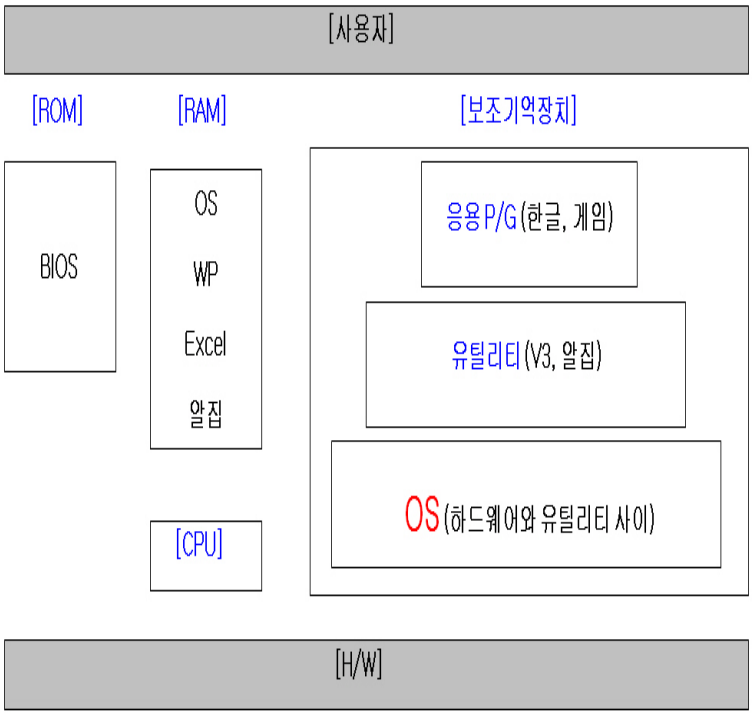
3과목 : 운영체제

Check 1. 운영체제 개념 [출제빈도: 中]

1. 운영체제 이해하기(operation System)

-컴퓨터 시스템 자원을 효율적으로 관리하고 사용자가 편리하게 사용하는 환경을 제공하는 S/W

* 컴퓨터 부팅 순서



2. 운영제체의 정의, 목적, 기능, 역할 ★★★★☆

- 사용자와 컴퓨터간의 인터페이스를 제공하는 소프트웨어
- 자원의 효율적인 스케줄링
  (프로세서, 기억장치, 주변장치, 파일 관리)
- 데이터 공유 및 주변장치 관리
- 처리 능력(Throughput), 신뢰도, 사용 가능도 향상
- 응답시간, 반환시간(Turn Around Time) 단축
- 입/출력 장치와 사용자 프로그램을 제어
- 스스로 어떤 유용한 기능도 수행하지 않고 다른 응용 프로그램이 유용한 작업을 할 수 있도록 환경 마련
- 시스템 사용 도중 발생하는 내부, 외부적인 오류로부터 시스템을 보호
- 컴퓨터 자원들인 기억장치, 프로세서, 파일 및 정보, 네트워크 및 보호 등을 효율적으로 관리할 수 있는 프로그램의 집합
- 오류 검사 및 복구 기능
- 컴퓨터를 초기화시켜 작업(JOB)을 수행할 수 있는 상태로 유지시키는 역할
- 운영체제 이외의 프로그램들은 운영체제가 제공한 기능에 의존하여 컴퓨터 시스템의 자원에 접근
- 응용 프로그램 유지보수(X)
- 실행 가능한 목적(object) 프로그램 생성(X)
  -> 컴파일러 인터프리터
- 한 가지 기종의 시스템에 전문적인 기능을 가지도록 설계 (X)

2. 운영체제가 자원들을 관리하는 과정 ★☆☆☆☆

1) 시스템 내 모든 자원들의 상태 파악
2) 어떤 프로세스에게 언제, 어떤 자원을 할당할 것인가를 결정하는 분배 정책 수립
3) 자원을 배당하고 운영함으로써 수립된 정책을 수행
4) 프로세스에 배당된 자원 회수

3. 운영체제 계층 구조 (=관리작업) ★☆☆☆☆

하드웨어-CPU관리-기억장치관리-프로세스관리-주변장치관리-파일시스템관리-사용자프로세스

4. 운영체제종류 : MS-DOS, MS-Windows, UNIX, Linux

5. 운영체제 기능적 분류-제어 프로그램 ★☆☆☆☆

-시스템 전체의 작동 상태 감시, 작업의 순서 지정, 작업에 사용되는 데이터 관리 등의 역할을 수행하는 P/G

1) 감시 프로그램 (Supervisor Program)
2) 작업 제어 프로그램 (Job Control Program)
: 어떤 업무를 처리하고 다른 업무로의 이행을 자동적으로 수행하기 위한 준비 및 그 처리 완료를 담당하는 기능을 수행한다. 즉, 작업의 연속 처리를 위한 스케줄 및 시스템 자원 할당 등을 담당한다.
3) 데이터 관리 프로그램 (Data Management Program)
: 주기억장치와 보조기억장치 사이의 자료 전송, 파일의 조작 및 처리, 입/출력 자료와 프로그램간의 논리적 연결 등, 시스템에서 취급하는 파일과 데이터를 표준적인 방법으로 처리할 수 있도록 관리

6. 운영체제 기능적 분류-처리 프로그램 ★☆☆☆☆

-제어 프로그램의 지시를 받아 사용자가 요구한 문제를 해결하기 위한 프로그램

1) 서비스 프로그램 (Service Program)
: 효율성을 위해 사용 빈도가 높은 P/G
2) 문제 프로그램 (Problem Program)
: 특정 업무 해결을 위해 사용자가 작성한 P/G
3) 언어 번역 프로그램 (Language Translator Program)
: 어셈블러, 컴파일러, 인터프리터

*주의: 언어 번역 프로그램은 선택적이다.

7. 운영체제 세대별 발달 과정 ★★★☆☆

1세대) 일괄 처리 시스템 (Batch Processing System)
-> 가장 먼저 생겨난 방식
: 유사한 성격의 작업을 한꺼번에 모아서 처리

2세대) 다중 프로그래밍 (Multi Programming)
-> 처리량의 극대화
: 한 대 컴퓨터, 여러 프로그램들 실행

2세대) 시분할 시스템 (Time Sharing System)
-> 응답시간의 최소화
: 여러 명의 사용자가 사용하는 시스템에서 컴퓨터가 사용자들의 프로그램을 번갈아 가며 처리해 줌으로서 각 사용자가 각자 독립된 컴퓨터를 사용하는 느낌을 주는 시스템

2세대) 다중 프로세싱 (Multi Processing)
: 한 대의 컴퓨터에 중앙처리장치(CPU)가 2개 이상 설치, 여러 프로그램들 실행

2세대) 실시간 시스템 (Real-Time System)
: 한정된 시간 제약조건에서 자료를 분석하여 처리 (ex. 비행기 제어 시스템, 교통 제어)

3세대) 다중 모드(mode) 시스템
: 1, 2 세대 혼합 시스템

4세대) 분산 처리 시스템 (Distributed Processing System)
: 여러 대의 컴퓨터들에 의해 작업들을 나누어 처리, 그 내용이나 결과를 통신망을 이용하여 상호 교환

Check 2. 시스템소프트웨어 종료 [출제빈도: 中]

1. S/W 분류

1) 시스템 S/W : 제어 프로그램, 처리 프로그램
2) 유틸리티 : 압축, 디스크 관리, 백신 등 프로그램
3) 응용 S/W : OA용, 통신용, 그래픽, 멀티미디어 등 프로그램

2. 시스템 소프트웨어 ★☆☆☆☆

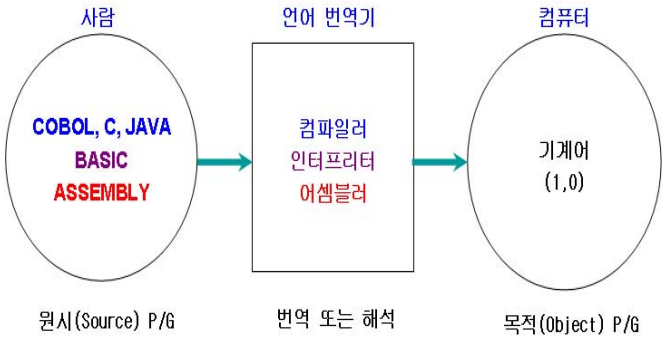
1) 정의: 시스템 전체를 작동시키는 프로그램
(시스템 소프트웨어=운영체제+언어번역P/G 등)
2) 종류

- 언어 번역 P/G : 어셈블러, 컴파일러, 인터프리터
- 메모리에 프로그램을 적재 P/G : 로더
- 반복되는 부분을 한 개의 이름으로 묶어 사용 : 매크로 프로세서

3. 언어 번역 및 실행 과정

- 원시(source)p/g→번역→목적(object)p/g생성→Link→Lode→실행

4. 컴파일러, 인터프리터



구분		컴파일러	인터프리터
공통점		고급언어→기계어	
차이점	번역단위	전체를 번역	줄 단위로 번역
	목적p/g 생성여부	생성 O	생성 X
	실행속도	빠르다	느리다.

