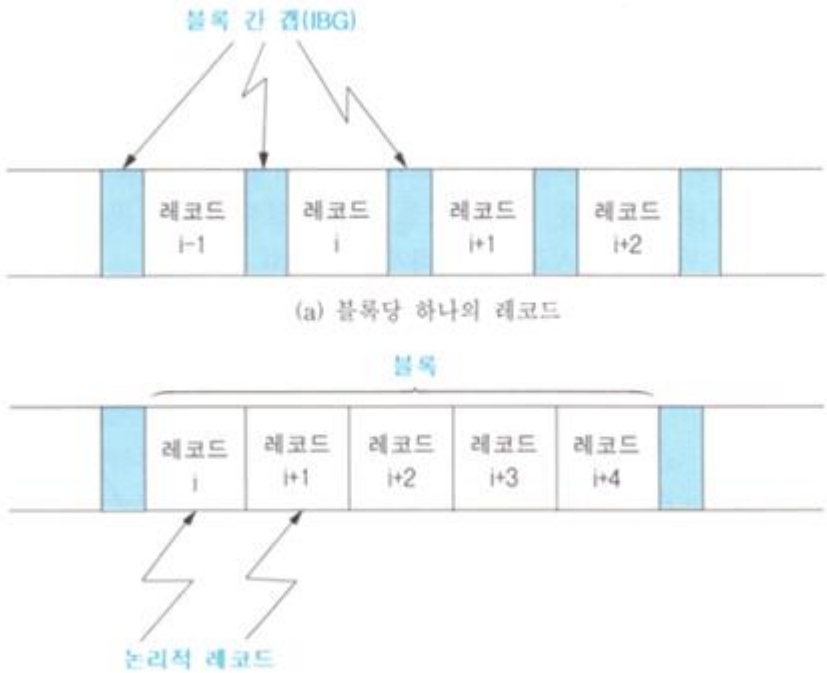
(5) 고정길이 레코드

① 구성된 파일에서의 레코드 길이는 모드 같음

(6) 가변길이 레코드(회원가입 시 필수입력사항이 아닌 것들 )

① 구성된 파일에서의 레코드 길이는 다양

② 최대 크기는 블록의 크기와 동일




6) 파일시스템의 기능

(1) 사용자가 파일을 생성, 수정, 삭제 가능.

(2) 파일 공유를 위한 제어 방법 제공

(3) 여러 가지 접근 제어 방법 제공(여러 경로를 통해 접근이 가능)

(4) 다양한 형태로 파일을 재구성 방법 제공(옮기고 붙이고 등등 수정?)

(5) 백업과 복구를 위한 기능(휴지통에서 다시 돌릴 수 있음)

(6) 사용자와 장치간의 독립성을 유지하기 위해 기호화된 이름 재공

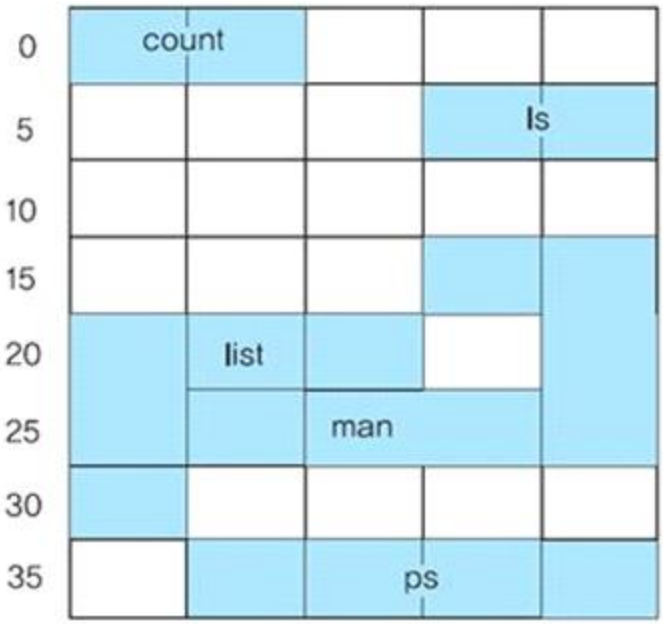
(7) 정보가 보호되고 비밀이 보장될 수 있도록, 암호화 복호화기능 제공

(8) 인터페이스 제공

7) 파일의 구조

(1) 파일을 구성하는 레코드들이 보조기억장치에 배치되는 방법.
(2) 순차 파일
① 논리적인 레코드를 물리적인 순서에 따라 순차적으로 저장 및 검색하도록 저장.
② 디스크, 자기 테이프, 프린터로 된 출력 등에 사용
③ 순차접근(SASD) 기억장치에 저장.
④ 일괄처리에서 많이 사용
⑤ 빠른 다음 레코드 접근이 쉬움, 단점은 파일 접근 방식이 저장된 레코드 순서와 다를 때 프로그램 접근 속도가 저하. (즉, 비디오 같은거면, 도중부터 보기 이런거?)
(3) 직접 파일
① 다른 레코드를 참조하지 않고 임의 레코드를 직접 접근 가능한 파일 구조
② 특정 응용 분야에 적합한 순으로 레코드를 직접 접근 기억장치에 저장.
③ 키 값에서 보조기억장치의 주소로 사상시키는 사상함수가 필요
④ 특정 레코드를 저장하고 있는 기억장소의 주소를 신속하게 계산해야 함
⑤ 대화형 처리에 유용
⑥ 디스크와 같은 직접 접근 기억장치에 저장 (DASD)
⑦ 장점은 특정 레코드의 검색, 삽입, 수정, 삭제가 쉬우면 단점은 키 값의 순서에 의한 순차 검색이 어려움.
(4) 색인된 순차 파일
① 순차 및 직접 접근을 모두 처리할 수 있는 파일 구조
② 레코드는 각 레코드의 키 값에 따라 논리적 순으로 배열
③ 레코드는 키 값에 순차적으로 접근될 수도 있고, 시스템에 의해 관리되는 인덱스 블록의 검색을 통하여 직접 접근도 가능
④ 보통 디스크와 같은 직접 접근 기억장치에 저장
⑤ 일괄처리 및 대화형 처리를 목적으로 하는 파일 지원
⑥ 장점은 융통성이 많고 검색 성능도 우수하며, 단점은 설계시 고려할 사항이 많음.
(5) 파일 공간의 할당과 회수
① 보조기억장치상의 공간은 작은 공간들로 단편화되는 현상이 발생.- 문제 해결을 위해 주기적인 집약을 수행, - (가용공간의 집합은 한 개의 큰 불록이나 블록의 그룹이 되도록 결합.)
(6) 비트백터
① 비트 맵 혹은 비트 테이블이라고도 불림
② 각 디스크 블록 당 하나의 비트가 할당되어서 관리하는 방법
③ 해당 블록이 자유공간이면 1로 표현되고, 0이면 파일이 할당된 상태
④ 자유 블록이나 연속된 자유 블록들을 찾기가 쉬운 장점
⑤ 리눅스 운영체제의 자유 공간 관리에 비트 백터를 사용.
(7) 연속할당
① 파일들이 보조기억장치 내에 연속적으로 인접된 공간에 파일을 할당.
② 장점 - 연속하는 논리적 블록들이 물리적으로 서로 인접할 경우 빠른 접근이 가능
③ 단점 - 새로운 파일을 생성하기 위해서는 공간 크기를 미리 명시, 만약 원하는 만큼 기

