14 운영체제

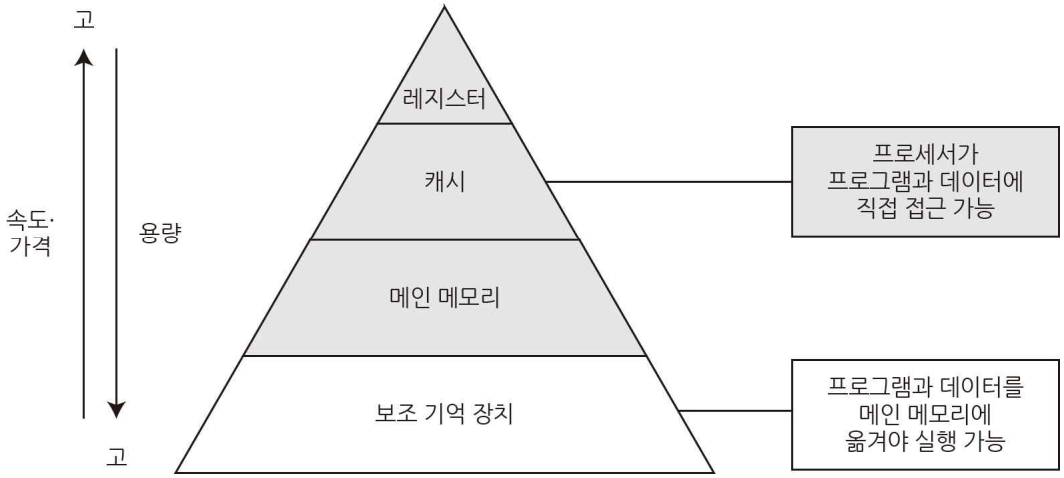
Section 1. 운영체제 기초

1. 기억장치

(1) 기억장치의 개념

- 데이터, 프로그램, 연산의 중간 결과 등을 일시적 또는 영구적으로 저장하는 장치

(2) 기억장치의 종류



① 레지스터

- 중앙처리장치 내부에 존재하는 기억장치
- 접근 시간이 중앙처리장치의 처리 속도와 비슷

② 캐시 메모리

- 중앙처리장치가 주기억장치에 접근할 때 속도 차이를 줄이기 위해 사용
- 실행 중인 프로그램의 명령어와 데이터를 저장

③ 주기억장치

- 중앙처리장치가 직접 데이터를 읽고 쓸 수 있는 장치
- 레지스터나 캐시 메모리보다 기억 용량이 크다.
- 종류

종류	설명
ROM (Read Only Memory)	- 읽기만 가능한 읽기 전용 메모리 - 비 휘발성 메모리 - 종류 : mask-ROM, PROM, EPROM, EEPROM
RAM (Random Access Memory)	- 기억장소를 임의로 접근할 수 있는 메모리 - 읽고 쓰기가 가능한 휘발성 메모리 - SRAM : 전원이 공급되는 중에 내용이 사라지지 않음 (캐쉬메모리로 사용) - DRAM : 일반적인 주기억장치로, 일정 시간이 지나면 내용이 사라지는 RAM

④ 보조기억장치

- 주기억장치에 비해 접근 시간은 느리지만 기억 용량이 크다.
- 종류 : HDD, SSD(Solid State Drive), CD, USB, 플로피 디스크 등

2. 시스템 소프트웨어

(1) 시스템 소프트웨어의 개념

- 응용 소프트웨어를 실행하기 위한 플랫폼을 제공
- 컴퓨터 하드웨어를 동작하고 접근한다.
- 종류
  - 로더
  - 운영체제
  - 장치 드라이버
  - 번역기 (컴파일러, 어셈블러)
  - 링커
  - 유틸리티

(2) 시스템 소프트웨어의 구성

① 제어 프로그램

- 감시 프로그램 (Superviosr Program)
  - 각종 프로그램의 실행과 시스템 전체의 작동 상태를 감시/감독하는 프로그램
- 작업관리 프로그램 (Job Control Program)
  - 연속 처리를 위한 스케줄 및 시스템 자원 할당 등 담당
- 데이터 관리 프로그램
  - 주기억 보조기억장치 사이의 자료전송, 파일의 조작 및 처리, 입출력 자료와 프로그램간의 논리적 연결 등을 처리할 수 있도록 관리

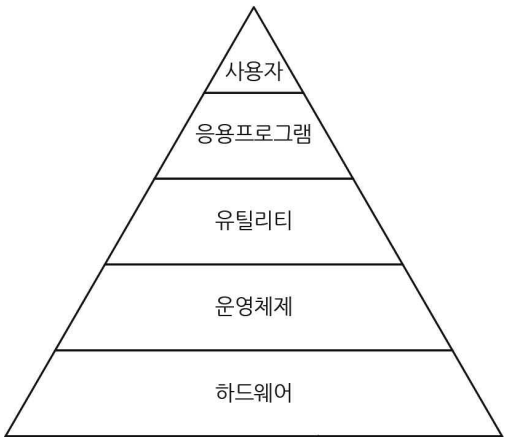
② 처리 프로그램

- 서비스 프로그램(Service Program)
  - 효율성을 위해 사용 빈도가 높은 프로그램
- 문제 프로그램(Problem Program)
  - 특정 업무 해결을 위해 사용자가 작성한 프로그램
- 언어 번역 프로그램(Language Translator Program)
  - 어셈블러, 컴파일러, 인터프리터

3. 운영체제

(1) 운영체제의 개념

- 응용프로그램이 실행되는 과정에서 하드웨어들을 제어하여 응용프로그램을 실행시키고 실행 결과를 보일 수 있도록 컴퓨터 내부 동작을 관리하는 소프트웨어
- 컴퓨터 시스템의 자원들을 효율적으로 관리하며, 사용자가 컴퓨터를 편리하고, 효과적으로 사용할 수 있도록 환경을 제공하는 여러 프로그램의 모임



(2) 운영체제의 특징

- 여러 프로그램을 같이 실행시키면서 서로 충돌없이 실행되도록 프로그램 실행을 제어
- 컴퓨터 하드웨어와 프로그램 사이의 인터페이스
- 컴퓨터에서 발생할 수 있는 각종 오류나 예외상황을 감지하고 상위 계층에게 보고
- 컴퓨터 자원의 이용 현황에 대한 통계자료를 제공

(3) 운영체제의 기능

- 프로세스 관리
  - 프로세스를 생성하고 실행을 제어, 관리하는 기능
- 메모리 관리
  - 프로세스가 실행될 수 있도록 메모리 공간을 할당하고 회수하는 기능
- 파일 관리
  - 파일을 보조기억장치에 저장하고 파일 시스템을 운영하는 기능
- 입출력 관리
  - 컴퓨터 시스템에서의 입력과 출력을 관리하는 기능
- 보조기억장치 관리
  - 보조기억장치의 공간을 할당하고 관리하는 기능
- 네트워킹
  - 컴퓨터 통신에 필요한 제어 관리 기능
- 정보 보안 관리
  - 사용자 인증 및 실행권한 관리
- 명령해석 시스템
  - 사용자가 운영체제에 전달하는 명령을 해석하고 관련 함수를 실행 시키는 기능

⑷ 운영체제 운용 기법

- 일괄 처리 시스템(batch processing system)
  - 초기 운영체제의 형태. 여러 작업을 한 번에 묶어서 처리한다.
- 다중 프로그래밍 시스템(multi programming system)
  - 하나의 작업이 입/출력중일 때 다른 작업을 할당하여 cpu 사용률과 처리량 향상시키는 기법
  - 사용자 입장에서는 하나의 cpu지만 동시에 여러 프로그램이 실행되는 것처럼 보인다.
- 시분할 시스템 (time sharing system)
  - 타임 슬라이스 또는 타임 퀀텀이라 부르는 일정 작업 시간 동안 작업 실행
- 다중 처리 시스템 (multi-processing system)
  - 여러 개의 cpu를 통하여 동시에 여러 개의 작업을 처리하는 운용 기법
  - 병렬 처리 시스템(parallel processing system)이라고도 한다.
- 실시간 처리 시스템 (real time processing system)
  - 요청한 실행을 즉시 실행하는 기법
- 다중 모드 시스템 (multi-mode system)
  - 일괄 처리, 다중 프로그래밍, 시분할, 다중 처리, 실시간 처리시스템을 모두 혼용하여 사용
- 분산 처리 시스템(distribute processing system)
  - 둘 이상의 독립된 시스템이 통신으로 연결되고 상호작용하는 약 결합 방식

⑸ 운영체제 성능 평가 기준

- 처리량(Throughput)
  - 일정 시간 내에 시스템이 처리하는 일의 양
- 반환시간(Turnaround Time)
  - 요청한 작업에 대하여 결과를 돌려줄 때까지 소요되는 시간
  - 대기시간 + 실행시간 + 응답시간
- 신뢰도(Reliability)
  - 작업의 결과가 얼마나 정확하고 믿을 수 있는가의 기준
- 사용 가능도(Availability)
  - 시스템을 사용할 필요가 있을 때 즉시 사용 가능한 정도

4. 운영체제의 종류

(1) 윈도우(Windows)

- MS-DOS의 멀티태스킹 기능과 GUI 환경을 제공하는 운영체제
- 마이크로소프트사에서 개발한 운영체제
- 윈도우즈 95를 시작으로 98, ME, XP, 7, 8, 10 등의 버전으로 출시
- 특징