5. 어셈블러 ★☆☆☆☆

-어셈블리어(저급언어)로 작성된 P/G을 기계어로 번역해 주는 프로그램

1) 번역방식
-1Pass: 신속하지만 어렵다.
-2Pass: 느리지만 쉽다. → 프로그램 작성이 용이

2) 두 개의 패스(Pass)로 구성하는 이유
-한 개의 패스만을 사용하면 기호를 모두 정의한 뒤에 해당기호를 사용해야만 하기 때문
-기호를 정의하기 전에 사용할 수 있어 프로그램 작성이 용이하기 때문에

6. 로더 ★★★☆☆

-목적 P/G을 주기억 장치에 적재하여 실행 가능하도록 해주는 시스템 프로그램

1) 기능
: 할당(Allocation), 연결(Link), 재배치(Relocation), 적재(Load)
- Compile (X)

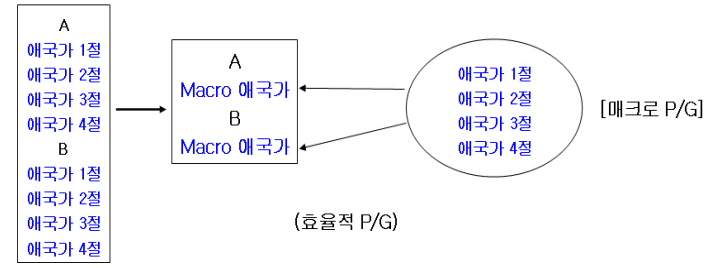
2) 종류

- **Compile-and-Go 로더**
: 번역기가 로더의 역할까지 담당 (번역+로더)
- **절대(Absolute) 로더**
: 적재 기능만 하는 간단한 로더
(할당,연결-프로그래머, 재배치-언어번역기)
- **직접 연결 로더(Direct linking loader)**
: 가장 일반적 (전체 기능)

3) 링커(Linkage Editor, 연결 편집기)
: 목적프로그램 등을 연결하여 실행 가능한 로드 모듈을 만드는 프로그램

7. 매크로 프로세서 ★★☆☆☆

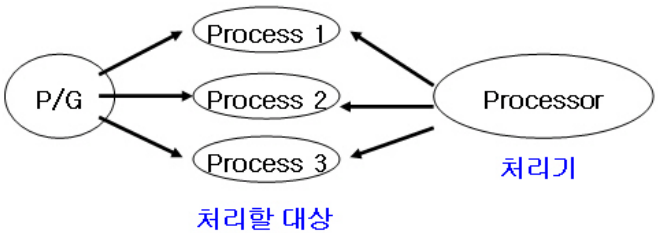
1) 처리 과정
: 매크로 정의 **인식** → 매크로 정의 **저장** → 매크로 호출 **인식** → 매크로 호출 **확장**
2) 특징 : 매크로 내에 매크로를 정의할 수 있다.
3) 매크로(개방형)와 부프로그램(폐쇄형) 차이점 : 매크로 내용 삽입 → M/M 절약 X, 실행 빠르다.



Check 3. 프로세스 개요 [출제빈도: 中]

1. 프로세스(Process) 정의 ★★☆☆☆

- 주기억장치에 저장된 프로그램(실행중인 프로그램)
- 운영체제가 관리하는 최소 단위의 작업
- 비동기적(비연속적) 행위를 일으키는 주체
- 프로시저(프로그램 일부)의 활동
- PCB를 가진 프로그램
- 프로세서가 할당되는 실체
- CPU에 의해 수행되는 사용자 및 시스템 프로그램
- 프로세스가 자원을 이용하는 정상적인 작동의 순서
  : 요청 -> 사용 -> 해제
- 지정된 결과를 얻기 위한 일련의 동작
- **디스크(보조기억장치)에 저장된 프로그램 (X)**
- **하드웨어에 의해 사용되는 입/출력 장치 (X)**



2. 스래드(Thread) 정의 ★☆☆☆☆

- 프로세스를 분할하여 운영체제의 성능을 개선하려는 소프트 웨어적 접근 방법
- 하나의 프로세스 내에서 병행성을 증대시키기 위한 기법
- 스레드는 동일 프로세스 환경에서 서로 독립적인 다중 수행이 가능하다
- 프로세스의 생성이나 문맥 교환 등의 오버헤드를 줄여 운영체제의 성능이 개선 된다.

3. 프로세스 제어 블록: PCB(Process Control Block) ★☆☆☆☆

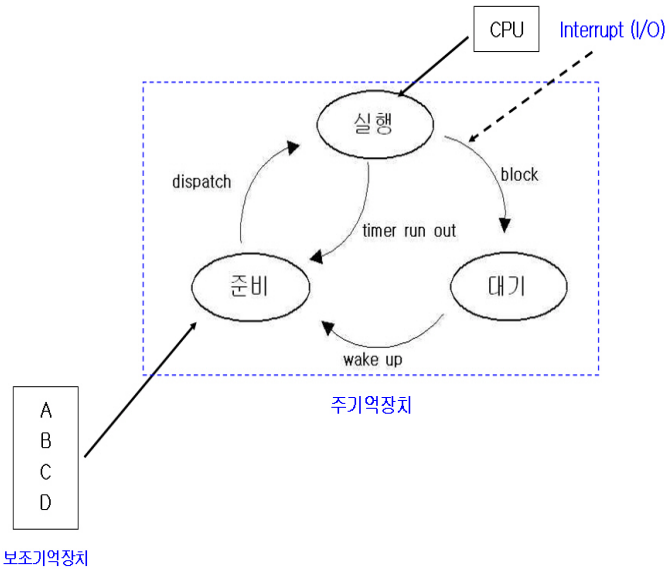
- 운영체제가 프로세스에 대한 중요한 정보를 저장해 놓은 곳 (프로세스 정보 리스트)
- 각 프로세스가 생성될 때마다 PCB가 생성되고, 완료되면 PCB는 제거
- O/S에게 Process에 대한 정보를 제공해 주는 자료구조 테이블
- 부모 프로세스와 자식 프로세스는 PCB를 공유 (X)

4. 프로세서 제어 블록- 저장정보 ★★★☆☆

- **프로세스의 현 상태**
- **프로세스의 우선 순위**
- **프로세스 식별자**
- 레지스터 저장 장소
- 할당된 자원에 대한 포인터
- 관련 레지스터 정보

- 프로세스의 사용 빈도 (X),
  할당되지 않은 주변장치의 상태 정보 (X)
- 모든 프로세스의 상태에 대한 조사와 통제 정보 (X)
- 파일할당 테이블(FAT) (X), 우선 순위를 위한 스케줄러 (X)
- **페이지 부재(page fault) 발생 횟수 (X)**
- 프로세스 오류의 수정 방법 (X)
- **프로세스의 CPU 사용율 (X)**, 프로세스의 처리기 종류 (X)
- 초기값 정보 (X), **프로세스의 크기 (X)**

5. 프로세스 상태 전이도 ★★☆☆☆

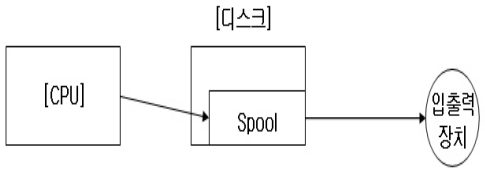


- **실행 상태(Run)** : 프로세스가 CPU를 차지하여 실행 중인 상태
- 타이머 런 아웃
  : CPU를 할당 받아 실행중인 프로세스가 할당시간을 초과하면 CPU를 다른 프로세스에게 양도하고 자신은 준비상태로 전이되는 것
- **준비 상태(Ready)**
  : 프로세스가 CPU를 사용하여 실행될 수 있는 상태
- **대기 상태(Wait, Block)** : 어떤 사건이 발생하기를 기다리는 상태
- 블록
  : 실행 중인 프로세스가 지정된 시간 이전에 다른 작업을 위해 스스로 프로세서를 양도하고 대기 상태로 전이되는 것

6. 스풀링(spooling) ★★☆☆☆

- 다중프로그래밍 환경 하에서 용량이 크고 신속한 액세스가 가능한 디스크를 이용하여 각 사용자 프로그램의 출력할 데이터를 직접

프린터로 보내지 않고 디스크에 모았다가 나중에 한꺼번에 출력함으로써 프린터 장치의 공유 및 프린터 처리 속도를 보완하는 기법
- 어떤 작업의 입/출력과 다른 작업의 계산을 병행 처리하는 기법
- 스풀링은 디스크 일부를 매우 큰 버퍼처럼 사용하는 방법



	스풀링	버퍼링
공통점	저속의 입출력 장치와 고속의 CPU간의 속도차이를 해소하기 위해서 나온 방법	
차이점	디스크	주기억장치

7. 인터럽트 ★★★☆☆

1) 프로그램 검사(Program Check) 인터럽트

: 수행 중인 프로그램에서 0으로 나누는 연산이나 허용되지 않는 명령어의 수행, 스택의 오버플로우(overflow)등과 같은 잘못이 있을 때 발생

2) SVC(Super Visor Call) 인터럽트 = 감시자(운영체제) 호출

- 프로세서에게 컴퓨터 제어권을 운영체제 수퍼바이저 프로그램에 넘길 것을 지시
: 입/출력 수행, 기억 장치 할당, 오퍼레이터와의 대화 등을 위하여 발생

3) 기계 검사 인터럽트

: 컴퓨터 자체 내의 기계적인 장애나 오류로 인하여 발생

4) 외부 인터럽트

: 시스템 타이머에서 일정한 시간이 만료된 경우나 오퍼레이터가 콘솔상의 인터럽트 키를 입력한 경우 발생

Check 4. 프로세스 스케줄링 [출제빈도: 上]

1. 프로세스 스케줄링(=CPU 스케줄링) ★★☆☆☆

- 정의
: 컴퓨터 시스템의 성능을 높이기 위해 그 사용 순서를 결정하기 위한 정책
- 목적(성능 평가)
: 처리율 증가, CPU 이용률 증가, 우선 순위 제 도, 오버헤드(부하) 최소화, 응답 시간 / 반환 시간 / 대기 시간 최소화, 균형 있는 자원의 사용, 무한 연기 회피

2. 프로세스 스케줄링 기법 ★★★☆☆

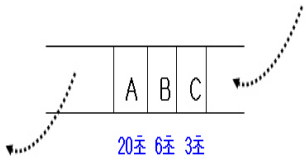
1) **비**선점 스케줄링 (Non Preemptive) : 비효율적, 비양보
- 프로세스에게 이미 할당된 CPU를 강제로 빼앗을 수 없고, 사용이 끝날 때까지 기다려야 하는 방법
- 일괄 처리(오버헤드 발생 X), 실시간 처리가 안되므로 중요한 작업이 기다리는 경우 발생
- 대표적인 스케줄링 : FIFO, SJF, HRN

2) **선점** 스케줄링 (양보) : 효율적
- 우선 순위가 높은 다른 프로세스가 할당된 CPU를 강제로 빼앗을 수 있는 방법
- 실시간 처리, 대화식 시분할 처리(오버헤드 발생 O)
- 대표적인 스케줄링 : RR, SRT

Check 5. 프로세스 스케줄링- 비선점 [출제빈도: 上]