억공간이 확보되지 않으면 그 파일은 생성되지 못함, 새로운 파일들은 비어있는 가용 공간의 크기와 일치해야 함, 충분한 기억장소를 확보하기 위하여 주기적인 집약이 필요. 데이터가 항상 꽉 차있는 것이 아니기 때문에 단편화는 필수적.



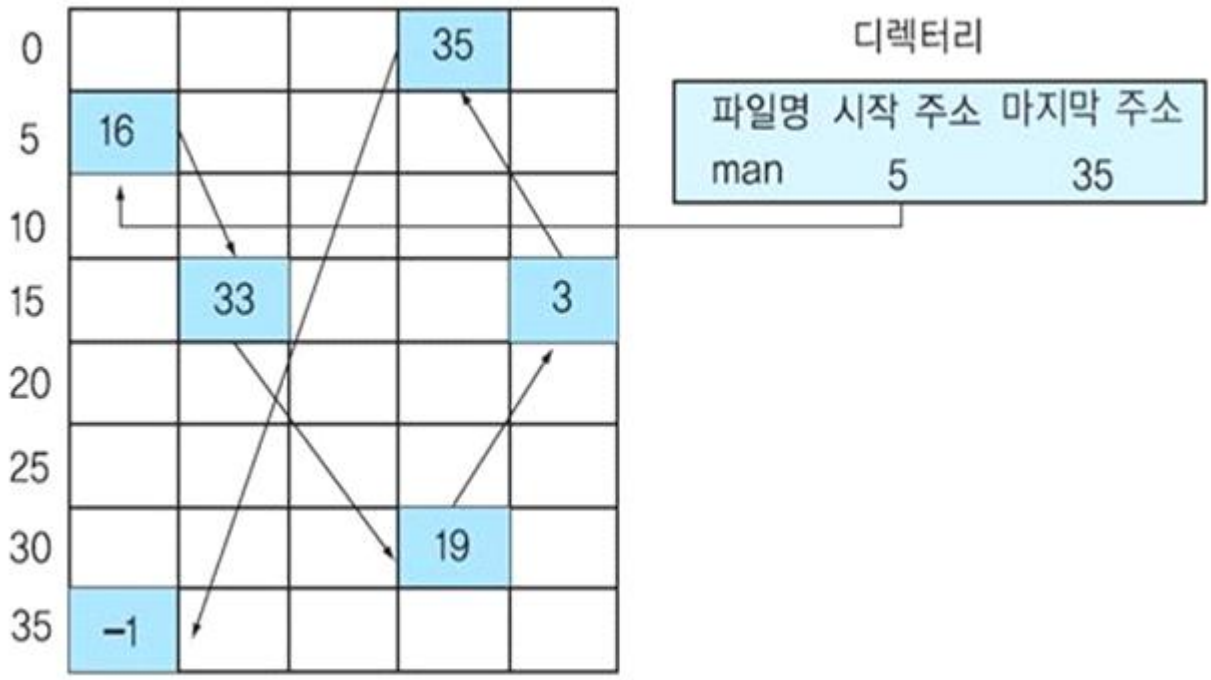

디렉터리

| 파일명 | 시작 주소 | 길이 |
|---|---|---|
| count | 0 | 2 |
| ls | 8 | 2 |
| list | 18 | 5 |
| man | 24 | 7 |
| ps | 36 | 4 |

(8) 불연속 할당(두 가지 방법)
① 연결리스트
② 색인 블록

(9) 불연속 할당 (연결리스트)
① 동일한 파일에 속해 있는 섹터들이 서로 연결 리스트의 형태를 취하면서 다른 것과의 연결을 위한 포인터를 지님.
② 디렉터리에는 처음 해당 파일이 시작되는 시작 주소 및 마지막 주소에 대한 포인터정보 유지.
③ 문제점 - 연속된 블록들의 검색에는 긴 시간이 요구
④ 연결리스트 구조를 유지하는데 필요한 시간이 추가적으로 소요
⑤ 연결리스트 내에 있는 포인터들은 파일 데이터를 위한 가용공간을 감소. (예 - block size - 512b 최소 저장공간 32비트 기준 32비트 = 4바이트 실제 주소 제외 저장공간 = 512b - 4b = 508b)

(10) 불연속 할당 (색인 블록)
① 각 파일마다 하나의 색인 블록을 두고, 여기에 파일의 블록 항목이 산재해 있는 주소에 대한 포인터 정보 유지
② 단편화의 문제 없이 해당 블록에 대한 직접 접근이 가능
③ 연결리스트보다 기억장소의 낭비를 초래할 가능성 농후, (예로 두 개의 섹터를 가지는 파일에 대하여 색인 블록 방법에서는 두 포인터만을 사용하기에 나머지 부분은 낭비.
④ 연결기법(링크드 스키마) - 색인 블록간 포인터 사용




⑤ 다중색인 - 다단계 색인 쁠록 사용 ( 다단계 페이징과 유사)

⑥ 혼합기법 - 직접 블록과 다중색인을 결합한 내용. (너무 큰 파일의 경우의 해결방법들 3개)

(11) 파일의 보호

① 물리적인 손상으로부터의 보호와 파일에서 부당한 접근으로부터의 보호

(12) 보호 기법

① 파일 이름 - 다른 사용자가 파일의 이름을 알지 못하는 경우에 접근 불가.