특징	설명
GUI 제공	- 그래픽 사용자 인터페이스 제공
선점형 멀티태스킹 방식	- 동시에 여러 개의 프로그램을 실행 - 운영체제가 각 작업의 CPU 이용시간 제어
자동감지 기능 제공 (Plug and Play)	- 하드웨어 설치 시, 필요한 시스템 환경을 운영체제가 자동 구성
OLE (Object Liking and Embedding) 사용	- 개체를 현재 작성중인 문서에 삽입하여 편집할 수 있게 하는 기능 제공

(2) 리눅스(Linux)

- 1991년 리누스 토발즈에 의해 오픈소스로 개발된 유닉스 호환 OS
- 특징

특징	설명
다중 사용자 시스템	- 하나의 시스템에 다수의 사용자가 동시 접근 가능 - 사용자들은 동시에 여러 프로그램 사용 가능
오픈 소스	- 커널과 내장된 응용프로그램이 공개되어 있어, 누구나 용도에 맞게 수정해서 사용 가능
파일 시스템	- 리눅스 고유의 파일 시스템 EXT2~EXT4 와 FAT, FAT32등의 파일 시스템도 지원
이식성/유연성/확장성	- C언어로 개발되어, 다른 시스템이 이식이 가능
다양한 배포판	- 서버, 개발용, PC용 모두 사용 가능하고, 그에 맞는 다양한 배포판이 존재

- 장/단점

장점	단점
- 유닉스와 완벽하게 호환이 가능 - 공개 운영체제 - PC용 OS보다 안정적 - 강력한 보안기능 - 개발 환경이 풍부	- 기술지원 부족 - 사용자의 숙련된 기술 필요 - 한글 입출력을 하는데 어려움이 존재

⑶ 유닉스(Unix)

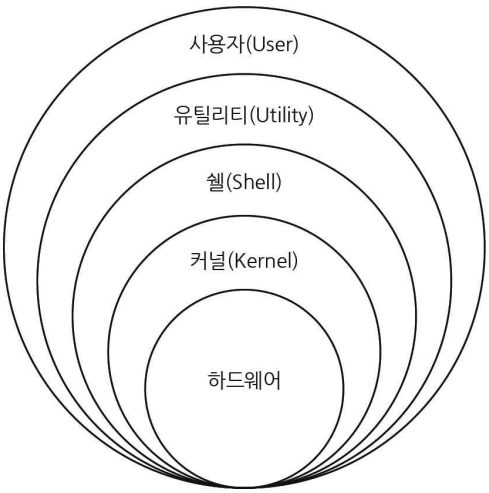
① 유닉스 개요

- 1969년 미국 AT&T 벨연구소(Bell Lab.)가 개발한 공개형 오픈소스 운영체제(OS)
- 1969년 벨연구소 소속의 켄 톰슨(Ken Tompson)이 어셈블리 언어를 사용하여 개발했으며, 1972년 데니스 리치(Dennis Ritchie)가 C 언어를 사용하여 다시 작성
- 벨 연구소에서 개발한 운영 체제로, 대부분의 현대적 컴퓨터 운영 체제의 원형
- 윈도우를 제외한 macOS, iOS 등의 대부분의 운영 체제가 유닉스를 뿌리로 하고 있다.

② 유닉스 특징

- 시분할 시스템(Time Sharing System)을 위해 설계된 대화식 운영체제
- 대부분 C언어로 작성되어 있어 이식성이 높으며 장치, 프로세스 간의 호환성이 높다.
- 다중 사용자, 다중 작업을 지원
- 많은 네트워킹 기능을 제공하므로 통신망 관리용 운영체제로 적합
- 트리구조의 파일 시스템
- 전문적인 프로그램 개발에 용이
- 다양한 유틸리티 프로그램 존재

③ Unix 시스템의 구성



- 커널(Kernel)
  - UNIX의 가장 핵심적인 부분
  - 컴퓨터가 부팅될 때, 주 기억장치에 적재된 후 상주하면서 실행
  - 하드웨어를 보호하고, 프로그램과 하드웨어 간의 인터페이스 역할 담당
  - CPU 스케줄링, 기억장치 관리, 파일 관리, 입출력 관리, 프로세스간 통신, 데이터 전송 및 변환 등 여러 가지 기능 수행
- 쉘(Shell)
  - 명령어 해석기
  - 사용자의 명령어를 인식하여 프로그램을 호출하고 명령을 수행하는 명령어 해석기
  - 시스템과 사용자 간의 인터페이스 담당
  - DOS의 COMMAND.COM과 같은 기능
  - 공용 Shell(Bourne Shell, C Shell, Korn Shell)이나, 사용자가 만든 Shell 사용 가능

- Utility Program
  - 일반 사용자가 작성한 응용 프로그램을 처리하는 데 사용
  - 에디터, 컴파일러, 인터프리터, 디버거 등

④ Unix 파일 시스템

- Unix 파일 시스템 특징
  - UNIX 파일 시스템의 구조는 트리 구조로 되어 있다.
  - 디렉터리나 주변장치도 모두 파일로 취급
  - 파일 생성 및 삭제 기능, 보호 기능
  - 파일 형식은 일반파일, 디렉터리 파일, 특수파일 형태 제공
- Unix 파일 시스템의 구조
  - 부트블록 : 부팅시 필요한 코드를 저장하고 있는 블록
  - 슈퍼블록 : 전체 파일 시스템에 대한 정보를 저장하고 있는 블록
  - I-node 블록 : 각 파일이나 디렉터리에 대한 모든 정보를 저장하고 있는 블록
  - 데이터 블록 : 실제 파일에 대한 데이터가 저장된 블록

⑤ 파일 디스크립터 (FD, File Descriptor)

- 파일 디스크립터 특징
  - 유닉스 시스템에서 프로세스가 파일들을 접근할 때 이용
  - 파일 제어 블록(File control Block)이라고도 한다.
  - 파일 관리를 위해 시스템이 필요로 하는 정보를 가지고 있다.
  - 보조기억장치에 저장되어 있다가 파일이 OPEN 되면 주기억장치로 이동된다.
  - 파일마다 독립적으로 존재하며, 시스템에 따라 다른 구조를 가질 수 있다.
  - 사용자가 직접 참조 할 수 없다.
- 파일 디스크립터 정보
  - 파일 이름 및 파일 크기
  - 보조기억장치에서의 파일 위치
  - 파일 구조(순차파일/색인순차파일/색인파일 등)
  - 보조기억장치의 유형(자기 디스크/자기테이프 등)
  - 액세스 제어 정보
  - 파일 유형(텍스트파일, 목적프로그램파일 등)
  - 생성 날짜와 시간, 제거 날짜와 시간
  - 최종 수정 날짜 및 시간
  - 액세스한 횟수(파일 사용 횟수)

(4) MacOS

- 애플사가 개발한 유닉스 기반의 운영체제
- iOS, tvOS, watchOS 등 애플의 다른 운영체제들과 상당수의 코드베이스를 공유
- SwiftUI와 함께 Cocoa, Core Foundation 등의 시스템 API를 제공
- macOS는 라이선스 상으로 애플 기기 이외의 기기에서는 구동시킬 수 없다.
- 애플의 하드웨어가 아닌 다른 x86-64 컴퓨터에서 macOS를 구동시키는 것을 해킨토시라 부른다.

5. 운영체제의 명령어

(1) Windows 기본 명령어

명령어	설명
CMD	Windows 명령 프롬프트 창을 열어준다.
DIR	현재 디렉터리의 파일 목록을 출력
CD	현재 디렉터리의 위치를 보여주거나 다른 위치로 이동 ex) cd c:₩windows
COPY	파일을 복사한다.
DEL	파일을 삭제한다.
MOVE	파일을 이동한다.
REN	파일의 이름을 변경한다.
MD	디렉터리를 생성한다.
CLS	화면의 내용을 지운다.
CHKDSK	디스크의 상태를 점검한다.
FORMAT	디스크를 초기화 한다.
CALL	프로그램을 호출한다.
ATTRIB	파일의 속성을 변경한다.
FIND	파일에서 내용을 검색한다.