1. FIFO (First-In First-Out) = FCFS (First-Come First-Service) ★☆☆☆☆

-준비상태에서 도착한 순서에 따라 CPU 할당



실행시간	A(20초), B(6초), C(3초)	평균 실행시간 = 29/3
대기시간	A(0초), B(20초), C(26초)	평균 대기시간 = 46/3
반환시간	A(20초), B(26초), C(29초)	평균 반환시간 = 75/3

-> 평균 반환시간 = 평균 실행시간+평균 대기시간

2. SJF (Shortest Job First) ★★★☆☆

- 작업이 끝나기까지의 실행 시간 추정치가 가장 작은 작업을 먼저 실행
- FIFO 보다 평균 대기 시간이 작지만 긴 작업의 경우 FIFO 기법보다 더 크고 예측이 더욱 어렵다
- 작업 시간이 큰 경우 오랫동안 대기하여야 한다

실행시간 : A(20초), B(6초), C(3초) → 3초 6초 20초 [C B A]

실행시간	C(3초), B(6초), A(20초)	평균 실행시간 = 29/3
대기시간	C(0초), B(3초), A(9초)	평균 대기시간 = 12/3
반환시간	C(3초), B(9초), A(29초)	평균 반환시간 = 41/3

* 제출(도착)시간이 주어졌을 경우 : A(0초), B(1초), C(2초) → 20초 3초 6초 [A C B]

실행시간	A(20초), C(3초), B(6초)	평균 실행시간 = 29/3
대기시간	A(0초), C(20-2초), B(23-1초)	평균 대기시간 = 40/3
반환시간	A(20+0초), B(3+20-2초), C(6+23-1초)	평균 반환시간 = 69/3

-> 실행시간이 큰 작업은 무한연기(기근현상) 가능성 있음 -> 해결(Aging 기법):강제 우선순위 부여

3. HRN (Highest response ratio Next) ★★★★☆

- SJF 방식의 단점(긴 작업과 짧은 작업간의 지나친 불평등)을 보완하는 기법
- 우선순위 계산식 : (대기 시간+서비스 시간)/서비스 시간

기출) 우선 순위가 가장 높은 작업

작 업	대기시간	서비스시간
A	5	5
B	10	6
C	15	7
D	20	8

* 그 외 비선점 스케줄링 ★☆☆☆☆

- **우선순위** : 대기 큐에서 기다리는 각 프로세스마다 우선 순위를 부여하여 그 중 가장 높은 프로세스에게 먼저 CPU를 할당하는 기법

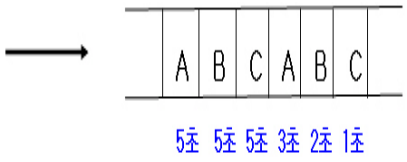
- **기한부** : 프로세스에게 일정한 시간을 주어 그 시간 안에 프로세스를 완료하도록 하는 기법

Check 6. 프로세스 스케줄링- 선점 [출제빈도: 上]

1. RR (Round Robin) ★★★★☆

- 대화식 시분할 시스템(Time Sharing System)을 위해 고안된 방식으로, FIFO 방식으로 선점형 기법
- 할당되는 시간이 클 경우 FCFS 기법과 같아지고, 할당되는 시간이 작을 경우 문맥교환 및 오버헤드가 자주 발생됨

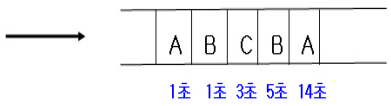
실행시간 : A(8초), B(7초), C(6초)
Time Slice(시간 할당량) : 5초



2. SRT (Shortest Remaining Time) ★★☆☆☆

- SJF 방식으로 선점형 기법, 현재 실행중인 프로세스의 남은 시간과 준비상태 큐에 새로 도착한 프로세스의 실행 시간을 비교하여 가장 짧은 실행 시간을 요구하는 프로세스에게 CPU를 할당하는 기법

작업	도착 시간	실행 시간
A	0	15
B	1	6
C	2	3



* 그 외 선점 스케줄링 ★☆☆☆☆

- **다단계 큐 (MQ, Multi level Queue)**
  : 프로세스들을 우선 순위에 따라 시스템 프로세스, 대화형 프로세스, 일괄처리 프로세스 등으로 상위, 중위, 하위 단계의 단계별 준비 큐를 배치하는 CPU 스케줄링 기법

- **다단계 피드백 큐 (MFQ, Multi level Feedback Queue)**
  : 여러 개의 큐를 두어 낮은 단계로 내려갈수록 프로세스의 시간 할당량을 크게 하는 프로세스 스케줄링 방식

Check 7. 프로세스 스케줄링- 기타 [출제빈도: 上]

1. 문맥교환 (Context switching) ★★★☆☆

: 다중 프로그래밍 시스템에서 운영체제에 의하여 중앙처리장치가 할당되는 프로세스를 변경하기 위하여 현재 중앙처리장치를 사용하여 실행되고 있는 프로세스의 상태 정보를 저장하고, 앞으로 실행될 프로세스의 상태 정보를 설정한 다음에 중앙처리장치를 할당하여 실행이 되도록 하는 작업을 의미하는 것

-> 운영체제에서 **overhead의 큰 요인 중 하나**

2. 노화(aging) 기법 ★★☆☆☆

: 자원이 할당되기를 오랜 시간 동안 기다린 프로세스에 대하여 기다린 시간에 비례하는 높은 우선 순위를 부여하여 가까운 시간 안에 자원이 할당되도록 하는 기법

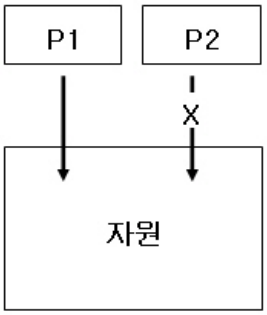
-> 우선 순위스케줄링에서 **무한 연기**를 방지하기 위한 기법

Check 8. 병행 프로세스와 교착상태 [출제빈도: 中]

1. 병행 프로세스(Concurrent) ★★☆☆☆

1) 정의
: 두 개 이상의 프로세스들이 동시에 존재하며 실행 상태에 있는 것

2) 병행 프로세스의 문제점
- 동시에 2개 이상의 프로세스를 병행 처리하면 한정된 자원(CPU, 메모리, 디스크, I/O 장치 등)에 대한 사용 순서 등 여러 가지 문제가 발생 할 수 있다.
  (다중 프로그래밍 기법 이용으로 병행성 문제 생김)
- ex) 사무실에서 공유 프린터 사용시 사용 순서에 따라 출력되지 않을 경우 출력물은 섞여진다.
- 문제 해결책 : 임계구역, 상호배제 기법, 동기화 기법



* 임계구역
  : 하나의 프로세스만 자원을 이용할 수 있도록 보호된 영역
* 상호배제 기법 : 임계구역을 지키기 위한 알고리즘(기법)
* 동기화 기법 : 상호배제의 원리를 보장하는데 사용

- 두 개 이상의 프로세스에 대해 특정 한 시점에 대해서 동시에 처리할 수 없으므로 각 프로세스에 대한 처리 순서를 결정하는 기법 (세마포어, 모니터)

2. 임계구역(Critical Section) ★☆☆☆☆

1) 정의
: 다중 프로그래밍 운영체제에서 한 순간에 여러 개의 프로세스에 의하여 공유되는 데이터 및 자원에 대하여, 한 순간에는 반드시 하나의 프로세스에 의해서만 자원 또는 데이터가 사용되도록 하고, 이러한 자원이 프로세스에 의하여 반납된 후 비로소 다른 프로세스에서 자원을 이용하거나 데이터를 접근할 수 있도록 지정된 영역
(하나의 프로세스만 자원을 이용할 수 있도록 보호된 영역)

2) 특징
- 특정 프로세스가 독점할 수 없다.
- 프로세스가 임계구역에 대한 진입을 요청하면 일정 시간 내에 진입을 허락해야 한다.
-ex) 사무실에서 공유 프린터(임계구역)으로 정해서 한 명의 사원(프로세스)이 독점해서 사용하지 않도록 할 수 있다.

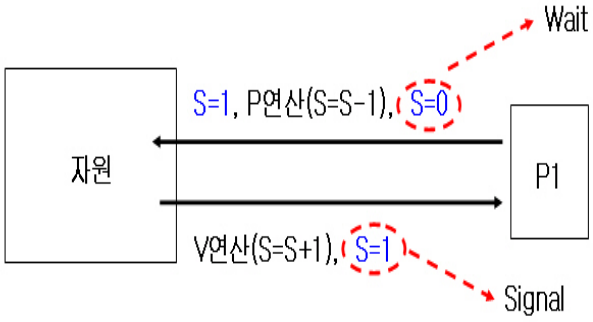
3. 상호 배제(Mutual Exclusion) ★★☆☆☆

1) 정의
: 한 프로세스가 공유 메모리 혹은 공유 파일을 사용하고 있을 때 다른 프로세스들이 사용하지 못하도록 배제시키는 제어 기법

2) 데커(Dekker) 알고리즘
- 교착상태가 발생하지 않음을 보장
- 공유 데이터에 대한 처리에 있어서 상호 배제를 보장
- 별도 특수 명령어 없이 순수하게 소프트웨어로 해결 된다.