② 암호 - 장점 - 보호목적으로 작은 노력과 기억장소가 소요., 단점 - 파일 보호가 불확실, 쉽게 노출될 수 있으며, 불법 사용자가 바뀐 암호를 사용.

③ 접근 제어 - 사용자에 따라 접근이 가능한 유형을 다르게 명시 unix시스템에서는 세종류의 접근유형 (r : read, w : write x : execute)와 세 종류의 사용자 유형(파일 소유자, 그룹 사용자, 모든 사용자)을 정의하여 접근 제어

8) 디렉터리 구조

(1) 하나의 파일은 장치 디렉터리 또는 VTOC(VTOC)에 있는 항목으로 표현

(2) 장치 디렉터리는 그 장치 안에 있는 모든 파일들에 대한 파일 이름, 적재위치, 크기, 형태 등의 관한 정보

(3) 파일 시스템 내부에 있는 많은 파일들을 조직화하는 메커니즘

9) 디렉터리에서 실행되어야 할 기능

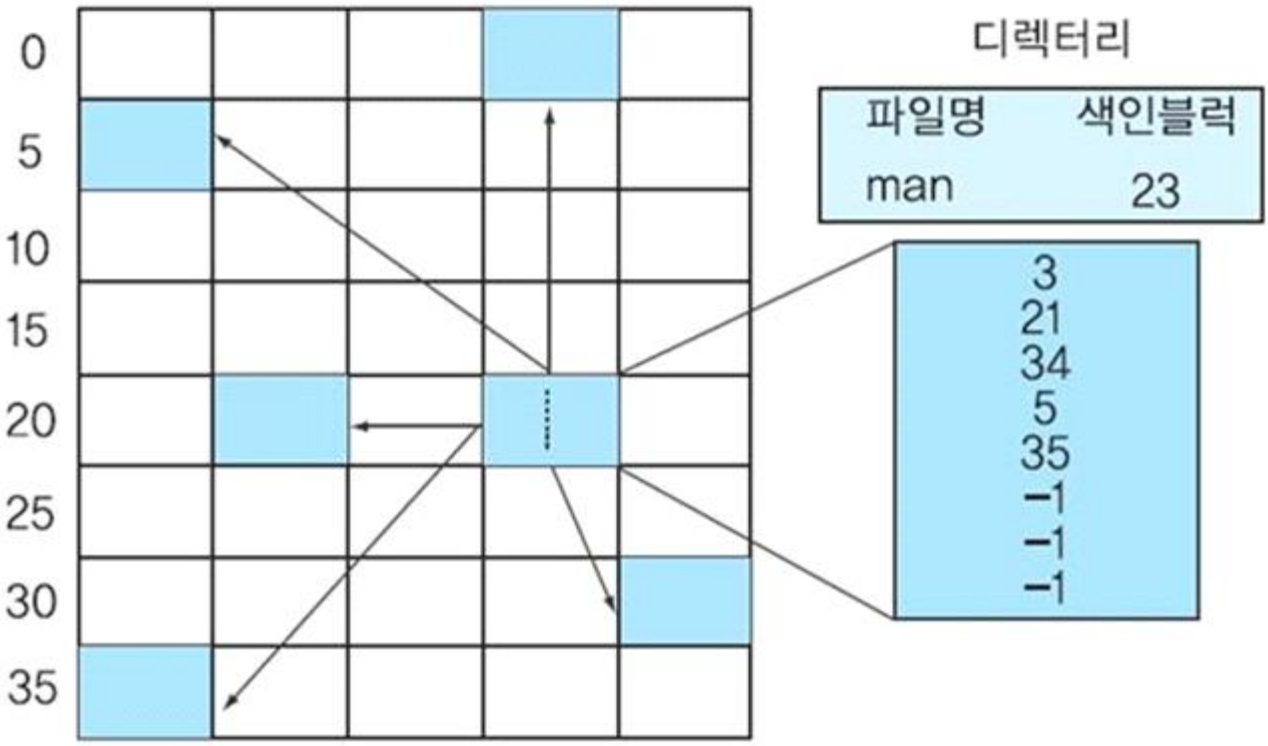
(1) 탐색

① 특정 파일에 대한 항목을 발견하기 위해서 디렉터리를 찾는 기능

(2) 파일생성

① 새로운 파일들을 생성해야 할 필요가 있으면 이를 디렉토리에 첨가하는 기능

(3) 파일삭제




① 파일이 더 이상 필요 없을 때 디렉터리에서 삭제 기능

(4) 디렉터리 리스트

① 디렉터리 내용들을 보여줄 수 있어야하고 그 리스트 내의 각 파일에 대한 디렉터리 항목의 내용을 보여줄 수 있어야 하는 기능.

(5) 백업

① 만약의 경우에 대비하여 일반적으로 파일들을 자기 테이프 등에 복사하는 기능.

10) 디렉터리 내에 저장되어 있는 각 파일에 대한 정보

(1) 파일명

① 기호화된 파일의 이름

(2) 파일형태

① 각기 다른 형태의 파일이 있기 때문에, 시스템들을 위한 파일 형태.