⑵ Unix 주요 명령어

명령어	설명
access	파일의 접근 가능 여부 결정
chmod	파일 또는 디렉토리에 대한 접근권한을 변경
close	FCB(File Control Block)를 닫는다.
chgrp	파일의 그룹명 변경
chown	파일의 소유자 변경
chdir	디렉터리 변경 명령
mkdir	디렉터리 생성 명령
rmdir	디렉터리 삭제 명령
mount	파일 시스템에 새로운 파일 시스템을 연결할 때 사용
umount	파일 시스템에서 서브 디렉터리 제거시 사용
exit	프로세스 종료
kill	프로세스 제거
fork	새로운 프로세스를 생성, 복제하는 명령
getpid	자신의 프로세스 명, 그룹명, 부 프로세스의 정보를 얻는다.
getppid	부모 프로세스의 ID를 얻는다.
sleep	프로세스를 일정 시간 동안 중단
uname	현재 운영체제의 버전 정보를 확인
ps	프로세스 상태 출력
exec	새로운 프로그램을 수행시키는 명령
vi	편집기 명령어
cat	파일 내용을 화면에 출력
rm	파일이나 디렉터리 삭제
cp	파일을 복사
mv	파일 이동
grep	파일이나 프로세스를 찾는 명령
ls	파일 목록 확인
du	파일의 사용량 출력
finger	사용자 정보 표시

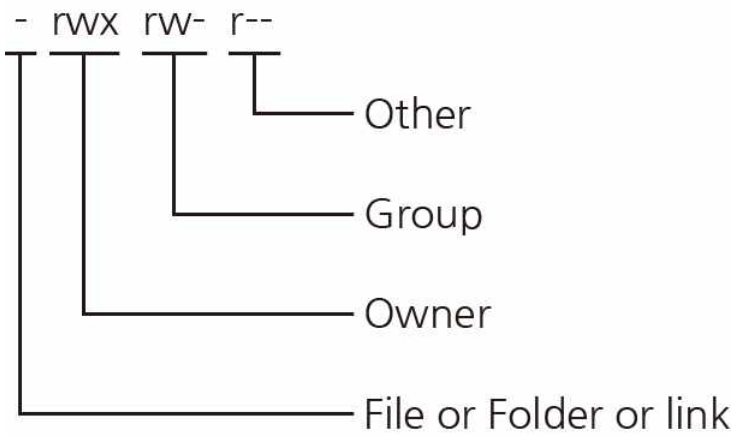
(3) Linux, Unix 파일 접근 권한 관리

① 파일 확인 방법

- ls -al 명령을 이용하여 파일의 상세정보 및 파일 접근 권한을 확인할 수 있다.

```
[root@db1 data]# ls -al
합 계  2959848
drwxrwxr-x 2 srtplay srtplay         51  1월   7  2021 .
drwxrwxr-x 4 srtplay srtplay         30  1월   7  2021 ..
-rw-r--r-- 1 srtplay srtplay 1438807183  5월  18  2020 dump_0515.0834.sql
-rw-rw-r-- 1 srtplay srtplay 1592073446  5월  28  2020 dump_0527.sql
```

② 필드별 의미



1	2	3	4	5	6	7	8	9	10
-	R	W	-	R	-	-	R	-	-

- 1번 필드 : 타입
  - 파일 : -
  - 디렉터리 : d
  - 다른 파일을 가리키는 링크 : l
  - 두 개의 프로그램을 연결하는 파이프 파일 : p
  - 블록 단위로 하드웨어와 반응하는 파일 : b
  - 스트림 단위로 하드웨어와 반응하는 파일 : c
- 2~4번 필드 : 소유주(USER) 권한
- 5~7번 필드 : 그룹(GROUP) 권한
- 8~10번 필드 : 나머지(OTHER) 권한

③ 권한별 값

구분	값	설명
R	4	읽기 권한
W	2	쓰기 권한
X	1	실행 권한
-	0	권한 없음

④ 권한 변경

- chmod 0751 file명
  - 해당 파일에 대해서 소유주는 읽기, 쓰기, 실행 권한이 있다.
  - 해당 파일에 대해서 그룹은 읽기, 실행 권한이 있다.
  - 해당 파일에 대해서 나머지는 실행 권한이 있다.
- chmod 0775 file명
  - 해당 파일에 대해서 소유주는 읽기, 쓰기, 실행 권한이 있다.
  - 해당 파일에 대해서 그룹은 읽기, 쓰기, 실행 권한이 있다.
  - 해당 파일에 대해서 나머지는 읽기, 실행 권한이 있다.

⑤ umask (접근 권한 마스크)

- 앞으로 만들어질 파일 권한에 대한 설정
- umask 로 지정한 8진수는 새로 만들어질 파일에서 제거될 권한을 명시
- umask 022
  - 앞으로 만들어지는 파일은 644, 디렉터리는 755 권한을 가진다.
  - 소유주는 읽기, 쓰기권한
  - 그룹은 읽기 권한
  - 나머지는 읽기 권한

⑥ chown (소유주 변경)

- chown hungjik file명
  - 해당 파일의 소유주를 hungjik 로 변경

Section 2. 메모리 관리

1. 기억장치 관리 전략

(1) 기억장치 관리 전략의 개념

- 보조기억장치의 프로그램이나 데이터를 주기억장치에 적재시키는 시기, 위치 등을 지정하여, 한정된 주기억장치의 공간을 효율적으로 사용한다.

(2) 기억장치 관리 전략

① 반입(Fetch) 전략

- 보조기억장치에 보관중인 프로그램이나 데이터를 언제 주기억장치로 적재할 것인지를 결정하는 전략

요구 반입 (Demand Fetch)	- 실행중인 프로그램이 특정 프로그램이나 데이터 등의 참조를 요구할 때 적재하는 방법
예상 반입 (Anticipatory)	- 실행중인 프로그램에 의해 참조될 프로그램이나 데이터를 미리 예상하여 적재하는 방법

② 배치(Placement) 전략

- 새로 반입되는 프로그램이나 데이터를 주기억장치의 어디에 위치시킬 것인지를 결정하는 전략

최초 적합 (First Fit)	- 프로그램이나 데이터가 들어갈 수 있는 크기의 빈 영역 중에서 첫 번째 분할 영역에 배치
최적 적합 (Best Fit)	- 프로그램이나 데이터가 들어갈 수 있는 크기의 빈 영역 중에서 단편화를 가장 작게 남기는 분할 영역에 배치
최악 적합 (Worst Fit)	- 프로그램이나 데이터가 들어갈 수 있는 크기의 빈 영역 중에서 단편화를 가장 많이 남기는 분할 영역에 배치

- 다음 메모리 상태표에서 16K 의 프로그램을 할당하였을 때, 배치 전략은 아래와 같다.

<<메모리 상태표>>

영역 번호	크기
1	20K
2	16K
3	8K
4	40K

<<배치전략>>

배치전략	영역 번호
최초 적합	1
최적 적합	2
최악 적합	4

③ 교체(Replacement) 전략

- 주기억장치의 모든 영역이 이미 사용중인 상태에서 새로운 프로그램이나 데이터를 주기억장치에 배치하려고 할 때, 이미 사용되고 있는 영역 중에서 어느 영역을 교체하여 사용할 것인지를 결정하는 전략
- 종류 : FIFO, OPT, LRU, LFU, NUR, SCR 등

2. 주기억장치 할당 기법

(1) 단일 분할 할당 기법

- 주기억장치의 사용시, 한 순간에는 오직 한 명의 사용자만이 주기억장치의 사용자 영역을 사용하는 기법
- 가장 단순한 기법으로 초기의 운영체제에서 많이 사용했던 기법
- 운영체제를 보호하고, 프로그램이 사용자 영역만을 사용하기 위해 운영체제 영역과 사용자 영역을 구분하는 경계 레지스터(Boundary Register)가 사용 된다.
- 프로그램의 크기가 작을 경우 사용자 영역이 낭비 될 수 있다.
- 주기억장치보다 큰 프로그램을 실행하기 위해 오버레이 기법과 스와핑 기법을 사용함

오버레이 (Overlay)	- 보조기억장치에 저장된 프로그램을 여러 개의 조각으로 분할한 후 필요한 조각을 차례로 주기억 장치에 적재시켜 프로그램을 실행 - 주기억 장치의 공간이 부족하면 불필요한 조각이 위치한 장소에 새로운 프로그램 조각을 중첩(Overlay) 하여 적재 - 프로그램을 조각으로 분할하는 작업은 프로그래머가 수행하므로 시스템 구조나 프로그램 구조를 알아야 한다.
스와핑 (Swapping)	- 하나의 프로그램 전체를 주기억장치에 할당하여 사용하다 필요에 따라 다른 프로그램과 교체하는 기법 - 가상기억장치의 페이징 기법으로 발전

(2) 다중 분할 할당 기법

① 고정 분할 할당 기법

- 주기억장치의 사용자 영역을 여러 개의 고정된 크기로 분할하고, 준비상태 큐에서 준비중인 프로그램을 각 영역에 할당하여 수행하는 기법
- 프로그램을 실행하려면 프로그램 전체가 주기억장치에 위치해야 한다.
- 프로그램이 분할된 영역보다 커서 들어갈 수 없는 경우가 발생할 수 있다.
- 내부 단편화 및 외부 단편화가 발생하여 주기억장치의 낭비가 많다.
- 실행할 프로그램의 크기를 미리 알고 있어야 한다.

② 가변 분할 할당 기법

- 고정 분할 할당 기법의 단편화를 줄이기 위한 기법
- 프로그램을 주기억장치에 적재하면서 필요한 만큼의 크기로 영역을 분할하는 기법
- 주기억장치를 효율적으로 사용할 수 있으며, 다중 프로그래밍의 정도를 높일 수 있다
- 단편화를 상당 부분 해결하였으나, 영역과 영역 사이에 단편화가 발생한다.