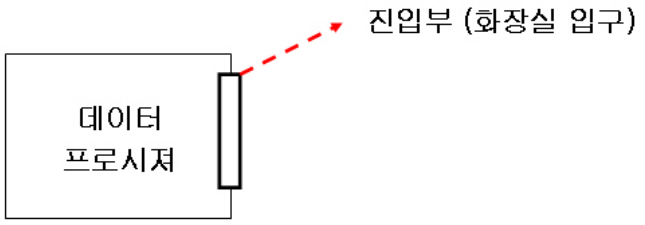
4. 동기화 기법 - 세마포어(Semaphore) ★★★☆☆

- 세마포어 : 신호기, 깃발
- 각 프로세스에 제어 신호를 전달하여 순서대로 작업을 수행하도록 하는 기법
- 다익스트라(Dijkstra) 가 제안
- P와 V라는 2개의 연산에 의해서 동기화를 유지시키고, 상호 배제의 원리를 보장
- P 연산은 임계 영역을 사용하려는 프로세스들의 진입여부를 결정하는 조작 (Wait 동작, S = S-1)
- V 연산은 블록 큐에 대기 중인 프로세스를 깨우는 신호(Wake Up) (Signal 동작, S = S+1)
- S는 P와 V 연산으로만 접근 가능한 세마포어 변수(제어신호)로, 공유 자원의 개수를 나타내며 0(사용중)과 1(사용가능) 혹은 0과 양의 값을 가질 수 있음



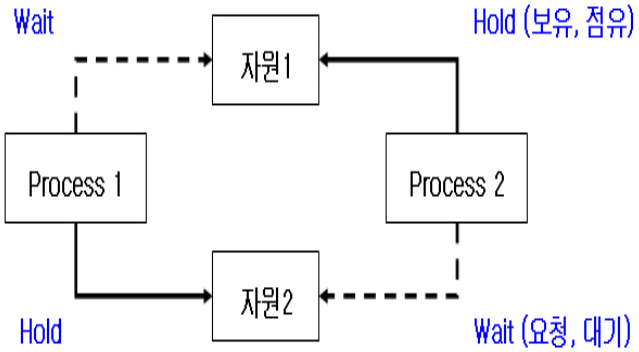
5. 동기화 기법 - 모니터(Monitor) ★★★☆☆

- 모니터 : 임계구역과 유사한 개념
- 동기화를 구현하기 위한 특수 프로그램 기법으로 특정 공유 자원을 프로세스에게 할당하는데 필요한 데이터와 이 데이터를 처리하는 프로시저로 구성됨
- 자료 추상화와 정보 은폐 개념을 기초로 하며 공유 자원을 할당하기 위한 병행성 구조
- 모니터 내의 공유 자원을 사용하려면 프로세스는 반드시 모니터의 진입부를 호출해야 함
- 외부의 프로세스는 직접 액세스할 수 없으며, 모니터의 경계에서 상호 배제가 시행됨
- 한 순간에 하나의 프로세스만 진입하여 자원을 사용할 수 있음
- 모니터에서 사용되는 연산은 Wait와 Signal이 있다.



6. 교착상태(Dead Look): 예측 못한 다운 ★★☆☆☆

: 상호 배제에 의해 나타나는 문제점으로, 둘 이상의 프로세스들이 자원을 점유한 상태에서 서로 다른 프로세스가 점유하고 있는 자원을 요구하며 무한정 기다리는 현상



7. 교착상태 발생 4가지 필요충분 조건 (동시만족 시 발생) ★★★★☆

1) 상호배제(Mutual Exclusion)
: 한 번에 한 개의 프로세스만이 공유 자원을 사용할 수 있어야 함

2) 점유와 대기(Hold & Wait)
: 최소한 하나의 자원을 점유하고 있으면서 다른 프로세스에 할당되어 사용되고 있는 자원을 추가로 점유하기 위해 대기하는 프로세스가 있어야 함

3) 비선점(nonpreemption)
: 프로세스에 할당된 자원은 사용이 끝날 때까지 강제로 빼앗을 수 없음

4) 환형대기(Circular Wait)
: 공유 자원과 공유 자원을 사용하기 위해 대기하는 프로세스들이 원형으로 구성되어 있어 자신에게 할당된 자원을 점유하면서 앞이나 뒤에 있는 프로세스의 자원을 요구해야 함

8. 교착상태 해결방안 ★★★☆☆

1) 예방 기법(Prevention)

: 교착 상태가 발생되지 않도록 사전에 시스템을 제어하는 방법으로, 교착 상태 발생의 4가지 조건 중에서 상호 배제를 제외한 어느 하나를 제거(부정)함으로써 수행됨

- 상호배제 부정 : 여러 프로세스가 공유 자원을 이용 (사용 X)
- 비선점 부정 : 선점
- 점유와 대기 부정
  : 프로세스가 실행되기 전 필요한 모든 자원을 점유하여 프로세스 대기를 없앰
- 환형대기 부정 : 자원을 선형 순서로 분류하여 각 프로세스는 현재 어느 한쪽 방향으로만 자원을 요구하도록 하는 것
- 해결방안 중 자원의 낭비가 가장 심함

2) 회피 기법(Avoidance)

: 교착 상태 해결 방안으로 발생 가능성을 인정하고 교착 상태가 발생하려고 할 때, 교착상태 가능성을 피해가는 방법, 주로 은행원 알고리즘(Banker's Algorithm)이 사용됨

* 은행원 알고리즘 ★★★☆☆

- 다익스트라(Dijkstra) 가 제안한 회피(Avoidance) 기법
- 각 프로세스에게 자원을 할당하여 교착 상태가 발생하지 않으며 모든 프로세스가 완료될 수 있는 상태를 안전 상태, 교착 상태가 발생할 수 있는 상태를 불안전 상태라고 함

[기출문제 풀이]

- 자원이 총 12개이고, 현재 할당된 양이 10개일 경우 아래 시스템을 안전 상태가 되기 위한 A, B ?

대출자 (Process)	대출된 금액 (현재 할당량)	대출 한도액 (최대 요구량)	추가 대출 요청액 (추가 요구량)
P1	2	5	3
P2	4	A	B
P3	4	8	4

→A=6, B=2로 할당되면 안전 상태를 유지할 수 있다.

3) 발견 기법(Detection)

: 시스템에 교착 상태가 발생했는지 점검하여 교착 상태에 있는 프로세스와 자원을 발견하는 것

4) 회복 기법(Recovery)

: 교착 상태를 일으킨 프로세스를 종료하거나 교착상태의 프로세스에 할당된 자원을 선점하여 프로세스나 자원을 회복하는 것

- Ctrl+Alt+Del => 작업 관리자 => 프로세스 끝내기

Check 9. 기억장치 관리기법-주기억장치 [출제빈도: 上]

1. 기억장치 관리 전략 ★

1) 반입(Fetch) 전략

- 보조기억장치의 프로그램이나 데이터를 언제 주기억장치로 적재할 것인지를 결정

2) 배치(Placement) 전략

- 주기억장치의 어디에 위치시킬 것인지를 결정

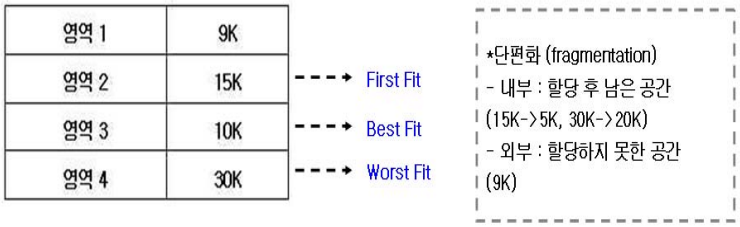
① 최초 적합(First Fit) : 첫 번째 배치시키는 방법 (속도↑, 공간↓)
② 최적 적합(Best Fit) : 단편화를 가장 작게 남기는 분할 영역에 배치시키는 방법 (속도↓, 공간↑)
③ 최악 적합(Worst Fit) : 단편화를 가장 많이 남기는 분할 영역에 배치시키는 방법 (속도↓, 공간↓)

3) 교체(Replacement) 전략

- 주기억장치의 모든 영역이 이미 사용중인 상태에서 주기억장치에 배치하려고 할 때, 이미 사용되고 있는 영역 중에서 어느 영역을 교체하여 사용할 것인지를 결정 (FIFO,OPT,LRU,LFU,NUR,SCR)

[기출문제]

First Fit, Best Fit, Worst Fit 방법에 대해서 10K 프로그램이 할당 부분? ★★☆☆☆



2. 단편화 해결 방법 ★★★☆☆

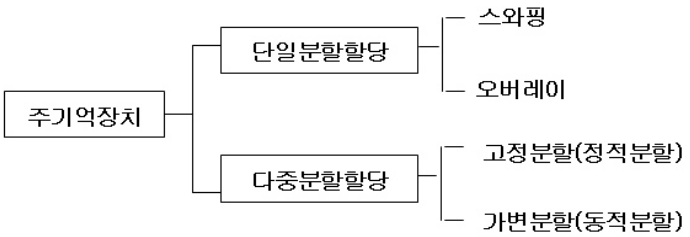
1) 통합(Coalescing) 기법

: 주기억장치 내에 인접해 있는 단편화된 공간을 하나의 공간으로 통합

2) 집약(Compaction) 기법, 압축, 쓰레기 수집(Garbage Collection)

: 주기억장치 내에 분산되어 있는 단편화된 빈 공간을 결합하여 하나의 큰 가용 공간을 만드는 작업

3. 주기억장치 할당 기법



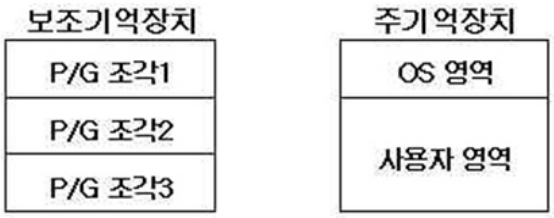
4. 단일분할 할당(단일 프로그래밍)

1) 스와핑(Swapping) ★☆☆☆☆

-하나의 프로그램 전체를 주기억장치에 할당하여 사용하다 필요에 따라 다른 프로그램과 교체하는 기법

2) 오버레이(Overlay) ★☆☆☆☆

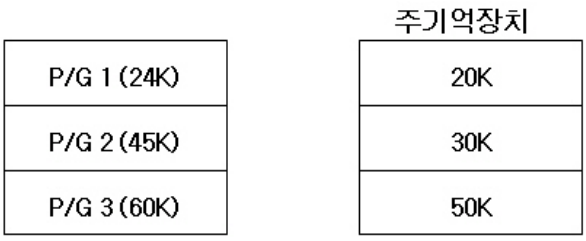
-실행되어야 할 작업의 크기가 커서 사용자 기억 공간에 수용될 수 없을 때 작업의 모든 부분들이 동시에 주기억 장소에 상주해 필요가 없다. 이 때 작업을 분할하여 필요한 부부만 교체하는 방법



5. 다중분할 할당(다중 프로그래밍)