(3) 위치

① 장치에서 파일 위치를 가리키는 포인터

(4) 크기

① 파일 크기와 허용되는 최대 크기

(5) 보호

① 읽기 쓰기 처리 등을 제어하기 위한 접근 제어 정보

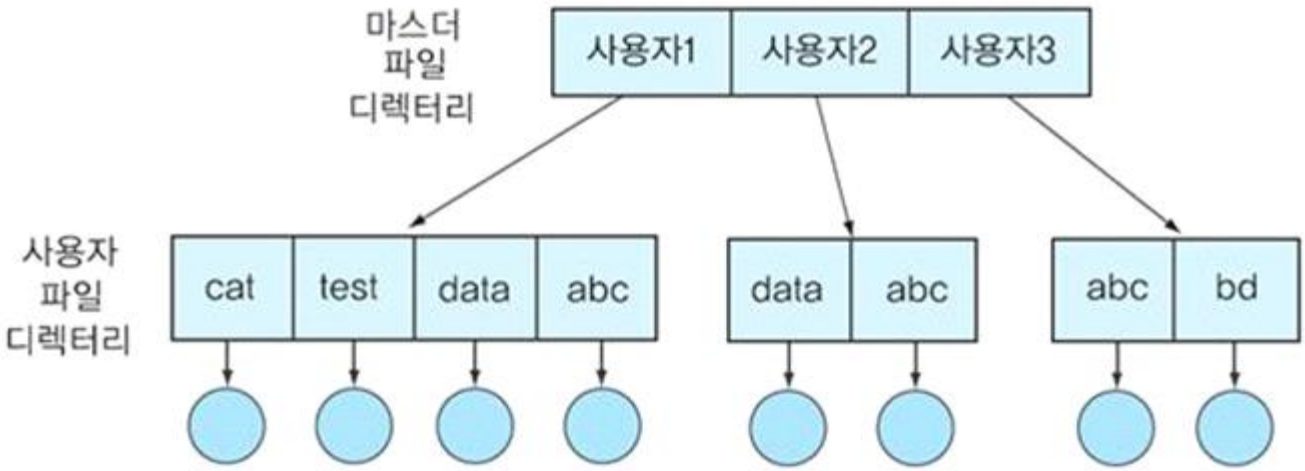
(6) 사용횟수

① 현재 이 파일을 사용하고 있는 프로세스들의 개수

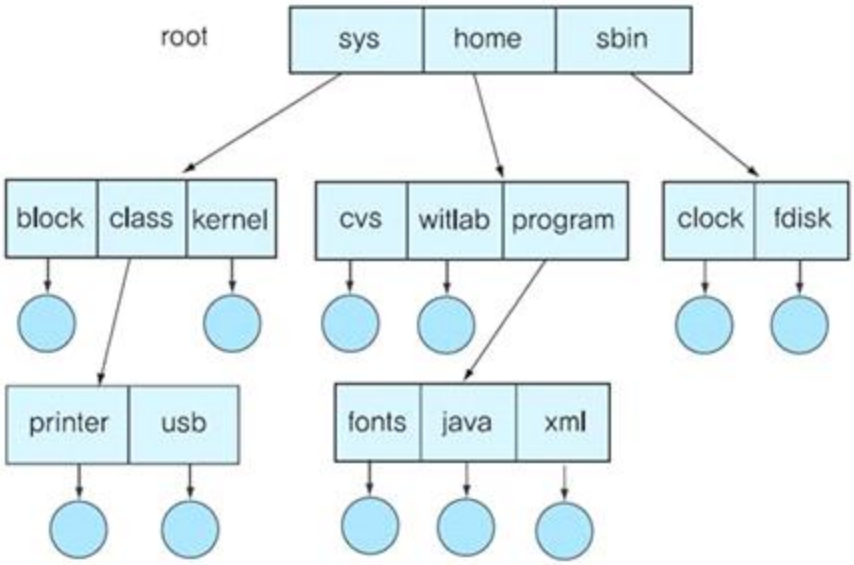
(7) 시간,날짜,프로세스 식별

① 보호와 사용 모니터링에 유용

11) 일단계 구조 디렉터리
   (1) 모든 파일들을 같은 디렉터리 내에 위치
   (2) 같은 디렉터리 내에 모두 상의한 이름을 가져야 함
12) 이단계 구조 디렉터리( 기본 폼을 가지고 사용자마다 하나씩 주는 것)
   (1) 2개의 단계인 마스터 파일 디렉터리, 사용자 파일 디렉터리로 구성
   (2) 일단계 디렉터리에서의 단점인 서로 다른 사용자들 간의 파일명의 혼란을 해결.
   (3) 새로운 사용자를 받아들이려고 할 때, 우선 시스템의 마스터파일 디렉터리를 검색.
   (4) MFD에는 사용자의 이름이 등록되어 있고, 각 항목은 한 사람의 사용자에 대한 사용자 디렉터리를 가리킴.




13) 트리 구조 디렉토리
   (1) 이단계 디렉터리 구조의 확장을 위한 일반화된 방법
   (2) 디렉터리 또는 서브디렉터리는 일단의 파일이나 또 다른 서브디렉터리들을 가짐.
   (3) 유닉스 파일 시스템은 트리로 구성
   (4) 경로 이름
      ① 절대 경로 이름 - 루트에서 시작하여 해당 파일이나 서브디렉터리에 이르는 경로상의 모든 디렉터리 이름을 정의(c드라이브/유저/.... 같은 것)
      ② 상대경로이름 - 현재 디렉터리에서부터의 경로를 정의(지금 있는 위치에서부터의 경로)
   (5) 삭제 방법
      ① 디렉터리가 비어 있다면 그 디렉터리는 간단히 삭제
      ② 비어있지 않다면, - 디렉터리를 삭제하기 위해서는 먼저 디렉터리 내에 있는 모든 파일을 삭제해야함. -디렉터리의 삭제 요구가 디렉터리 내의 모든 파일 뿐 아니라 서브디렉터리들도 제거해야 한다는 조건하에서 수행. 유닉스의 rm명령




14) 비순환 구조 디렉터리
    (1) 트리 구조로 된 디렉터리 시스템 중 가장 일반적인 방법
    (2) 디렉터리들이 서브디렉터리나 파일을 공유할 수 있도록 허용하고 순환을 허용하지 않음.
    (3) 공유방법
        ① 소프트 링크(노트필기 사진으로 올려야함 .)
        ② 하드링크
15) 일반적 그래프 구조 디렉터리
    (1) 순환이 허용
    (2) 그래프 탐색 알고리즘도 간단
    (3) 무한 순환이 되기 때문에, 카운터를 하거나 방문 횟수 제한을 해야함.
    (4) 각 디렉터리마다 불필요한 파일의 제거를 위한 쓰레기 수집
16) 파일시스템의 예
    (1) FAT
        ① 1개 파일은 4GB
        ② 2TB까지 지원
    (2) NTFS

① 암호기능 제공

(3) UFS

① 유닉스 계열 운영체제에 쓰임.

(4) Ext

① 리눅스 기본 파일시스템

(5) GFS

① 구글이 개발한 클라우드 서비스를 위한 시스템

② 청크 = 블록개념

③ 청크서버랑 처으랑은 다른 개념

(6) HDFS

① 하둡...