3. 단편화

(1) 단편화의 개념

- 주기억장치에 프로그램을 할당하고 반납하는 과정에서 발생하는 사용되지 않는 작은 조각 공간
- 주기억장치 상에서 빈번하게 기억장소가 할당되고 반납됨에 따라 기억장소들이 조각들로 나누어지는 현상

(2) 단편화의 종류

① 내부 단편화

- 주기억장치 공간이 프로그램보다 커서 프로그램의 사용 공간을 할당 후 사용되지 않고 남아있는 공간

② 외부 단편화

- 주기억장치 공간 보다 프로그램이 커서 프로그램이 할당될 수 없어 사용되지 않고 남아있는 공간

③ 단편화 계산

영역	분할 크기	작업 크기	단편화 크기
1	20K	10K	내부:10K
2	50K	60K	외부:50K
3	120K	160K	외부:120K
4	200K	100K	내부:100K
5	300K	150K	내부:150K

(3) 단편화 해결 방법

① 통합(Coalescing) 기법

- 인접해 있다면 두 개의 빈 분할 공간을 하나로 통합하여 효율성을 높이는 작업

<<메모리 상태표>>

영역 번호	상태	상태
1	Used	20
2	Free	20
3	Free	10
4	Used	30
5	Free	10

<<통합 후>>

영역 번호	상태	상태
1	Used	20
2	Free	30
3	Used	30
4	Free	10

② 압축(Compaction) 기법

- 주기억장치 내 분산되어 있는 단편화 공간들을 통합하여 하나의 커다란 빈 공간을 만드는 작업
- 가비지 컬렉션(Garbage Collection) 작업 이라고도 함

〈〈메모리 상태표〉〉

영역 번호	상태	상태
1	Used	20
2	Free	20
3	Free	10
4	Used	30
5	Free	10

〈〈압축 후〉〉

영역 번호	상태	상태
1	Used	20
2	Used	30
3	Free	40

③ 재배치 기법(Relocation)

- 압축을 실행하여 이 과정에서 프로그램의 주소를 새롭게 지정해주는 기법

Section 3. 가상 기억 장치

1. 가상 기억 장치

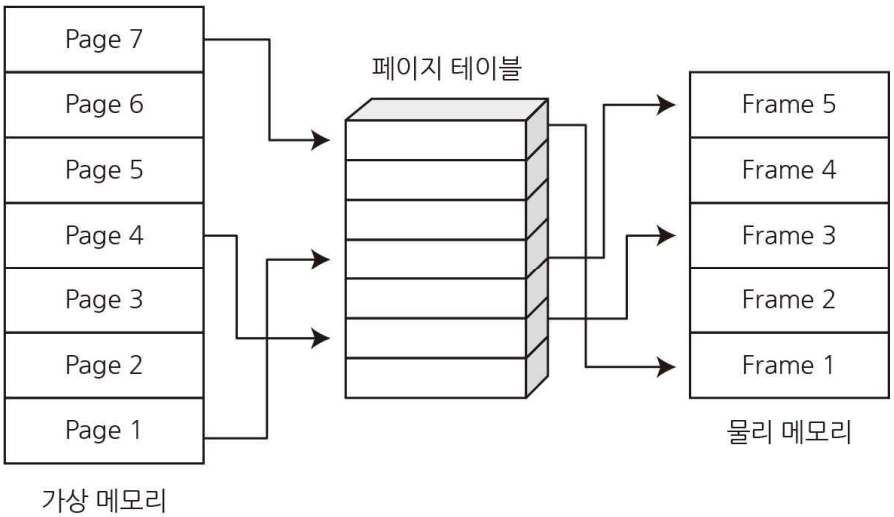
(1) 가상 기억 장치의 개념

- 보조기억장치(하드디스크)의 일부를 주기억장치처럼 사용하는 것
- 용량이 작은 주기억장치를 마치 큰 용량을 가진 것처럼 사용하는 기법
- 프로그램을 여러 개의 작은 블록 단위로 나누어서 가상기억장치에 보관해 놓고, 프로그램 실행 시 요구되는 블록만 주기억장치에 불연속적으로 할당하여 처리
- 주기억장치의 용량보다 큰 프로그램을 실행하기 위해 사용
- 주기억장치의 이용률과 다중 프로그래밍의 효율을 높일 수 있다.
- 가상기억장치에 저장된 프로그램을 실행하려면 가상기억장치의 주소를 주기억장치의 주소로 바꾸는 주소 변환 작업이 필요
- 블록 단위로 나누어 사용하므로 연속 할당 방식에서 발생할 수 있는 단편화를 해결할 수 있다.

(2) 블록 분할 방법

① 페이징(Paging) 기법

- 가상기억장치를 모두 같은 크기의 블록으로 편성하여 운용하는 기법
- 외부 단편화는 발생하지 않으나 내부 단편화는 발생
- 가상 메모리를 일정한 크기로 나눈 단위를 페이지(Page) 라고 하고, 물리 메모리를 일정한 크기로 나눈 블록을 프레임(Frame) 이라고 한다.
- 주소 변환을 위해서 페이지 맵 테이블이 필요



- 페이지 크기별 비교

페이지 크기	기억장소 효율	단편화	입출력 시간	맵 테이블
클수록	감소	증가	감소	감소
작을수록	증가	감소	증가	증가

② 세그먼테이션(Segmentation) 기법

- 가상 메모리를 서로 크기가 다른 논리적 단위인 세그먼트로 분할하고 메모리를 할당하는 기법
- 세그먼트 테이블을 참조하여 해당 세그먼트의 시작주소와 더해져서 실제적인 물리적 위치로 변환
- 세그먼테이션 기법 사용 이유는 기억공간을 절약하기 위함
- 세그먼트가 주기억장치에 적재될 때 다른 세그먼트에게 할당된 영역을 침범할 수 없으며, 이를 위해 기억장치 보호키(Storage Protection Key)가 필요
- 내부 단편화는 발생하지 않으나, 외부 단편화가 발생
- 주소 변환을 위해서 세그먼트 맵 테이블이 필요
- 아래 세그먼트 테이블에서 S=(2, 100) 의 실제 주소는 2100 임

세그먼트 번호	크기	시작주소
0	1200	4000
1	800	5700
2	1000	2000
3	500	3200

2. 가상기억장치 기타 관리 사항

(1) 페이지 크기

① 페이지 크기가 작을 경우

- 페이지 단편화가 감소되고, 한 개의 페이지를 주기억장치로 이동하는 시간이 줄어든다.
- 불필요한 내용이 주기억장치에 적재될 확률이 적으므로 효율적인 워킹 셋을 유지할 수 있다.
- 지역성(Locality)에 더 일치할 수 있기 때문에 기억장치 효율이 높아진다.
- 페이지 정보를 갖는 페이지 맵 테이블의 크기가 커지고, 매핑 속도가 늦어진다.
- 디스크 접근 횟수가 많아져 전체적인 입/출력 시간이 늘어난다.

② 페이지 크기가 클 경우

- 페이지 정보를 갖는 페이지 맵 테이블의 크기가 작아지고, 매핑 속도가 빨라진다.
- 디스크 접근 횟수가 줄어들어 전체적인 입/출력 효율성이 증가된다.
- 페이지 단편화가 증가되고, 한 개의 페이지를 주기억장치로 이동하는 시간이 늘어난다.
- 불필요한 내용까지도 주기억장치에 적재될 수 있다.

(2) 페이지 부재

- 프로세스 실행 시 참조할 페이지가 주기억 장치에 없는 현상
- 페이지 부재가 일어나는 횟수를 페이지 부재 빈도라고 한다.
- 운영체제는 프로세스 실행 초기에 임의의 페이지 프레임을 할당하고, 페이지 부재율을 지속적으로 감시하고 있다가 부재율이 상한선을 넘어가면 좀 더 많은 페이지 프레임을 할당하고, 부재율이 하한선을 넘어가면 페이지 프레임을 회수하는 방식을 사용

⑶ 지역성(Locality)

- 프로세스가 실행되는 동안 주기억장치를 참조할 때 일부 페이지만 집중적으로 참조하는 성질
- 스레싱을 방지하기 위한 워킹 셋 이론의 기반
- 지역성의 종류

시간 구역성 (Temporal Locality)	- 프로세스가 실행되면서 하나의 페이지를 일정 시간 동안 집중적으로 액세스 하는 현상 - 한 번 참조한 페이지는 가까운 시간 내에 계속 참조할 가능성이 높음 - Loop(반복), Stack(스택), 부 프로그램(Sub Routine) 등
공간 구역성 (Spatial Locality)	- 프로세스 실행 시 일정 위치의 페이지를 집중적으로 액세스 하는 현상 - 어느 하나의 페이지를 참조하면 그 근처의 페이지를 계속 참조할 가능성이 높음 - 배열순회, 순차적 코드 실행 등

⑷ 워킹 셋(Working Set)

- 프로세스가 일정 시간 동안 자주 참조하는 페이지들의 집합
- 자주 참조되는 워킹 셋을 주기억 장치에 상주시킴으로 페이지 부재 및 페이지 교체 현상을 줄여 프로세스의 기억장치 사용이 안정되게 한다.
- 시간이 지남에 따라 자주 참조하는 페이지들의 집합이 변하기 때문에 워킹셋은 시간에 따라 변함

⑸ 스래싱(Thrashing)

- 프로세스의 처리 시간보다 페이지 교체에 소요되는 시간이 더 많아지는 현상
- 가상기억장치를 사용하는 시스템에서 하나의 프로세스 수행 과정 중 자주 페이지 부재가 발생함으로써 나타나는 현상으로, 전체 시스템의 성능이 저하됨
- 스래싱 현상 방지 방법
  - 다중 프로그래밍의 정도를 적정 수준으로 유지
  - 페이지 부재 빈도를 조절하여 사용
  - 워킹 셋을 유지
  - 부족한 자원을 증설하고, 일부 프로세스 중단 시킴
  - CPU 성능에 대한 자료의 지속적인 관리 및 분석으로 임계치를 예상하여 운영