1) 고정 분할 ★☆☆☆☆

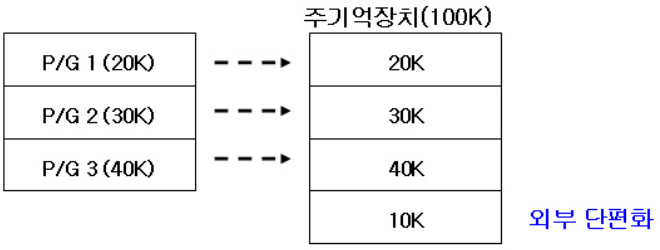
-주기억장치를 미리 몇 개의 고정된 개수와 크기의 부분으로 분할하여 여러 개의 여러개의 프로그램이 동시에 적재되어 실행되게 하는 방법



2) 가변 분할 ★☆☆☆☆

-고정 분할 할당 기법의 단편화를 줄이기 위한 것으로, 미리 주기억장치를 분할해 놓는 것이 아니라 프로그램을 주기억장치에 적재하면서 필요한 만큼의 크기로 영역을 분할하는 기법

ex) 식당 전체 공간에서 칸막이를 이용해서 손님의 수에 따라 자리를 만들어 줌

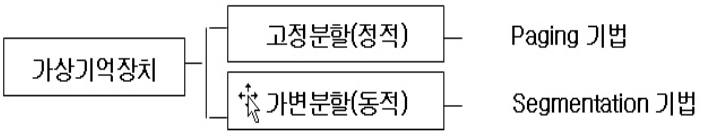


Check 10. 기억장치 관리기법-가상기억장치 [출제빈도: 上]

1. 가상기억장치 개요 ★☆☆☆☆

- 보조기억장치의 일부분을 주기억장치처럼 사용하는 것
- 용량이 적은 주기억장치를 마치 큰 용량이 있는 것처럼 사용하는 것
- 프로그램을 여러 개의 작은 블록으로 나누어서, 프로그램 실행 시 요구되는 블록만 주기억장치에 불연속적으로 할당하여 처리
- 주기억장치보다 용량이 큰 프로그램 실행하기 위해 사용
- 가상기억장치에 저장된 프로그램을 실행하려면 가상기억장치의 주소를 주기억장치의 주소로 변환하는 작업이 필요 (매핑)

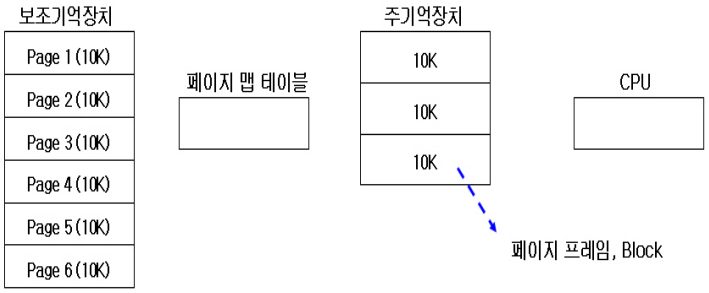
2. 가상기억장치 구현 기법



3. 페이징 (Paging) 기법 ★

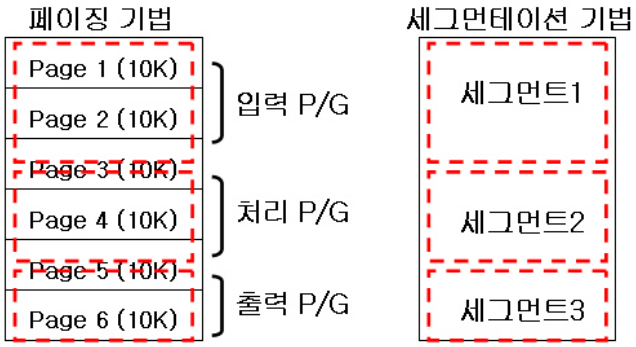
- 가상기억장치에 보관되어 있는 프로그램과 주기억장치의 영역을 동일한 크기로 나눈 후 나눠진 프로그램 (페이지)을 동일하게 나눠진 주기억장치의 영역(페이지 프레임)에 적재시켜 실행하는 기법
- 주소변환(Mapping) : 가상주소(보조기억장치)->실주소(주기억장치)
- 주소변환을 위해 페이지 맵핑 테이블(페이지 사상표)이 필요
  => 기억장소 낭비
- 페이지 부재(Page Fault)
  : P/G 실행시 참조한 페이지가 주기억장치에 없는 현상
- 외부단편화(X), 내부단편화(O)

* 페이지 크기가 작을 경우 (10K -> 1K)
- 페이지 수 증가 -> 페이지 맵핑 테이블 커진다
  -> 맵핑 속도 느리고 기억 공간 낭비 발생
- 디스크 접근 횟수 증가 →전체적인 입·출력 시간은 늘어남
- 한 개의 페이지를 주기억장치로 이동하는 시간이 줄어 듦
- 필요한 내용만 주기억장치에 적재
  →유용도 大 →기억장치 효율이 높아짐 (내부 단편화 감소)



4. 세그먼테이션(Segmentation) 기법 ★★☆☆☆

- 가상기억장치에 보관되어 있는 프로그램을 다양한 크기의 논리적인 단위로 나눈 후 주기억장치에 적재시켜 실행시키는 기법
  => 메모리 절약
- 논리적인 크기로 나눈 단위를 세그먼트라고 하며, 각 세그먼트는 고유한 이름과 크기를 갖고 있음
- 다른 세그먼트에게 할당된 영역을 침범할 수 없으며, 이를 위해 기억장치 보호키(Storage Protection Key)가 필요함
- 외부단편화(O), 내부단편화(X)



5. 가상기억장치의 성능에 영향을 미치는 요인 ★

1) 워킹 셋 (Working Set)

- 프로세스가 일정 시간 동안 자주 참조하는 페이지들의 집합으로, 자주 참조되는 워킹 셋을 주기억장치에 상주시킴으로써 페이지 부재 및 페이지 교체 현상을 줄임

2) 스래싱 (Thrashing)

- 프로세스의 처리 시간보다 페이지 교체 시간이 더 많아지는 현상
  -> CPU 이용률 저하
- 페이지 오류율(page fault)이 크면 스래싱이 많이 일어난 것이다
- 다중 프로그래밍의 정도가 높을수록 스래싱의 발생 빈도는 높아진다
- 스래싱 방지 방법 : 다중 프로그래밍의 정도를 줄인다, CPU 이용률을 높인다, Working set 방법을 사용

3) 구역성 (Locality, 국부성) : 참조국부성(locality of reference)

- 프로세스가 실행되는 동안 일부 페이지만 집중적으로 참조하는 성질

① 시간 구역성
- 최근에 참조된 기억 장소가 가까운 장래에도 계속 참조될 가능성이 높음

예)Loop(반복), 스택, 부프로그램(Sub Routine), 카운팅(Counting), 집계(Totaling)에 사용되는 변수

② 공간 구역성
- 하나의 기억 장소가 참조되면 그 근처의 기억 장소가 계속 참조될 가능성이 높음

예) 순차적 코드(수행) 실행, 배열 순회, 같은 영역에 있는 변수 참조

Check 11. 페이지교체 알고리즘 [출제빈도: 上]

1. 페이지교체(Replacement) 알고리즘

1) 정의

- 페이지 부재(page fault)가 발생하였을 경우, 가상기억장치의 필요한 페이지를 주기억장치의 어떤 페이지 프레임을 선택, 교체 해야 하는 가를 결정하는 기법

2) 종류

-OPT (OPTimal replacement, 최적교체), FIFO (First In First Out), LRU (Least Recently Used),
-LFU (Least Frequently Used), NUR (Not Used Recently)

2. FIFO (First In First Out) ★★☆☆☆

- 가장 먼저 들여온 페이지를 먼저 교체시키는 방법
  (주기억장치 내에 가장 오래 있었던 페이지를 교체)
- 벨레이디의 모순(Belady's Anomaly) 현상
  : 페이지 프레임 수가 증가하면 페이지 부재가 더 증가

참조 페이지 : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

페이지 프레임	1	1	1	4	4	4	5	5	5	5	5	5
		2	2	2	1	1	1	1	1	3	3	3
			3	3	3	2	2	2	2	2	4	4
페이지 부재 : 9	O	O	O	O	O	O	O			O	O	

참조 페이지 : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

페이지 프레임	1	1	1	1	1	1	5	5	5	5	4	4
		2	2	2	2	2	2	1	1	1	1	5
			3	3	3	3	3	3	2	2	2	2
				4	4	4	4	4	4	3	3	3
페이지 부재 : 10	O	O	O	O			O	O	O	O	O	O

3. OPT(OPTimal replacement) 최적교체

-앞으로 가장 오랫동안 사용하지 않을 페이지를 교체하는 기법
(실현 가능성X)

4. LRU (Least Recently Used) ★★★★☆

- 최근에 **가장 오랫동안 사용하지 않은 페이지**를 교체하는 기법
- 각 페이지마다 계수기를 두어 현 시점에서 볼 때 가장 오래 전에 사용된 페이지를 교체

참조 페이지 : 1, 2, 3, 4, 1, 3, 5, 3, 2, 3

페이지 프레임	1	1	1	4	4	4	5	5	5	5
		2	2	2	1	1	1	1	2	2
			3	3	3	3	3	3	3	3
페이지 부재 : 7	O	O	O	O	O		O		O	

5. LFU (Least Frequently Used)

-**사용횟수가** 가장 적은 페이지를 교체하는 기법

6. NUR (Not Used Recently) ★★★★☆

- 최근에 사용하지 않은 페이지를 교체하는 기법
- "근래에 쓰이지 않은 페이지들은 가까운 미래에도 쓰이지 않을 가능이 높다." 라는 이론에 근거
- 각 페이지마다 2개의 하드웨어 비트(호출 비트, 변형 비트)가 사용됨

페이지	1	2	3	4
호출(참조)비트	0	0	1	1
변형 비트	0	1	0	1
교체 순서	1	2	3	4

- 호출 비트 : 1 (최근 참조)
- 변형 비트 : 1 (최근 갱신)

- 가장 우선적으로 교체 대상 : 참조도 안되고 변형도 안된 페이지

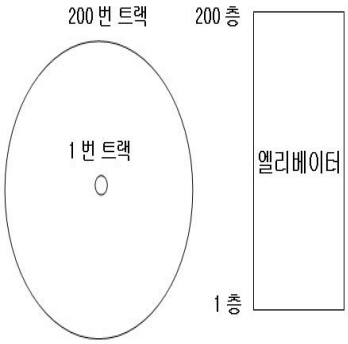
Check 12. 디스크 스케줄링 [출제빈도: 中]

1. 디스크 스케줄링(보조기억장치) ★☆☆☆☆

1) 정의
: 사용할 데이터가 디스크상의 여러 곳에 저장되어 있을 경우 데이터를 액세스하기 위해 디스크 헤더가 움직이는 경로를 결정하는 기법
2) 목적
: 처리량의 최대화, 응답시간의 최소화, 응답시간 편차의 최소화
3) 종류 : FCFS, SSTF, SCAN, C-SCAN 기법 등

2. FCFS(First-Come First-Service) ★★☆☆☆

- 입출력 요청 대기 큐에 들어온 순서대로 서비스를 하는 방법



* 대기 큐 : 108, 193, 47, 132, 24, 134, 75, 77
* 초기 헤드 위치 : 1

이동 순서 : 1 -> 108 -> 193 -> 47 -> .....
이동 거리 : 107 + 85 + 146 + ...