14. 프로세스 간 동기화 및 통신

1) 병행 프로세스

(1) 두 개 이상의 프로세스가 동시에 수행

(2) 서로 관련 없이 독립적으로 수행

2) 협력적 병행 프로세스

(1) 다른 프로세스들과의 협력을 통해서 기능을 수행

(2) 상호작용이 필요

(3) 동기화 필요 ( 교착상태, 임계영역 문제, 결과를 예측할 수 없는 상황 등 여러 문제들이 발생,)

3) 병행 처리의 문제점

(1) 한 기능을 공유해 수행하는 두 프로세스 간의 동기화 문제 ( 자원을 배타적으로 사용해야 함)

(2) 분산처리 시스템 ( 자료교환을 위한 메시지 전달 방식 등의 통신 문제)

(3) 실행 순서와는 무관하게 항상 같은 결과를 얻을 수 있어야 함.

(4) 교착상태 문제

(5) 프로그래밍 언어를 통한 병행 처리 문제

4) 임계구역

(1) 프로세스가 공유 자원을 접근하고 있는 동안의 상태

(2) 상호 배제를 보장

(3) 임계구역 문제 해결 3가지 요건

① 상호배제 (임계구역에는 오직 하나의 프로세스만,)

② 진행 (임계구역에 진입하려는 프로세스들이 있을 경우, 나머지 구역에서 실행 중이지 않은 프로세스들만 임계구역 진입 프로세스를 결정하는데 참여할 수 있으며, 이 선택은 무한 연기 되어서는 안 된다.)

③ 한정된 대기(진입 요청한 이후부터 그 요청이 허용될 때까지 다른 프로세스들의 임계구역 진입허용 횟수에 한계가 있어야 함) 기아현상을 예방.

(4) 1단계 알고리즘 상호배제 만족, 진행 불만족

while (turn !=i); //i의 차례가 아닐 동안 대기. 즉 누군가가 먼저 임계구역을 선점한 아이가 P(I)를

바꿔줘야 하는데, 아무도 없으면 무한정 대기.. 이게 다 순서가 정해져 있어서 그럼.
크리티컬 섹션
turn = j; //턴을 j로 넘겨줌

(5) 2단계 알고리즘

flag[i] = true ; //각 배열로 프로세스 상태 확인 , 1단계 알고리즘의 오류를 해소하기 위해 내가 바꾼다.
while(flag[]) //프로세스 j가 ture가 되어 들어갈 준비가 되었는지 확인. 즉 j님 지금 임계구역에 계심? 이라고 묻는 행위., 근데 문제는 때마침, j가 true를 해버리고 I눈치를 보면 서로 눈치보다가 끝남. 무한정 대기가 됨.

(6) 3단계 알고리즘

flag[i] = TRUE; //일단 나 들어갈 수 있음. 이라는 소리
turen = j; //j 너부터 들어가.
while ( flag[i] and turen == j) // j가 들어갈 준비가 된 상태에 j가 임계구역에 있는 두 조건이 만족하는 동안은 난 대기할게.

5) 하드웨어에 의한 동기화
   (1) 화장실 문 잠그고 들어가는 개념
   (2) 들어갈 때, TestAndSet(boolean & target) 함수로 target을 true로 바꿔주고 while문을 false로 탈출해 준다. 즉 처음에 target은 false여야 한다.
6) 세마포(프로세스가 n개 일 때)
   (1) 바쁜 대기

```
P(S)    {
                while (S ≤ 0) ;
                S-- ;
        }
```

```
V(S)    {
                S++ ;
        }
```

(2) 세마포를 이용한 상호배제

```
semaphore S=1;

while (1) {
        P(S) ;
        // 임계영역
        ...
        V(S) ;
    }
```

세마포를 이용한 동기화

semaphore sync = 0;

| wakeup시킬 프로세스 | block될 프로세스 |
|---|---|
| ..... | ..... |
| V(sync); | P(sync); |
| ..... | ..... |

세마포를 이용한 생산자 / 소비자 문제 ( 예 : 버퍼 )

- 공유변수

semaphore mutex = 1; 상호 배제를 위해.
semaphore empty = n; 버퍼에 비어있는 셀의 개수
Semaphore full = 0; 버퍼에 차있는 셀에 개수.

| 생산자 프로세스 | 소비자 프로세스 |
|---|---|
| while(1) {<br>// 데이터 생산<br>P(empty);<br>P(mutex);<br>// 생산된 데이터를 버퍼에 추가<br>V(mutex);<br>V(full);<br>} | while(1) {<br>P(full);<br>P(mutex);<br>// 버퍼에서 데이터 제거<br>V(mutex);<br>V(empty);<br>// 제거된 데이터 전송<br>} |

(3) 세마포를 이용한 읽기 / 쓰기 문제

① 여러 개의 읽기 프로세스가 동시에 공유 자료에 접근하는 것은 허용

② 읽기 프로세스와 쓰기 프로세스가 동시에 접근하는 것은 허용되지 않음

③ 여러 개의 쓰기 프로세스도 동시에 공유 자료에 접근할 수 없음

## 공유변수

```
semaphore wrt = 1;
semaphore mutex = 1;
int readcount = 0; 리드카운터가 0일때만 쓰기를 할 수 있도록.
```

| 쓰기 프로세스 | 읽기 프로세스 |
|---|---|
| | P(mutex);<br>readcount++;<br>if (readcount == 1) P(wrt);<br>V(mutex);<br>// 읽기 수행<br>P(mutex);<br>readcount--;<br>if (readcount == 0) V(wrt);<br>V(mutex); |
| P(wrt);<br>// 쓰기 실행<br>V(wrt); | |

15. 모니터 (세마포어와의 차이점은, 세마포는 동기화 함수코딩을 고려해야 하는 반면, 모니터는 프로시저 호출을 통해 간단히 해결이 가능. )

1) E.W.Dijkstra와 B.Hansen에 의해 제시되고, C. A. R. Hoare에 의해 개선된 모니터라는 개념

2) 순차적으로만 사용할 수 있는 특정 공유 자원이나 공유자원 그룹을 할당하는데 사용되며, 필요한 데이터 및 프로시듀어를 포함하는 병행성 구조.