3. 페이지 교체 알고리즘

⑴ FIFO (First In First Out)

- 가장 먼저 메모리에 적재된 페이지를 먼저 교체하는 기법
- 프레임 개수를 늘리면 부재 발생이 감소해야 하나, 오히려 더 늘어나는 Belady's Anomaly 이상현상 발생
- 페이지 프레임이 3개일 때, 페이지 결함 예

참조 페이지	1	2	3	1	2	4	1	2	5
페이지 프레임	1	1	1	1	1	4	4	4	5
		2	2	2	2	2	1	1	1
			3	3	3	3	3	2	2
페이지 부재	O	O	O	X	X	O	O	O	O

(2) OPT (Optimal replacement, 최적 교체)

- 앞으로 가장 사용 안 될 페이지를 교체
- 페이지 부재 횟수가 가장 적게 발생하는 가장 효율적인 알고리즘이나 참조 상황을 예측하기 어려움

(3) LRU (Least Recently Used)

- 최근에 가장 오랫동안 사용되지 않은 페이지를 교체
- 페이지 프레임이 3개일 때, 페이지 결함 예

참조 페이지	1	2	1	0	4	1	3
페이지 프레임	1	1	1	1	1	1	1
		2	2	2	4	4	4
				0	0	0	3
페이지 부재	O	O	X	O	O	X	O

(4) LFU (Least Frequently Used)

- 사용 빈도가 가장 적은 페이지를 교체
- 페이지 프레임이 3개일 때, 페이지 결함 예

참조 페이지	1	2	3	1	2	4	1	2	5
페이지 프레임	1	1	1	1	1	1	1	1	1
		2	2	2	2	2	2	2	2
			3	3	3	4	4	4	5
페이지 부재	O	O	O	X	X	O	X	X	O

(5) NUR(Not Used Recently)

- 최근의 사용여부를 확인하기 위해 각 페이지마다 두 개의 비트 사용
- 참조비트와 변형비트를 이용해서 페이지 교체
- 다음 중 가장 나중에 교체될 페이지는 4번

번호	1	2	3	4
참조비트	0	0	1	1
변형비트	0	1	0	1

(6) SCR(Second Chance Replacement)

- 가장 오랫동안 주기억장치에 있던 페이지 중 자주 사용되는 페이지의 교체를 방지하기 위한 것으로, FIFO 기법의 단점을 보완하는 기법

Section 4. 프로세스

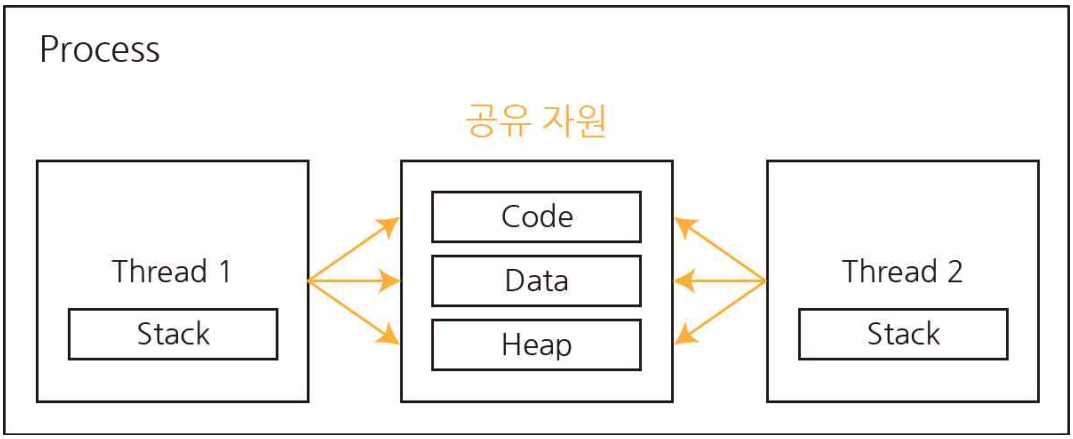
1. 프로세스

(1) 프로세스의 개념

- 컴퓨터에서 연속적으로 실행되고 있는 컴퓨터 프로그램
- 실행 가능한 PCB 가 있는 프로그램
- 운영체제가 관리하는 실행단위
- 프로세서가 할당되는 실체
- 활동 중인 프로시저

(2) 스레드(Thread)

① 스레드의 개념



- 프로세스 내에서 실행되는 흐름의 단위
- 프로그램은 하나 이상의 프로세스를 가지고 있고, 하나의 프로세스는 반드시 하나 이상의 스레드를 갖는다.
- 프로세스의 일부 특성을 가지고 있어 경량 프로세스라고도 한다.
- 스레드는 한 프로세스 내에서 동작되는 흐름으로, Stack영역만 별도로 할당 받고, 부모 프로세스의 Code, Data, Head영역은 공유한다.

② 스레드의 특징

- 프로그램의 일부분이 중단되어도 프로그램이 지속적으로 수행되어 응답성이 좋다.
- 스레드들은 부모 프로세서의 자원과 메모리를 공유하여 자원 공유가 쉽다.
- 프로세스를 할당하는 것보다 스레드를 할당하는 것이 비용이 적게 든다.
- 멀티프로세서 구조에서 각각의 스레드가 다른 프로세서에서 병렬로 수행될 수 있다.
- 구현 및 디버깅이 어렵다.
- 동기화 및 교착상태가 발생하지 않도록 주의해야 한다.

③ 스레드의 분류

- 사용자 수준의 스레드
  - 사용자가 만든 라이브러리를 사용하여 스레드를 운용한다.
  - 속도는 빠르지만 구현이 어렵다.

- 커널 수준의 스레드
  - 운영체제의 커널에 의해 스레드를 운용한다.
  - 구현이 쉽지만 속도가 느리다.

(3) 메모리상의 프로세스 영역

① 코드 영역

- 실행할 프로그램의 코드가 저장되는 영역
- 함수, 제어문, 상수 등이 지정

② 데이터 영역

- 전역변수와 정적변수(static) 변수가 할당 되는 부분
- 프로그램이 종료되면 메모리에서 소멸

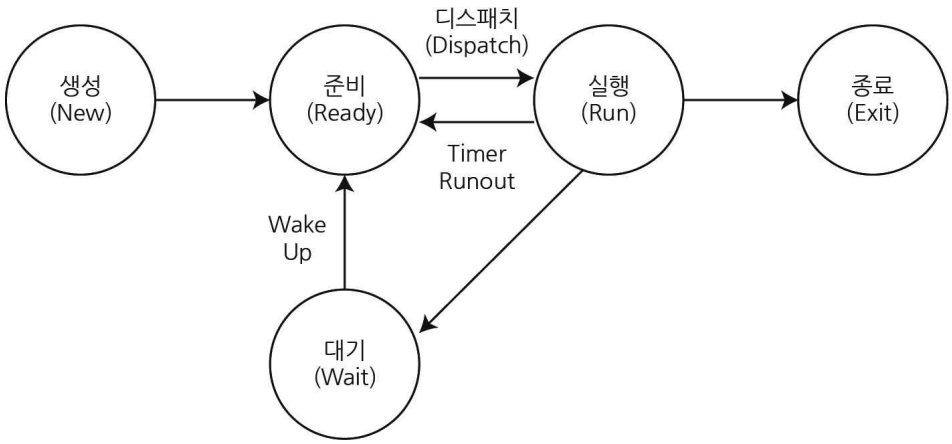
③ 힙 영역

- 프로그래머가 할당/해제하는 메모리 공간
- 동적 할당

④ 스택 영역

- 프로그램이 자동으로 사용하는 임시 메모리 영역
- 지역변수와 함수의 매개변수가 저장되는 영역으로 함수 호출이 완료되면 사라짐

(4) 프로세스 상태 전이



① 프로세스 상태 종류

- 제출(Submit)
  - 작업을 처리하기 위해 사용자가 작업을 시스템에 제출한 상태
- 접수(Hold)
  - 제출된 작업이 Spool 공간인 디스크의 할당 위치에 저장된 상태
- 준비(Ready)
  - 프로세스가 대기큐에서 프로세서를 할당받기 위해 기다리는 상태
  - 접수상태에서 준비상태로의 전이는 Job Scheduler에 의해 수행

- 실행(Running)
  - 프로세스가 프로세서를 할당받아 실행되는 상태
  - 프로세스 수행이 완료되기 전에 할당 시간이 종료되면 프로세스는 다시 준비 상태로 전이
  - 준비상태에서 실행상태로의 전이는 CPU Scheduler에 의해 수행됨(Dispatch)
- 대기(Wait)
  - 프로세스에 I/O 처리가 필요하여, 실행을 중단 시키고, I/O 처리가 끝날 때까지 대기중인 상태
  - I/O 가 완료되면 다시 준비상태로 전이 (Wake Up)
- 종료(Exit)
  - 프로세스의 실행이 끝나고 프로세스 할당이 해제된 상태

② 프로세스 상태 전이 용어

- Dispatch
  - 준비상태에서 실행상태로 전이되는 과정
- Wake Up
  - 대기상태에서 준비상태로 전이되는 과정
- Spooling
  - 입/출력 데이터를 직접 입/출력 장치에 보내지 않고, 모아뒀다가 한꺼번에 입/출력하기 위해 디스크에 저장해 놓는 과정

(5) 문맥 교환(Context Switching)