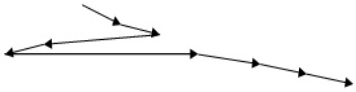
3. SSTF (Shortest Seek Time First) ★

- FCFS 보다 처리량이 많고 평균 응답 시간이 짧다
- 탐색 거리가 가장 짧은 트랙에 대한 요청을 먼저 서비스하는 기법
- 디스크 스케줄링 기법 중에서 현재 헤드 위치의 가까운 곳에 있는 모든 요구를 먼 곳보다 먼저 처리
- 탐색 시간 편차 ↑ : 안쪽이나 바깥쪽 트랙이 가운데 트랙보다 서비스를 덜 받는 경향

-> 헤드에서 멀리 떨어진 요청은 기아상태(starvation)가 발생할 수 있다.
-> 응답시간의 편차가 크므로 대화형 시스템에는 부적합
- 처리량이 많은 일괄처리 시스템에 유용

* 대기 큐 : 108, 193, 47, 132, 24, 134, 75, 77
* 초기 헤드 위치 : 63

안쪽 | 24, 47, 63,75, 77, 108, 132, 134, 193 | 바깥쪽



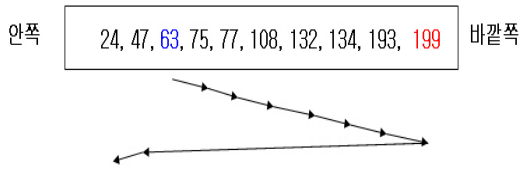
이동 순서 : 63 - 75 - 77 - 47 - 24 - 108 - 132 - 134 - 193
이동 거리 : 12 + 2 + 30 + 23 + 84 + 24 + 2 + 59 = 236

4. SCAN 한 방향으로 가장 짧은 거리 ★★★★☆

- SSTF가 갖는 탐색 시간의 편차를 해소하기 위한 기법
- 현재 진행중인 방향으로 가장 짧은 탐색 거리에 있는 요청을 먼저 서비스
- 현재 헤드의 위치에서 진행 방향이 결정되면 탐색 거리가 짧은 순서에 따라 그 방향의 모든 요청을 서비스하고, 끝까지 이동한 후 역방향의 요청 사항을 서비스함

=> 끝까지 이동하지 않을 경우 (LOOK 기법) ★☆☆☆☆
- 디스크 스케줄링 기본 전략

* 대기 큐 : 108, 193, 47, 132, 24, 134, 75, 77
* 번호가 0부터 199인 200개의 트랙, 헤드의 위치가 63에 있고 바깥쪽 방향으로 이동 중



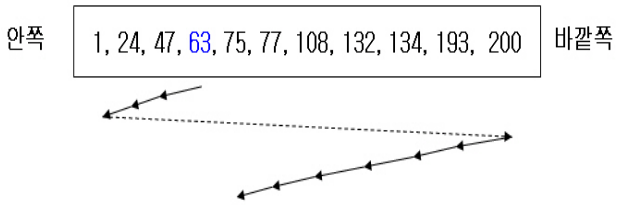
이동 순서 : 63 - 75 - 77 - 108 - 132 - 134 - 193 - 199 - 47 - 24
이동 거리 : 12 + 2 + 31 + 24 + 2 + 59 + 6 + 152 + 23 = 311

안쪽, 바깥쪽 트랙 번호가 없을 경우는 대기 큐에 있는 작업만 탐색한다.

5. C-SCAN (Circular SCAN) 바깥->안 가장 짧은 거리 ★★☆☆☆

- 항상 바깥쪽에서 안쪽으로 움직이면서 가장 짧은 탐색거리를 갖는 요청을 서비스
- 디스크 스케줄링 기법 중 가장 안쪽과 가장 바깥쪽의 실린더에 대한 차별대우를 없앤 기법
- 헤드는 트랙의 바깥쪽에서 안쪽으로 한 방향으로만 움직이며 서비스하여 끝까지 이동한 후, 안쪽에 더 이상의 요청이 없으면 헤드는 가장 바깥쪽의 끝으로 이동한 후 다시 안쪽으로 이동하면서 요청을 서비스함
  => 끝까지 이동하지 않을 경우 (C-LOOK 기법)

* 대기 큐 : 1, 200, 108, 193, 47, 132, 24, 134, 75, 77
* 초기 헤드 위치 : 63



이동 순서 : 63 - 47 - 24 - 1 - 200 - 193 - 134 - 132 - 108 - 77 - 75
이동 거리 : 16 + 23 + 23 + 199 + 7 + 59 + 2 + 24 + 31 + 2 = 386

6. N-step SCAN ★☆☆☆☆

- SCAN의 무한 대기 발생 가능성을 제거한 것으로 SCAN보다 응답시간의 편차가 적고, SCAN과 같이 진행 방향상의 요청을 서비스하지만, 진행 중에 새로이 추가된 요청은 서비스하지 않고 다음 진행시에 서비스하는 디스크 스케줄링

Check 13. 파일관리 [출제빈도: 上]

1. 파일시스템 특징 ★★☆☆☆

1) 파일 특성을 결정하는 기준
- 소멸성(Volatility) : 파일 추가/제거 빈도수
- 활성률(Activity) : 프로그램 한 번 수행 시 처리되는 레코드 수의 백분율
- 크기(Size) : 파일의 정보량

2) 파일 시스템의 기능
- 사용자가 파일을 생성, 수정, 제거할 수 있도록 한다.
- 적절한 제어방식을 통해 다른 사람의 파일을 공동으로 사용할 수 있도록 한다.
- 사용자가 이용하기 편리하도록 사용자에게 익숙한 인터페이스를 제공해야 한다.
- 정보의 암호화와 해독에 대한 기능을 제공한다.
- 불의의 사태에 대비한 예비(backup)와 복구(recovery) 능력을 갖추어야 한다.
- 파일의 무결성과 보안을 유지할 수 있는 방안 제공
- 번역기능 (X)

2. 파일 구성방식: 데이터베이스 10강 자류구조(파일편성)과 동일

3. 종류

1) 순차파일 ★★☆☆☆
- 적합한 기억 매체로는 자기 테이프를 쓰면 편리하다.
- 필요한 레코드를 삽입하는 경우 파일 전체를 복사해야 한다.
- 기억장치의 효율이 높다.
- 검색 시에 효율이 나쁘다. (다음 레코드 접근이 빠르다.)
- 부가적인 정보를 보관하지 않으므로 불필요한 공간 낭비가 없다.
- 파일 구성이 쉽다.
- 대화식 처리보다 일괄 처리에 적합한 구조이다.

2) 색인순차파일 ★★★☆☆
- 각 레코드는 레코드 킷값에 따라 논리적으로 배열된다.
- 시스템은 각 레코드의 실제주소가 저장된 인덱스를 관리한다.
- 일반적으로 디스크 기억장치에 많이 이용된다.
- 색인 구성 : 실린더 색인, 트랙 색인, 마스터 색인

3) 직접파일 ★★★☆☆
: 해싱 등의 사상 함수를 사용하여 레코드 키에 의한 주소 계산을 통해 레코드를 접근할 수 있도록 구성한 파일

- 적합한 장치로는 자기디스크를 주로 사용한다.
- 직접 접근 기억장치의 물리적 주소를 통해 직접 레코드에 접근한다.
- 키에 일정한 함수를 적용하여 상대 레코드 주소를 얻고, 그 주소를 레코드에 저장하는 파일 구조이다.

- 직접 접근 기억장치의 물리적 구조에 대한 지식이 필요하다.
- 판독이나 기록의 순서에는 제약이 없다.

Check 14. 파일 디스크립터 [출제빈도: 上]

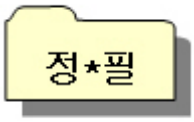
1. 파일 디스크립터 = FCB : File Control Block ★★★★☆
(파일 제어 블록)

- 파일을 관리하기 위한 시스템이 필요로 하는 파일에 대한 정보를 갖는 제어 블록 => 사용자 직접 참조 X
- 파일이 엑세스되는 동안 운영체제가 관리 목적으로 알아야 할 정보를 모아 놓은 자료구조이다
- 파일마다 독립적으로 존재, 시스템 마다 다른 구조 가짐
- 보통 보조기억장치에 저장되었다가 파일이 오픈 될 때 주기억장치로 전달
- 정보 : 생성 날짜 및 시간, 위치, 액세스 횟수, 이름, 구조, 크기, 접근 제어, 수정 시간
- 파일 작성자 (X)
- 오류에 대한 수정 방법 (X)
- 파일의 백업 방법 (X)

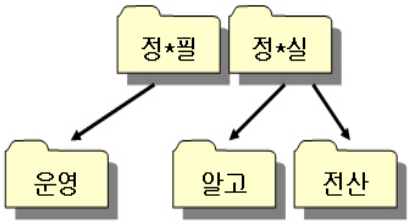
Check 15. 디렉토리 구조 [출제빈도: 上]