3) 공유자원에 접근하고자 하는 프로세스는 반더시 특정 모니터의 진입부를 호출해야 함.
4) 이미 사용 중인 모니터에 들어가려고 하는 프로세스는 반드시 대기
5) 데이터와 이들 데이터를 처리하는 프로시듀어의 집합
6) 정보 은폐 = 모니터 내의 데이터는 모니터 내부에 의해서만 접근이 가능하므로 모니터 외부의 프로세스는 접근을 할 수 없음,
7) 신뢰성 높은 소프트웨어 개발을 가능하게 하는 시스템 구조화 기법

```
monitorname : monitor;
{
declarations of private data;  //local monitor variables 가변 모니터 지역
void pub name(formal parameters){ // public procedures
  procedure body;
}
void priv name() {
initialization of monitor data; //모니터 데이터 초기화
}
}
```

모니터를 이용한 간단한 자원할당 예

```
monitor resource_allocator
{
boolean resource_in_use; //사용중인가 아닌가의 확인.
condition resource_is_free; //사용중이지 않은 자원.
  void get_resource(int time){ // 공유자원 진입
    if(resource_in_use)  //만약 사용중이면
    resource_is_free.wait(time); // free가 ture가 될 때까지 기다리고.
  resource_in_use = true;  //사용할 때 true로 바꿔서 아무도 접근 못하도록 함.
  }
  void return_resouce(){  //자원을 사용 마치고 반납할 경우,
    resouce_in_use = false; //사용중을 false로 해주고
    resource_is_free.signal() // free 신호를 보냄
  }
  void init(){
    resource_in_use = false; // 초기화 소스코드
  }
```

환형 버퍼에서 생산자-소비자 문제에 대한 모니터 사용 예

```
void fill_a_slot(char x){ //x를 생산하는 함수
  if (slot_in_use == N wait (ring_buffer_has_space); // 슬롯이 꽉 차있으면, 링 버퍼가 비워질 때까지 대기.
  ring_buffer[next_slot_to_fill] = X; // 버퍼 배열에 다음 채워야 하는 부분에 X를 넣는다.
  slot_in_use = slot_in_use + 1; // 사용되고 있는 슬롯은 +1 시켜줘야 한다.
  next_slot_to_fill = (next_slot_fill + 1) mod N; 다음을 가리키기 위해 modular 연산을 해준다.
  signal(ring_buffer_has_data); //데이터가 있다는 걸 알린다.
}
void empty_a_slot(char x){ // 소비자 메소드
  if (slot_in_use == 0 ) wait(ring_buffer_has_data); //만약 슬롯이 없으면 채워질 때까지 기다려야 함
  x = ring_buffer[next_slot_to_empty]; // 비워져야 하는 슬롯에 x가 들어있다.
  slot_in_use = slot_in_use-1; // 사용되고 있는 술롯을 줄여준다.
  next_slot_to_empty = (next_slot_to_empty +1 )mod N; // 비워져야 하는 슬롯은 1칸 옆으로 이동해 모듀러를 해준다.
  signal(ring_buffer_has_space); // 링 버퍼에 스페이스가 있다는 시그널을 보낸다.
}
{
  slot_in_use = 0; //사용되고 있는 부분
  next_slot_to_fill = 0; // 다음 채워져야 하는 부분.
  next_slot_to_empty = 0; // 비워져야 하는 부분
}
```

읽기 쓰기 문제에 대한 모니터 사용 예

```
monitor readers_and_writers;
int readers; //읽고 있는 사람의 수.
boolean someone_is_writing; // 쓰고 있는 사람이 있는가 없는가.
condition reading_allowed, writing_allowed; // 읽기와 쓰기 권한의 상태
void begin_reading(){ //읽기 시작.
  if(someone_is_writing || queue(writing_allowed)) wait(reading_allowed );
 //누군가가 쓰고있거나 쓰려는 사람이 큐에 있을 때, 읽기권한이 올때까지 기다림.
  readers = readers +1; // 읽고있는 사람 카운트를 1 올려줌.
  signal(reading_allowed); //읽기 허가를 줌.
}
void finished_reading(){ //읽기가 끝났을 때,
  readers = readers-1; // 읽기 카운트를 1 내려줌.
  if(reader == 0) signal(writing_allowed); // 만약 읽고 있는 사람이 0일 때 쓰기권한을 보냄.
}
void begin_writing(){
  if(readers > 0 || someone_is_writing) wait(writing_allowed); // 만약 읽고 있는 사람이 있을 경우나
쓰고 있는 사람이 있을 경우, 쓰기권한을 기다림.
  someone_is_writing = true; //누군가가 쓰고있음.
}
void finished_writing(){  //쓰기가 끝났을 경우
someone_is_writing = false; // 아무도 쓰고 있지 않음.
if (queue(reading_allowed)) signal(reading_allowed; // 만약에 큐에 읽기를 기다리고 있으면 읽기 권한
을 보냄.
else signal(writing_allowed); //그게 아니면 쓰기 권한을 계속 줌.
}
{
  readers = 0; //초기화 수치.
  someone_is_wrting = false;
}
```

16. 메시지
    1) 송신측 프로세스와 수신측 프로세스 간에 교환될 수 있는 정보의 집합 (많은 다중 프로그래밍 운영체제들이 몇 가지 종류의 프로세스 간 통신을 지원하기 위하여 메시지 메커니즘을 채택
    2) 메시지 형식 : 송, 수신측 식별자, 형식, 길이, 데이터


3) 고려사항

(1) 네이밍 문제

① 메시지 교환에 관계되는 해당 프로세스들이 상대 프로세스를 설정하는 방법을 결정하는 것.