- 하나의 프로세스가 CPU를 사용 중인 상태에서 다른 프로세스가 CPU를 사용하도록 하기 위해 이전의 프로세스의 상태를 PCB에 보관하고 또 다른 프로세스의 정보를 PCB에서 읽어 레지스터에 적재하는 과정
- 문맥 교환은 멀티태스킹(=멀티프로세싱)이 가능하도록 해준다.
- 하나의 CPU에서 여러 개의 프로세스가 동시에 수행되는 것처럼 보이는 이유는 문맥 교환이 빠르게 일어나기 때문이다.
- 문맥 교환이 일어나는 시점
  - 멀티 태스킹
  - 인터럽트 처리
  - 사용자 및 커널 모드 전환

(6) PCB (Process Control Block, 프로세스 제어 블록)

- 운영체제가 프로세스에 대한 정보를 저장해 놓는 공간
- 각 프로세스가 생성될 때 마다 고유한 PCB가 생성되고, 프로세스가 완료되면 해당 PCB는 제거
- PCB에 저장되는 정보
  - 프로세스의 현재 상태
  - 포인터(부모 프로세스, 자식 프로세스 등)
  - 프로세스 고유 식별자
  - 스케줄링, 프로세스 우선순위
  - CPU 레지스터 정보
  - 주기억장치 관리 정보
  - I/O 상태정보
  - 계정정보(CPU 사용 시간, 실제 사용시간, 한정된 시간)

2. 프로세스 스케줄링

(1) 스케줄링(Scheduling)의 개념

- 메모리에 올라온 프로세스들 중 어떤 프로세스를 먼저 처리할지 순서를 정하는 것
- Ready Queue 에 있는 프로세스들 중 누구에게 CPU를 할당해 줄 것인지 정하는 것

(2) 스케줄링의 목적

- 공평성
  - 모든 프로세스가 자원을 공평하게 배정 받아야 하며, 특정 프로세스가 배제되어서는 안된다.
- 효율성
  - 시스템 자원을 최대한 활용하여 스케줄링 해야 한다.
- 안정성
  - 중요한 프로세스가 먼저 처리 되도록 한다.
- 반응 시간 보장
  - 적절한 시간 안에 프로세스의 요구에 반응해야 한다.
- 무한 연기 방지
  - 특정 프로세스의 작업이 무한하게 연기 되어서는 안된다.

(3) 스케줄링 성능척도

① 프로세서 차원

- CPU 사용률
- Throughput (처리량)

② 프로세스 차원

- 응답시간 (Response Time) : 프로세스가 대기 상태에 들어와 CPU를 최초로 얻기까지 걸리는 시간
- 반환시간 (Turn-around Time) : 프로세스가 생성되어 종료된 후 자원을 모두 반환하는데 걸리는 시간
- 대기시간 (Waiting Time) : 프로세스가 CPU를 할당 받기 전 대기 상태일 때의 시간

(4) 스케줄링 기법

① 선점형 스케줄링 (Preemptive)

- 다른 프로세스가 실행 중이더라도 운영체제가 CPU를 강제로 뺏을 수 있는 방식
- CPU 처리 시간이 매우 긴 프로세스가 CPU 사용 독점을 막을 수 있어 효율적인 운영이 가능
- 잦은 문맥 교환으로 오버헤드가 많이 발생
- 종류 : Round Robin, SRT, 다단계 큐, 다단계 피드백 큐 등

② 비선점형 스케줄링 (Non-Preemptive)

- 프로세스가 CPU를 점유하고 있다면 이를 빼앗을 수 없는 방식
- 순서대로 처리되는 공정성 보장
- 다음에 처리해야 할 프로세스와 관계없이 응답시간을 예상할 수 있음
- 낮은 문맥 교환으로 오버헤드가 적다.
- CPU 사용시간이 짧은 여러 프로세스가 오래 기다릴 수 있어, 처리율이 떨어질 수 있다.
- 종류 : FCFS, SJF, HRN, 우선순위, 기한부 등

3. 스케줄링 알고리즘

(1) 선점형 기법

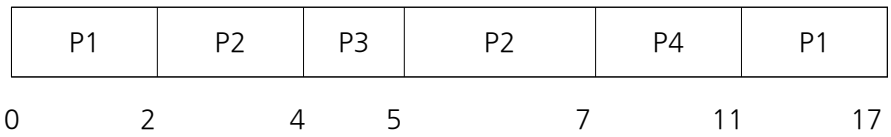
① Round Robin

- 프로세스들 사이에 우선순위를 두지 않고, 순서대로 시간단위(Time Quantum/Slice)로 CPU를 할당하는 방식
- 컴퓨터 자원을 사용할 수 있는 기회를 프로그램 프로세스들에게 공정하게 부여하기 위한 방법

② SRT(Shortest Remaining Time)

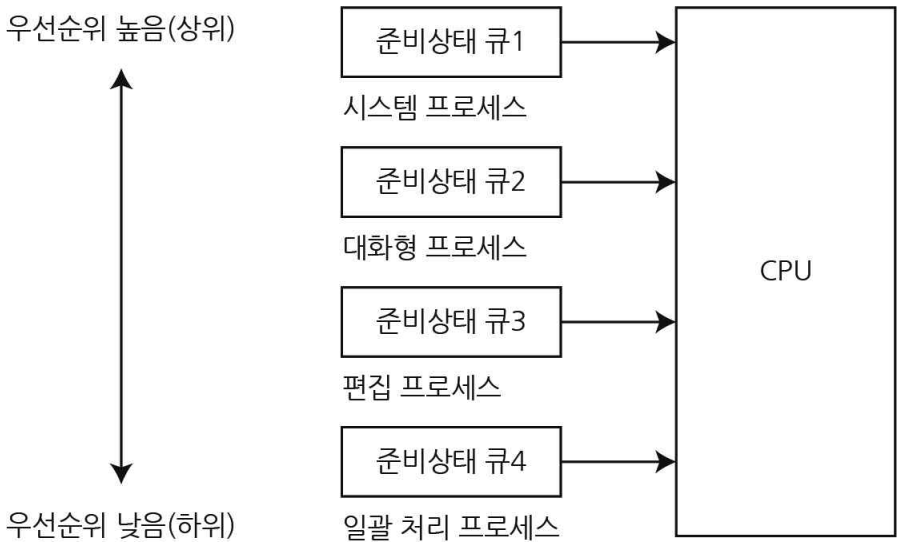
- 비선점 스케줄링인 SJF 기법을 선점 형태로 변경한 기법
- CPU 점유 시간이 가장 짧은 프로세스에 CPU를 먼저 할당하는 방식
- SRT 스케줄링의 프로세스 처리

프로세스	도착 시간	실행 시간	반환 시간	평균 반환 시간
P1	0	8	17 - 0 = 17	28 / 4 = 7
P2	2	4	7 - 2 = 5	
P3	4	1	5 - 4 = 1	
P4	6	4	11 - 6 = 5	



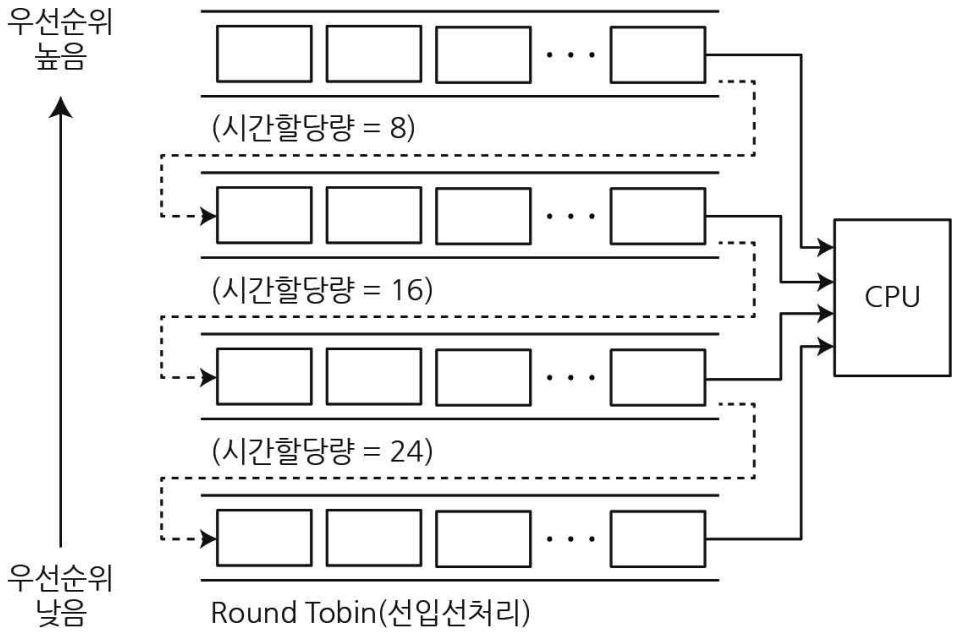
③ 다단계 큐(MQ, Multi-level Queue)

- 프로세스를 특정 그룹으로 분류할 수 있을 경우 그룹에 따라 각기 다른 준비 상태 큐를 사용하는 기법
- 특정 그룹의 준비상태 큐에 들어갈 경우 다른 준비상태 큐로 이동할 수 없다.
- 하위 준비 상태 큐에 있는 프로세스를 실행하는 도중이라도 상위 준비 상태 큐에 프로세스가 들어오면 상위 프로세스에게 CPU를 할당