1. 디렉토리 구조 ★★★★☆

1) 1단계 구조
- 가장 간단하고, 모든 파일이 하나의 디렉토리 내에 위치하여 관리되는 구조
- 관리 불편 -> 모든 파일명 다르므로



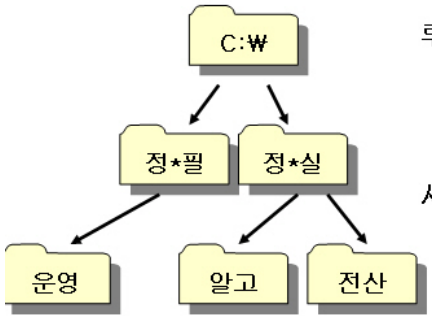
2) 2단계 구조
- 마스터 / 사용자 파일 디렉토리
- 서로 다른 디렉토리에서는 동일한 파일 이름을 사용할 수 있음



마스터 파일 디렉토리

사용자 파일 디렉토리

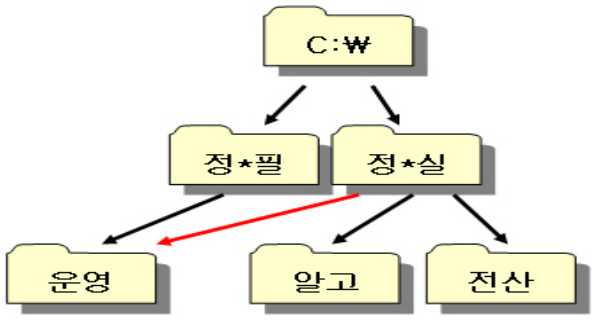
3) 트리 구조
- 루트 / 종속(서브) 디렉토리
- DOS, Windows, UNIX 등의 운영체제에서 사용되는 디렉토리 구조
- 동일한 이름의 파일이나 디렉토리를 생성할 수 있음
- 디렉토리의 생성과 파괴가 비교적 용이함



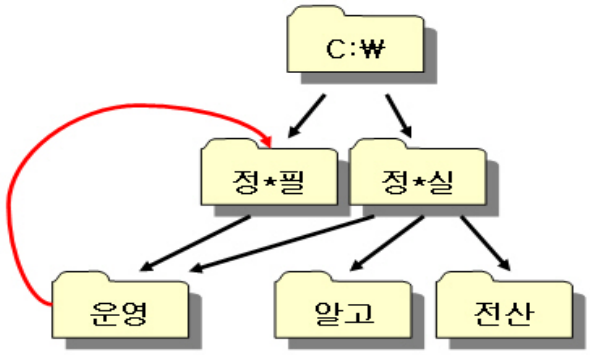
루트 디렉토리

서브(부) 디렉토리

4) 비순환 그래프 구조
- 부 디렉토리, 파일 공유(O), 사이클(X)
- 디스크 공간을 절약할 수 있음
- 하나의 파일이나 디렉토리가 여러 개의 경로, 이름을 가질 수 있음



5) 일반 그래프 구조
- 트리 구조에 링크(Link)를 첨가 -> 순환(O)
- 그래프 탐색 알고리즘이 간단
- 원하는 파일로 접근이 쉽다.



Check 16. 여러 가지 기법 [출제빈도: 上]

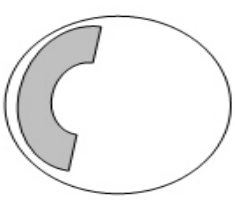
1. 디스크 공간 할당 기법 ★★★☆☆

1) 연속 할당 (단일 P/G)
- 논리적으로 연속된 레코드들이 물리적으로 서로 인접하게 저장

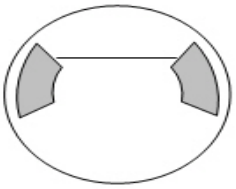
→ 액세스 시간 감소
- 생성되는 파일 크기만큼의 공간이 있어야 함 (외부 단편화 O)

2) 불연속 할당 (링크, 다중 P/G)
- 프로그램 적재 효율적 (외부 단편화 X)
- 파일 생성시 파일의 크기를 알 필요가 없다

- 섹터 단위
- 블록 단위 : 블럭체인, 인덱스 블럭체인, 블럭단위파일 사상
- 파일 할당 표(FAT) : 사용자가 해당 블록의 포인트를 실수로 지워지게 하는 것을 예방하고 블록 접근을 빠르게 하기 위하여 포인터를 모아 놓은 곳



[연속 할당]



[불연속 할당]

2. 자원 보호 기법 ★★★☆☆

: 컴퓨터 시스템에서 사용되는 자원들(파일, 프로세스, 메모리 등)에 대하여 불법적인 접근방지와 손상 발생 방지

1) 접근 제어 행렬(access control matrix)

: 자원 보호의 일반적인 모델로, 객체에 대한 접근 권한을 행렬로써 표시한 기법

영역 \ 객체	파일	프로세스	메모리
권우석	E	REW	E
김영희	RW	NONE	R

2) 접근 제어 리스트(access control list)

→ 접근제어행렬에서 열(객체) 중심

: 객체와 그 객체에 허용된 조작 리스트 이며, 영역과 결합되어 있으나 사용자에 의해 간접적으로 액세스되는 기법

객체	접근 제어 리스트
파일	(권,E), (김,RW)
프로세스	(권,REW)
메모리	(권,E), (김,R)

3) 권한 리스트(capability list) → 접근제어행렬에서 행(영역) 중심

: 접근 제어 행렬에 있는 각 행, 즉 영역을 중심으로 구성한 것으로서 각 사용자에 대한 자격들로 구성되며, 자격은 객체와 그 객체에 허용된 연산 리스트

권우석	
파일	E
프로세스	REW
메모리	E

김영희	
파일	RW
프로세스	NONE
메모리	E

3. 파일 보호 기법 ★★★☆☆

1) 파일의 명명 (Naming)

: 파일 이름을 모르는 사용자를 접근 대상에서 제외시키는 기법

2) 비밀번호 (Password, 암호)

: 각 파일에 판독 암호와 기록 암호를 부여하여 암호를 아는 사용자에게만 접근을 허용하는 기법

3) 접근 제어 (Access Control)

: 사용자의 신원에 따라 서로 다른 접근 권한을 허용한다 (접근 제어 행렬 응용)

4. 보안 기법 ★★★★★

1) 외부 보안
: 불법 침입자나 천재지변으로부터 시스템을 보호하는 것
- 시설 보안 : 감지 기능을 통해 외부 침입자나 화재,홍수와 같은 천재지변으로부터의 보안

2) 내부 보안 : 하드웨어나 운영체제의 내장된 기능

3) 사용자 인터페이스 보안
: 사용자의 신원을 운영체제가 확인하는 절차를 통해 불법침입자로부터 보호

* 인증 : 컴퓨터 시스템에서 전송 정보가 오직 인가된 당사자에 의해서만 수정될 수 있도록 통제하는 것
* 백업 : 천재지변이나 사고로 인해 정보의 손실이나 파괴를 막기 위해 취할 수 있는 방법

5. 암호화 기법 ★★★☆☆

1) 비밀키 시스템 (Private Key System, 개인키 시스템)
- 암호화키 = 복호화키
- 대칭 암호화 방식 : DES

2) 공용키 시스템 (Public Key System, 공개키 시스템)
- 암호화키 <> 복호화키 (비대칭 암호화 기법)
- 대표적 암호화 방식 : RSA
- 키의 분배가 용이하다.
- 암호키는 공개되어 있어 누구나 사용할 수 있으나 해독키는 당사자만 알고 있다.
- 암호화키와 해독키가 따로 존재한다.

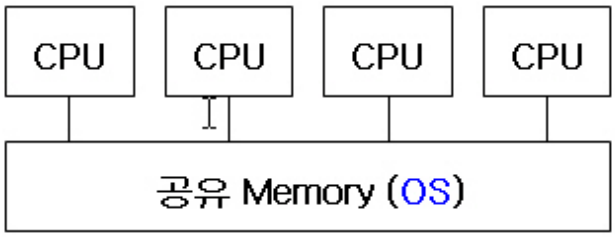
* 인증 교환 기법 : 수신자가 메시지 전송도중에 변경되지 않았음을 확인할 수 있으며, 메시지가 정당한 상대방으로부터 전달된 것임을 확인할 수 있는 기법

Check 17. 분산처리시스템 [출제빈도: 上]

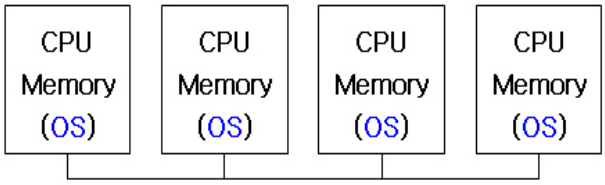
1. 컴퓨터 시스템의 구조 ★★★★☆

- Flynn이 제안한 4가지 병렬처리 방식
① SISD ② SIMD(배열, array) ③ MISD(실제구현X)
④ MIMD(다중 처리기, 다중 컴퓨터)

1) 강결합 (다중 처리기)
- 여러 개의 처리기(CPU)와 하나의 기억장치(공유 메모리)를 두어 처리
- 프로세스간의 통신은 공유메모리를 이용한다
- 메모리에 대한 프로세스 간의 경쟁 최소화가 고려되어야 한다
- 가장 복잡하지만 가장 강력한 구조이다
- 프로세서의 수를 늘린다고 해도 시스템 효율은 향상되지 않는다
- 운영체제가 여러 CPU 간의 기억장치를 공유하기 위한 스케쥴링이 복잡해진다

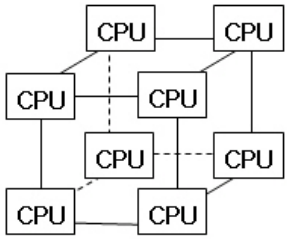


2) 약결합 (다중 컴퓨터, 분산 처리)
- 여러 개의 처리기와 독자적인 기억장치(OS)를 두어 통신 회선을 연결해서 처리
- 둘 이상의 독립된 컴퓨터 시스템을 통신 링크를 이용하여 연결한 시스템
- 기억장치 공유 (X)