(2) 직접 네이밍

① 송신자가 특정 수신자를  설정해야 하고, 반대로 메시지 수신을 희망하는 수신자는 송신자를 설정

② 일대일 통신 관계로 인하여 보다 안전하고 확실한 통신이 가능.

```
process A;
..
send(B, message);
..
process B;
..
receive(A, message);
..
```

(3) 간접 네이밍

① 메시지가 특정 목적만을 위하여 사용되는 영역에 송신되고 또 그 영억으로부터 수신(영역끼리의 송 수신)

② 특정 역역을 우편함이라 함

③ hems 프로세스 간의 메시지 교환은 이 우편함을 통함

④ 송신자와 수신자 간에 일대일, 일대 다수, 다수 대 일, 다수 대 다수의 통신 관계가 가능하다는 점에서 매우 유용

```
process A;
..
send(mailbox1, message);
..
process B;
..
receive(mailbox1, message);
..
```

(4) 복사 문제
① 메시지를 수신자 주소공간에 복사하여 두거나, 또는 두 프로세스 간의 메시지에 대한 포인터를 전달함으로써 해결.
② 한 프로세스에서 오류가 발생하더라도 이로 인해 메시지의 지역 복사본만이 손상.
③ 메시지 복사 과정에서 동적인 기억장치 풀(버퍼, 스풀링)

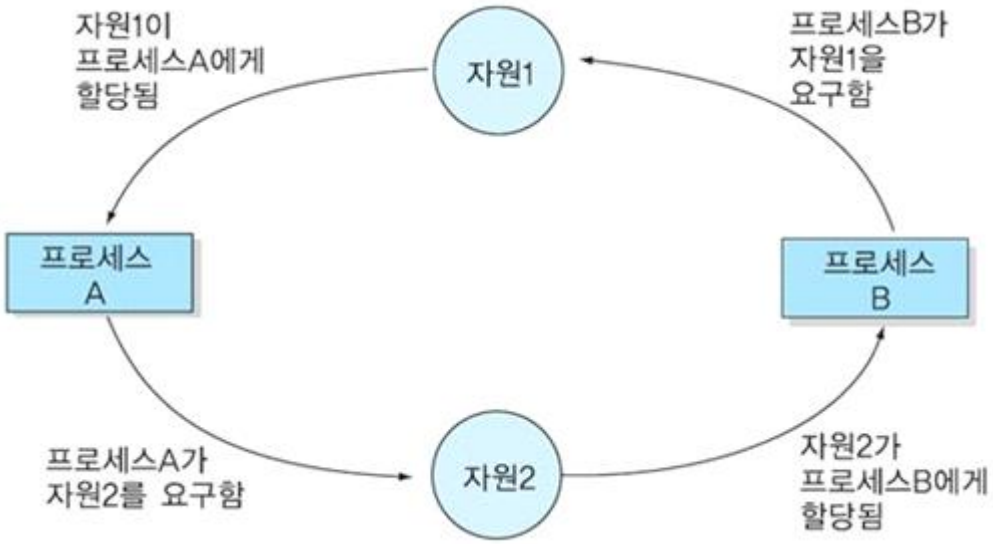
(5) 버퍼링 문제
① 수신자가 준비되어 있지 않을 경우 버퍼에 저장
② 비동기적 통신을 가능하게 하기 위해서 버퍼링 필요
③ 수신자 상황에 관계없이 자신의 실행을 계속 할 수 있는 과정을 set and forget 운영이라 하며, 이는 시스템 병행성을 향상

(6) 길이 문제
① 메시지의 길이를 고정적으로 할 것인지 또는 가변적으로 할 것인지를 결정
② 고정적 길이의 메시지 - 구현이 용이하나 버퍼의 크기에 비하여 메시지가 작을 경우 버퍼의 낭비를 초래.
③ 가변길이의 메시지 - 동적으로 버퍼를 생성하는 것으로 구현이 복잡하기는 하나, 높은 적용력을 가짐.

17. 교착상태
1) 하나 또는, 그 이상의 프로세스가 발생될 수 없는 어떤 특정 사건(event)을 기다리고 있는 상태(자원을 기다리고 있는 상태), 특정 프로세스가 특정 자원을 위하여 무한정 기다려도 도저히 해결할 수 없는 상태.
2) 시스템의 효율을 급격히 떨어뜨리는 문제점 발생
3) 정의 - 서로 강의 반대방향으로 향하는 두 사람이 징검다리를 건너면서 같은 돌을 디디려 할 때 발생하는 문제
4) 프로토콜을 이용하여 해결
(1) 반대편에서 강을 건너는 사람의 유무를 확인
(2) 두 사람이 동시에 출발해도 교착 상태가 발생하고, 두사람이 다 기다려도 교착상태가 발생
(3) 하나 이상의 프로세스가 강을 건너기 위해 무작정 기다리는 경우 기아상태라고 함.
5) 정상적인 프로세스 자원 이용 순서

(1) 요청 (다른 프로세스에 의하여 자원 사용 중일 때, 요청한 자원을 얻을 수 있을 때까지 기다려야 함.)
(2) 사용 (프로세스가 자원을 얻어 작동)
(3) 해제 (자원이 미리 요구되어 있거나 할당되어 있으면, 반납해야 함)

6) 순환대기 모델




7) 무한대기 - 교착상태와 유사한 문제
(1) 시스템에 의해 어떤 프로세스가 처리되는 동안 다른 프로세스의 스케줄링은 끝없이 연기될 가능성. 우선순위가 낮은 프로세스가 무한정 기다리는 현상
(2) 해결책 - 에이징 - 기다리는 시간에 비례하여 프로세스에게 우선순위를 부여하는 방법

8) 교착상태 조건
(1) 상호 배제
① 자원에 대해 배타적인 사용
② 즉 하나의 자원은 반드시 비공유 되는 상태에서 점유되어야 함
(2) 점유와 대기
① 적어도 하나의 자원을 점유하면서, 다른 프로세스에 의해 점유된 다른 자원을 요구하고 할당 받기를 기다릴 경우
(3) 비선점
① 작업의 수행이 끝날 때까지 해당 자원을 반환하지 않음
(4) 환형대기
① 각 프로세스는 환형 내의 이전 프로세스가 요청하는 자원을 점유와 요청

9) 자원할당 그래프 ppt참조 (그릴 수 있어야 함)

10) 교착상태 예방
(1) 하벤더 방안
① 사전에 교착상태 발생 가능성을 없애는 방안

② 상호 배제 조건을 부정하지는 않음 ->교착 상태가 절대 발생하지 않음

(2) 점유와 대기조건 방지

① 각 프로세스는 필요한 자원들을 모두 한꺼번에 신청

② 요구한 자원을 전부 할당하여 주거나 또는 하나라도 부족하면 할당하지 않는 방식

③ 문제점 - 자원낭비 초래(처음에 조금만 사용하고 오래 시간 후 나머지 자원을 사용할 경우, 자원 요구가 많을 경우.) 한번에 많은 자원을 제공할 수 없는 경우에는 무한 연기의 원인이 됨.