④ 다단계 피드백 큐 (Multi-Level Feedback Queue)

- 프로세스 생성 시 가장 높은 우선 순위 준비 큐에 등록되며 등록된 프로세스는 FCFS 순서로 CPU를 할당받아 실행되고, 할당된 시간이 끝나면 다음 단계의 준비큐로 이동
- 단계가 내려갈수록 시간 할당량이 증가하고, 가장 하위큐는 Round Robin 방식으로 운영



(2) 비 선점형 기법

① FCFS (First Come , First Serve)

- 먼저 도착한 프로세스를 먼저 처리하는 스케쥴링 알고리즘
- 공평성이 유지되지만, 중요한 작업이 중요치 않은 작업에 밀려서 늦게 처리되는 문제점 발생
- 실행시간이 긴 프로세스가 먼저 도착하게 되면, 효율성이 떨어지는 현상 발생
- FCFS의 평균 반환 시간 예제

프로세스	실행시간
P1	9
P2	3
P3	12

	최대 평균 시간 (반환시간)	최소 평균 시간 (반환시간)
1번 수행	P3 (12)	P2 (3)
2번 수행	P1 (21)	P1 (12)
3번 수행	P2 (24)	P3 (24)
평균 반환 시간	57 / 3 = 19	39 / 3 = 13

② SJF (Shortest Job First)

- 실행시간이 가장 짧은 프로세스에게 CPU를 할당하는 방식
- 짧은 작업들을 우선적으로 처리하다 보니, 평균 대기시간이 가장 적은 최적의 스케줄링
- 기아현상이 발생할 수 있다.
- SJF의 반환 시간 예제

프로세스	실행시간	수행순서	대기시간	반환시간
P1	6	2	3	9
P2	3	1	0	3
P3	8	4	16	24
P4	7	3	9	16
평균			7	13

③ HRN (Highest Response Ratio Next)

- SJF 기법에서 비교적 실행시간이 긴 프로세스가 가질 수 있는 불리함을 보완한 스케줄링 방식
- 우선순위를 계산해서 프로세스들에게 순서를 부여하는 방식
- 우선순위 = (대기시간 + 실행시간) / 실행시간
- 실행시간이 짧은 프로세스에게 밀려서 오랫동안 기다린 프로세스가 더 높은 우선순위를 가지게 된다.
- HRN 우선순위 예제

프로세스	대기시간	실행시간	우선순위
P1	5	20	4
P2	40	20	1
P3	15	45	3
P4	20	20	2

④ 우선순위(Priority)

- 프로세스마다 우선순위를 부여하여 높은 우선순위를 가진 프로세스에게 먼저 자원을 할당
- 우선순위가 낮을 경우 기아상태가 일어날 수 있음

⑤ 기한부(Deadline)

- 프로세스에게 일정한 시간을 주어 그 시간 안에 완료하도록 하는 기법
- 주어진 시간 내에 완료되지 못할 경우, 해당 프로세스는 제거되거나 처음부터 다시 실행되어야 하므로 부담이 매우 큰 기법

Section 5. 병행 프로세스와 교착상태

1. 병행 프로세스

⑴ 병행 프로세스의 개념

- 두 개 이상의 프로세스들이 동시에 존재하며 실행 상태에 있는 것

⑵ 문제점과 해결책

① 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

② 문제 해결책

- 임계구역
- 상호배제 기법
- 동기화 기법

2. 병행 프로세스 문제 해결책

⑴ 임계구역(Critical Section)

- 공유 자원에 대해서, 한 순간에는 반드시 하나의 프로세스만 사용되도록 지정한 영역
- 병렬컴퓨팅에서 둘 이상의 스레드가 동시에 접근해서는 안되는 공유 자원
- 특징
  - 특정 프로세스가 독점할 수 없다.
  - 프로세스가 임계구역에 대한 진입을 요청하면 일정 시간 내에 진입을 허락해야 한다.

⑵ 상호 배제(Mutual Exclusion)

- 하나의 프로세스가 공유 메모리 혹은 공유 파일을 사용하고 있을 때 다른 프로세스들이 사용하지 못하도록 배제시키는 제어 기법
- 상호 배제 기법
  - 데커의 알고리즘 (Dekker's algorithm)
  - 피터슨의 알고리즘 (Peterson's algorithm)
  - 다익스트라 알고리즘 (Dijkstra algorithm)
  - 램포트의 베어커리 알고리즘 (Lamport's bakery algorithm)

⑶ 동기화 기법

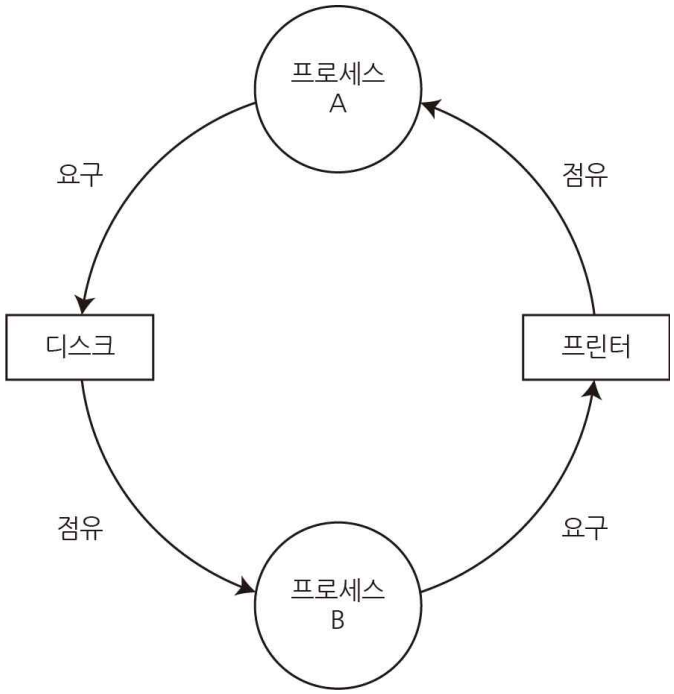
- 스레드들에게 하나의 자원에 대한 처리 권한을 주거나 순서를 조정해주는 기법
- 세마포어(Semaphore)
  - 각 프로세스에 제어 신호를 전달하여 순서대로 작업을 수행하도록 하는 기법
  - 다익스트라(Dijkstra) 가 제안
  - P와 V라는 2개의 연산에 의해서 동기화를 유지시키고, 상호 배제의 원리를 보장
  - P는 임계 구역에 들어가기 전에 수행되고, V는 임계 구역에서 나올 때 수행
  - 종류 : 계수 세마포어, 이진 세마포어

- 모니터(Monitor)
  - 프로그래밍 언어 수준에서 동시성을 제어하여 타이밍 오류를 해결한 상호 배제 기법
  - 모니터 내의 자원을 공유하려면 프로세스는 반드시 모니터의 진입부를 호출해야 함
  - 모니터 외부의 프로세스는 모니터 내부의 데이터를 직접 액세스 할 수 없음
  - Wait와 Signal 연산이 사용

3. 교착상태(Dead Lock)

(1) 교착상태의 개념

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



(2) 교착상태 발생 조건

- 상호배제(Mutual Exclusion)
  - 한 번에 한 개의 프로세스만이 공유 자원을 사용할 수 있어야 함
- 점유와 대기(Hold & Wait)
  - 자원을 점유하고 있으면서 다른 프로세스에 할당되어 있는 자원을 추가로 요구하며 대기
- 비선점(nonpreemption)
  - 프로세스에 할당된 자원은 사용이 끝날 때까지 강제로 빼앗을 수 없음
- 환형대기(Circular Wait)
  - 각 프로세스가 순차적으로 다음 프로세스가 요구하고 있는 자원을 가지고 있는 상태

⑶ 교착상태 해결 방법

- 예방 기법(Prevention)
  - 교착 상태가 발생되지 않도록 사전에 시스템을 제어하는 방법
- 회피 기법(Avoidance)
  - 교착 상태가 발생하려고 할 때, 교착상태 가능성을 피해가는 방법
  - 주로 은행원 알고리즘(Banker's Algorithm)이 사용
- 발견 기법(Detection)
  - 시스템에 교착 상태가 발생했는지 점검하여 교착 상태에 있는 프로세스와 자원을 발견하는 것
- 회복 기법(Recovery)
  - 교착상태의 프로세스에 할당된 자원을 선점하여 프로세스나 자원을 회복하는 것
  - 교착 상태를 일으킨 프로세스를 종료시킴으로 선점을 해제한다.