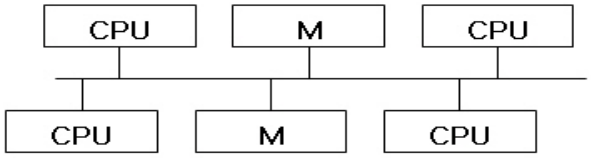


2. 처리기(Processor) 연결 방식 ★☆☆☆☆

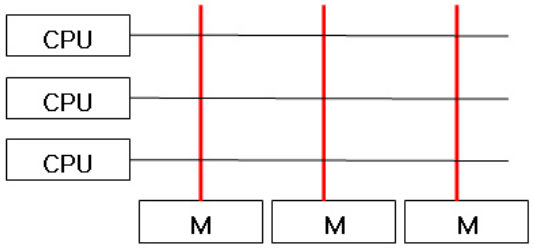
1) 하이퍼 큐브
- 연결점 수가 n 이면 프로세서의 수는 2^n 개



2) 공유 버스 기법
- 버스로 연결한 방식
- 버스에 이상이 발생하면 전체 시스템이 가동되지 않음
- 증설 절차가 간단



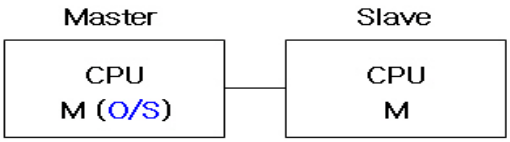
3) 크로스바 교환 행렬



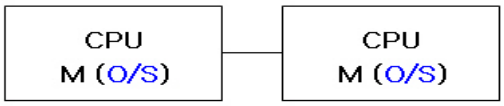
- 버스의 수를 기억장치 수만큼 증가시켜 연결한 방식

3. 다중 처리 운영체제 구성 ★★☆☆☆

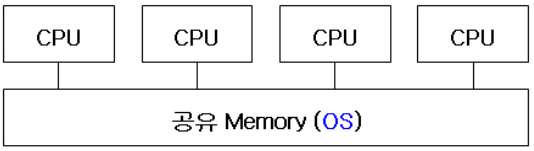
1) Master/Slave(주/종) 처리기
- 주 프로세서 : 입출력과 연산 담당, 운영체제를 수행
- 종 프로세서 : 연산만 담당, 사용자 프로그램만 담당
- 주 프로세서가 고장 나면 전체 시스템 다운



2) 분리 수행 처리기
- 주/종 처리기의 비대칭성을 보완하여 각 프로세서가 독자적인 운영체제를 가짐
- 한 프로세서가 고장 나더라도 전체 시스템이 다운되지 않음



3) 대칭적 처리기
- 분리 실행 처리기 구조의 문제점을 보완한 것으로, 여러 프로세서들이 완전한 기능을 갖는 하나의 운영체제를 공유



4. 분산 처리 시스템 ★

1) 목적

(중앙 집중 형태에서 분산처리 시스템으로 발전하게 된 이유)

: 자원 공유, 연산 속도 향상, 신뢰도 향상, 컴퓨터 통신

2) 특징

- 과부하를 줄일 수 있고
- 점진적 확장 가능

: 특정한 시스템 병목 현상을 제거하기 위해 필요한 자원을 추가할 수 있으므로 선택적인 성능 향상을 가능

- 빠른 반응 시간
- 사용가능도가 향상

: 다수의 구성 요소가 존재하므로 일부가 고장 나더라도 나머지 일부는 계속 작동 가능

- 설계 복잡 -> 소프트웨어 개발이 어렵다
- 보안문제가 발생한다
- 공유자원에 접근할 경우 시스템 유지를 위해 제어를 분산 할 필요가 있다
- 처리기와 입력 장치와 같은 물리적인 자원을 분산 할 수 있다
- 시스템 성능과 가용성을 증진하기 위해 자료를 분산 할 수 있다
- 분산된 노드들은 통신 네트워크를 이용하여 메시지를 주고받음으로서 정보를 교환한다
- 투명성

3) 투명성 (Transparency) ★★★☆☆

: 사용자가 분산된 여러 자원의 위치 정보를 알지 못하고 마치 하나의 커다란 컴퓨터 시스템을 사용하는 것처럼 인식하도록 한다

- 위치 투명성 : 사용자는 각 컴퓨터들이 어느 곳에 위치하는지 몰라서 자원을 사용할 수 있다.
- 이주 투명성 : 자원 이동에 제한이 없음
- 병행 투명성 : 다중 사용자들이 자원들을 자동으로 공유할 수 있다
- 복제 투명성 : 사용자에게 통지할 필요 없이 시스템 안에 자원들의 부가적인 복사를 자유롭게 할 수 있다

5. 분산 운영체제의 실제 예 ★☆☆☆☆

: 운영체제의 형태에 따른 분류 중 사용자는 컴퓨터들의 종류를 알 필요가 없으며, 원격지 자원들을 그들의 지역 자원에 접근하는 방식과 동일한 방식으로 접근하도록 처리하는 형태의 운영체제

1) NFS(Network File System) : 선 마이크로시스템
2) LoCUS : 캘리포니아 대학
3) Andrew : 카네기 멜론 대학

6. 위상(Topology)에 의한 분류 ★

1) 완전 연결 (Fully Connection)형 = 망형

- 각 사이트(노드)들이 시스템 내의 다른 모든 사이트들과 직접 연결된 구조
- 기본 비용은 많이 들지만 통신 비용은 적게 들고, 신뢰성이 높음
- 사이트들 간의 메시지 전달이 매우 빠르다

2) 계층형 (Hierarchy) = 트리형

- 분산 처리 시스템의 가장 대표적인 형태
- 부모 사이트가 고장나면 그 자식 사이트들은 통신이 불가능함

3) 성형 = 스타형

- 모든 사이트가 하나의 중앙 사이트에 직접 연결
- 중앙 사이트가 고장 날 경우 모든 통신이 단절됨

4) 링형 = 환형

- 인접하는 다른 두 사이트와만 직접 연결된 구조
- 정보는 단방향 또는 양방향으로 전달될 수 있음
- 목적 사이트에 데이터를 전달하기 위해 링을 순환할 경우 통신 비용이 증가함
- 새로운 노드를 추가할 경우 통신회선을 절단해야 한다

5) 다중 접근 버스 연결(Multi Access Bus Connection)형

- 하나의 공유 버스에 연결된 구조 (물리적 구조 간단)
- 사이트의 고장은 다른 사이트의 통신에 영향을 주지 않지만, 버스의 고장은 전체 시스템에 영향을 줌
- 노드의 추가와 삭제가 용이하다.

7. 클라이언트/서버 시스템 ★☆☆☆☆

: 서버(정보 제공 컴퓨터), 클라이언트(정보 요청 컴퓨터)로 구성된 방식

- 서버는 공유된 다양한 시스템 기능과 자원을 제공해야 한다
- 고성능 워크스테이션에서 가능한 그래픽 사용자 인터페이스를 용이하게 쓸 수 있다
- 시스템 확장이 용이하고 유연성이 있다
- 사용자 중심의 개별적인 클라이언트 운영환경이 가능하다
- 개방 시스템을 받아들이도록 참작하고 독려
- 많은 자원을 공유할 수 있다

Check 18. UNIX [출제빈도: 上]

1. 특징 ★

- 높은 이식성과 확장성
- 다양한 네트워킹 기능
- 대화식 시분할 운영체제
- 대부분 C언어로 작성
- 다중 사용자 시스템(Multi-user system)
- 다중 태스킹(작업) 운영체제 : 동시에 여러 가지 작업을 수행
- 파일 소유자, 그룹 및 그 외 다른 사람들로부터 사용자를 구분하여 파일을 보호
- 파일 시스템 : 계층(트리) 구조
- 사용자 위주의 시스템 명령어 제공
- 사용자는 하나 이상의 작업을 백그라운드에서 수행할 수 있어 여

러 개의 작업을 병행 처리할 수 있다
- 개방형 시스템 : 구조 공개, 제품의 공급업자가 많다, 라이센스 비용이 싸다
- 단일 작업용, Stand alone 시스템 (X)

* 파이프 라인(pipeline) : UNIX에서 두 프로세스를 연결하여 프로세스간 통신을 가능하게 하며, 한 프로세스의 출력이 다른 프로세스의 입력으로 사용됨으로써 프로세스간 정보 교환이 가능하도록 하는 것 (큐, FIFO)

2. 구성 ★

1) 커널 (Kernel)
- UNIX의 가장 핵심적인 부분
- 주기억장치에 적재된 후 상주하면서 실행
- 프로세스, 기억장치, 파일, 입·출력 관리
- 프로세스간 통신, 데이터 전송 및 변환 등 여러 가지 기능 수행
- 파일 시스템의 접근 권한을 처리
- 자원 활용도를 높이기 위해 스케줄링

2) 쉘 (Shell)
- 명령어 해석기
- 시스템과 사용자 간의 인터페이스 담당

3. 파일 시스템의 구조 (계층적 트리 구조)★

1) 부트 블록 : 부팅 시 필요한 코드를 저장하고 있는 블록

2) 슈퍼 블록 : 전체 파일 시스템에 대한 정보를 저장하고 있는 블록 (디스크 자체에 관련된 정보)

3) Inode 블록 : 각 파일이나 디렉토리에 대한 모든 정보를 저장하고 있는 블록 (FCB 과 유사)
- 파일 소유자 정보, 크기, 주소, 링크수, 종류(유형), 보호권한
- 파일 생성, 사용, 최종 수정시간
- 파일 최초 수정 시간(X), 파일 경로(X), 사용횟수(X), 파일이 사용된 시간대별 내역(X),
파일의 우선 순위(X)

4) 데이터 블록 : 디렉토리별로 디렉토리 엔트리와 실제 파일에 대한 데이터가 저장된 블록

4. 명령어★

1) 프로세스 관련
- **fork** : 새로운 프로세스 생성, 복제
(자식 프로세스 생성, 부모 프로세스를 복제)
- wait : 자식 프로세스의 하나가 종료될 때까지 부모 프로세스를 임시 중지

2) 파일 등 관련
- mount : 새로운 파일 시스템을 서브 디렉토리에 연결
- ls : 디렉토리 내용 보기 (파일의 조작과 무관)
- chmod : 파일의 권한 모드(읽기, 쓰기, 실행) 설정
-> 파일의 접근을 제한
- cat : 파일 내용을 화면에 표시
- & : 백그라운드 처리를 위해 명령
(장점 : 수행중인 명령문이 끝나기 전에 다른 명령문을 줄 수 있다)