(3) 비선점 조건 방지

① 추가 자원에 대한 요구가 거절당했다면 그 프로세스는 자원을 전부 반납

② 문제점 - 도중에 할당해 줄 수 없는 경우 프로세스를 중지, 조금 수행하다 다시 처음으로 돌아가는 경우가 있음, 무한 연기, 시스템 성능 저하

(4) 환형 대기조건 방지

① 모든 자원에 고유 번호를 부여

② 현재 가지고 있는 자원보다 더 큰 번호를 가진 자원만을 요청하도록 제한

③ 문제점 - 자원이 늘어나면 고유번호를 다시 세팅. 예상순서와 다르게 자원을 요구할 경우, 사용하기 전부터 자원을 점유하고 있어야 함으로 비용증가와 성능 감소,

11) 안전상태

(1) 시스템이 어떤 순서로든 요청하는 모든 자원을 교착 상태를 야기하지 않고 모두 할당해 줄 수 있다는 뜻.

(2) 불안전 상태 - 모든 프로세스를 무사히 마칠 수 있는 순서를 찾을 수 없으면, 불안전 하다고 한다.

12) 은행가 알고리즘 (N : 프로세스의 수, M 자원 종류의 수, 일단 다 할당해보는 방식

(1) 잔여량 - 각 종류의 자원이 현재 몇 개 사용 가능한지를 나타내는 백터, 크기는 M

(2) 최대 요구량 - 각 프로세스가 최대로 필요로 하는 자원의 개수를 나타내는 행렬 N * M

(3) 할당량 = 각 프로세스에게 현재 할당되어 있는 자원의 개수 행렬 n*m

(4) 필요량 = 각 프로세스가 향후 요구할 수 있는 자원의 개수 n * m

$Need\ i,j = k$ 라면, $P\ i$ 가 향후 $R\ j$ 를 $k$ 개까지 더 요구할 수 있음을 뜻함

$Need\ i,j = Max\ i,j - Allocation\ i,j$

13) 교착상태 회피

(1) 할당량과 필요량을 백터로 취급

(2) 안전 알고리즘

```
work 와 finish를 각 각 크기가 M,N인 백터라고 가정
work = available, finish[i] = false (i = 1,2,...,n)으로 초기화
finish[i] == false
needi <= work
work = work + allocationi
finish[i]= true
```

(3) 은행가 알고리즘의 단점

① 사용 가능한 일정량의 자원이 미리 존재하도록 요구됨

② 남아있는 사용자의 수가 수시로 변경됨
③ 모든 요청에 대하여 주어진 시간 내에서 자원을 할당할 수 있도록 더 강력한 보장이 필요.
④ 사전에 자기가 필요한 자원의 최대량을 제시해야 함.

14) 교착 상태 탐지
(1) 탐지 알고리즘

```
finish[i] == false
request <= work //끝나지 않았고, 요구량이 잔여량 보다 적은 경우
아니면, 교착상태에 빠짐
```

(2)

15) 교착 상태 회복
(1) 교착상태 회복 알고리즘
① 모든 프로세스 중지
② (희생자 선정)하나를 중지하고, 교착 상태 탐지를해봄. - 오버해드 발생 위험.
③ 문제점 - 교착상태인지 알기 어려움,
④ 프로세스 우선순위가 없으며, 오퍼레이터가 임의 결정을 내려야 함
(2) 희생자 선정 기준
① 프로세스들의 우선순위
② 연산한 기간, 완료하기 위한 시간
③ 자원의 사용량
④ 몇 개를 희생시킬 것인가
(3) 복귀문제
① 어느정도의 시간을 복귀시켜야 하는가.
(4) 기아현상 문제
① 반복적으로 희생자로 선정될 수 있음.
② 상한선을 둬야함.

18. 정보 보호
1) 보호란 - 사용자 접근을 제어하는 것
2) 보안이란 - 자료의 결함이 없이 보전되는 것
3) 정책 - 무엇을 수행할 것인가를 결정
4) 기법 - 어떻게 수행할 것인가를 결정
5) need to know 원칙
(1) 프로셋들은 접근 권한이 부여된 자원들에 대해서만 접근
(2) 프로세스는 일을 완료하기 위해서 자신이 필요로 하는 자원만을 접근
6) 접근 권한
(1) <객체 이름, 권한 집합>의 구성
7) 보호영역
(1) 어떤 프로세스가 시스템의 여러 가지 객체들에 대해 가지고 있는 접근 권한을 정의한 것.

8) 접근제어의 목적
   (1) 컴퓨터 시스템을 이루고 있는 컴퓨팅 자원, 통신 자원 및 정보자원 등에 대하여 허가되지 않은 접근을 방어
9) 허가되지 않은 접근이란,
   (1) 불법적인 자원의 사용, 노출, 수정, 파괴와 불법적인 명령어의 발행을 포함.
10) 각 자원에 대한 기밀성(비밀성) (권한이 없는 프로세스는 읽을 수 없도록 하는 것), 무결성(쓰기, 수정 금지), 가용성(실행, 접근권한이 있을 때만 사용 할 수 있도록) 및 합법적인 이용과 같은 보호서비스에 직접적으로 기여.
11) 접근 제어 기법 종류
   (1) ACL - 접근 행렬의 열을 표현
   (2) CL - 접근 행렬에 행을 표현
   (3) SL - 제어대상에 레이블을 붙이는
12) 접근 행렬의 구현
   (1) 접근 행렬을 구현하려면 전역 테이블을 써야함. 영역 객체 권한 집합을 적어줘야 함.
13) 접근 제어 리스트 (직접 접근이 불가하며, 오로지 운영체제를 통해서만 접근이 가능)
   (1) <사용자명 사용자 그룹명> *는 모든 사용자 또는 모든 그룹으로 영역을 나눔.
   (2) 단점 - 접근 제어 리스트의 변경이 현재 객체를 사용하고 있는 사용자에게 영향을 미치지 못하는 점.
   (3) 권한 리스트. - 각 행을 해당 영역과 결합한 것. <객체, 권한>
   (4)