Section 6. 디스크 스케줄링(Disk Scheduling)

1. 디스크 스케줄링

(1) 디스크 스케줄링 개념

- 사용할 데이터가 디스크 상의 여러 곳에 저장되어 있을 경우, 데이터를 액세스하기 위해 디스크 헤드를 움직이는 경로를 결정하는 기법
- 효율적으로 데이터를 액세스 하는 방법

(2) 디스크 스케줄링 목표

- 하드 디스크 검색으로 낭비되는 시간을 최소화
- 특정한 프로세스의 입출력 요청의 우선순위를 정함
- 디스크 대역을 실행중인 각 프로세스에 할당
- 정해진 기한까지 요청을 처리

(3) 디스크 스케줄링 종류

① FCFS 스케줄링(First Come First Served)

- 요청이 들어온 순서대로 처리
- 장점
  - 알고리즘이 다른 기법보다 단순하며, 공평하게 요청을 처리
- 단점
  - 비용이 많이 발생되어, 비효율적임
- FCFS 적용 예(현재 헤드의 위치가 53이라고 가정할 경우)

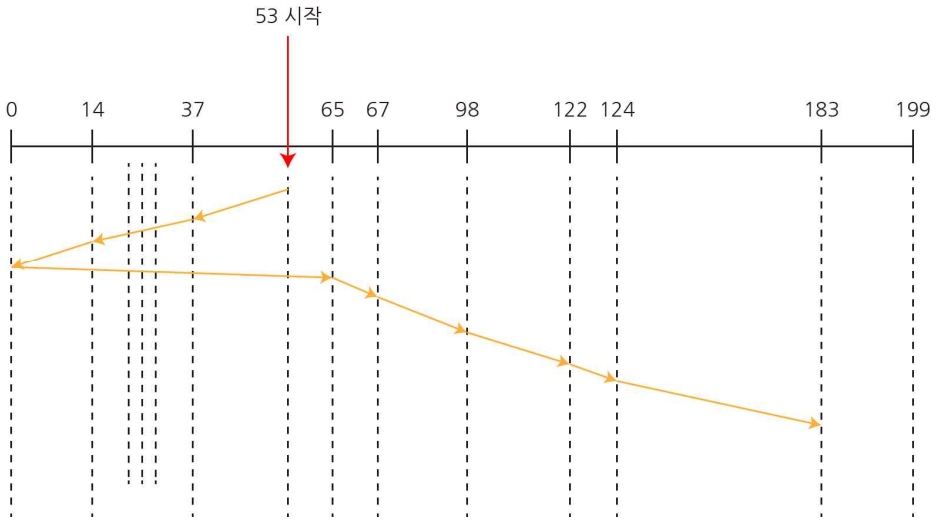
큐의 내용	98	183	37	122	14	124	65	67	합계
이동 순서	1	2	3	4	5	6	7	8	
이동 거리	45	85	146	85	108	110	59	2	640

② SSTF(Shortest Seek Time First)

- 현재 헤드에서 가장 가까운 트랙의 요청을 먼저 처리한다.
- 장점
  - Seek Time이 적다.
  - 트랙을 찾는 시간을 최소화 할 수 있다.
  - 처리량(Throughput)을 극대화 할 수 있다.
- 단점
  - 안쪽 및 바깥쪽에 있는 요청들은 기아 현상이 발생할 수 있다.
- SSTF 적용 예(현재 헤드의 위치가 53이라고 가정할 경우)

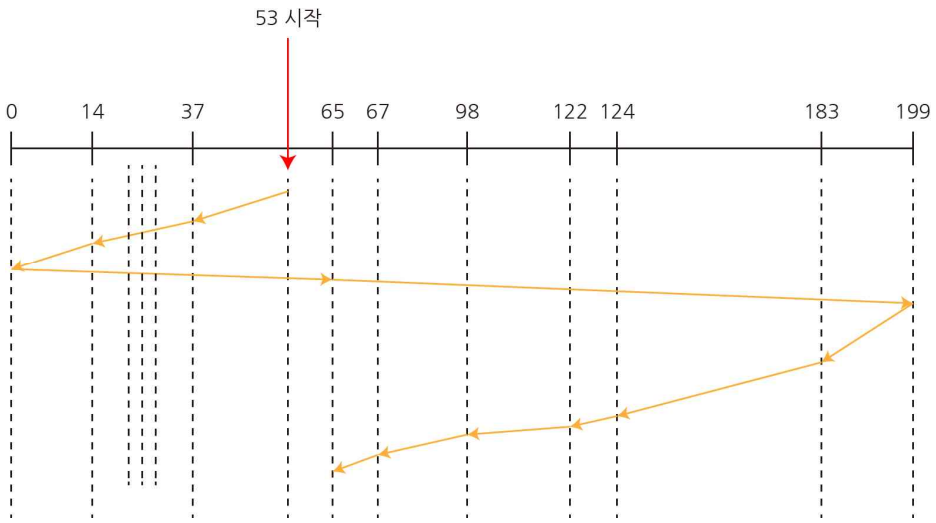
큐의 내용	98	183	37	122	14	124	65	67	합계
정렬	14	37	65	67	98	122	124	183	
이동 순서	4	3	1	2	5	6	7	8	
이동 거리	23	30	12	2	84	24	2	59	236

③ SCAN



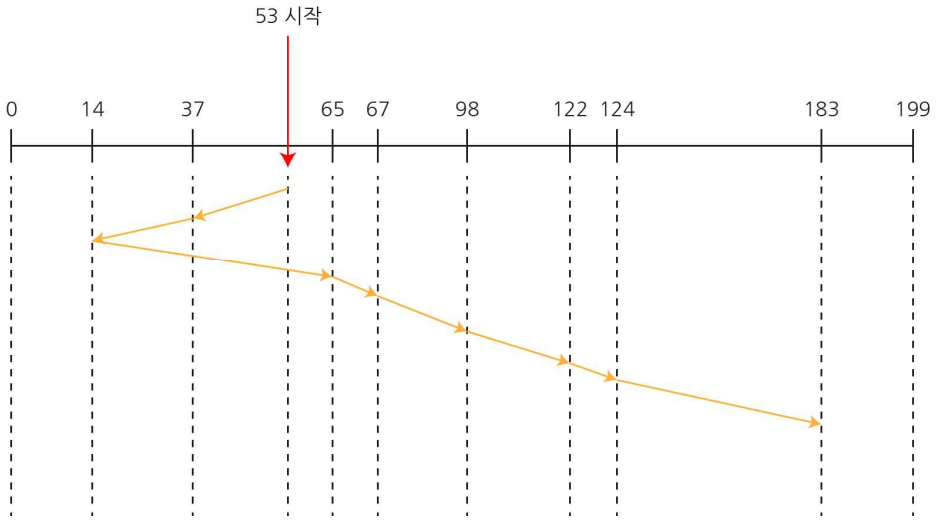
- 헤드의 진행방향에 있는 요청을 처리하고, 다시 반대 방향으로 틀어 반대방향에 있는 요청들을 처리한다.
- 엘리베이터가 동작하는 원리가 같아서 엘리베이터 기법이라고도 한다.
- 진행되는 과정에서 요청이 들어오면 해당 요청도 처리한다.
- 장점
  - SSTF의 바깥쪽 트랙의 기아가능성을 제거할 수 있고, 응답시간의 편차를 줄일 수 있다.
- 단점
  - 양 쪽 끝 트랙은 가운데 위치한 트랙보다 대기 시간이 길어진다.

④ C-SCAN



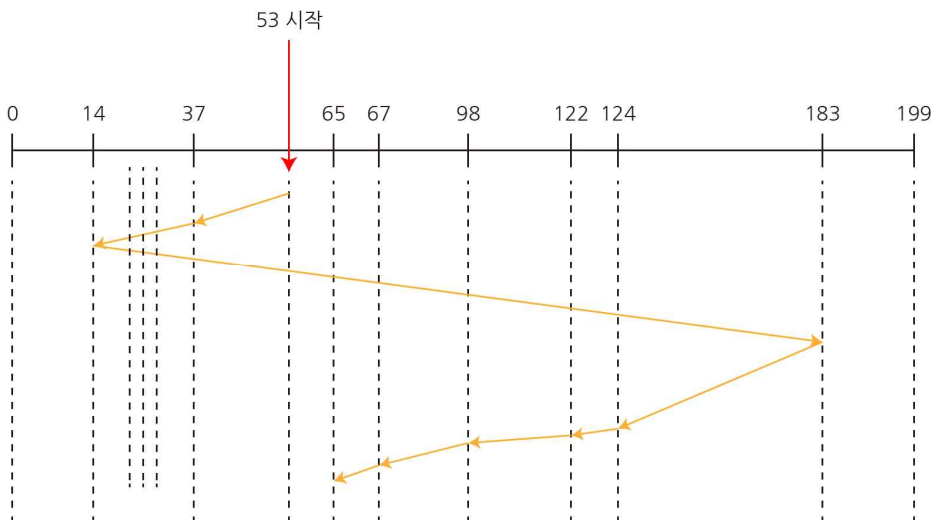
- 항상 한쪽 방향에서 반대방향으로 진행하며 트랙의 요청을 처리한다.
- 바깥쪽에서 안쪽으로 진행하며 요청을 처리한다.
- SCAN의 변형된 형태로 조금 더 시간을 균등하게 배분할 수 있다.
- 진행되는 과정에서 요청이 들어오면 해당 요청은 처리하지 않는다.
- 장점
  - 응답시간의 편차가 매우 적음, SCAN보다 시간균등성이 좋음
- 단점
  - 안쪽이나, 바깥쪽으로 처리할 요청이 없어도 헤드셋이 끝까지 이동하기 때문에 비효율적

⑤ LOOK



- SCAN 기법을 기초로 사용하며, 진행 방향의 마지막 요청을 처리한 후 반대 방향으로 처리하는 기법

⑥ C-LOOK



- C-SCAN 기법을 기초로 사용하며, 바깥쪽에서 안쪽 방향의 모든 요청을 처리한 후, 가장 바깥쪽으로 이동한 후 다시 안쪽 방향으로 서비스 하는 기법

⑦ N-STEP SCAN

- SCAN 기법을 기초로 두고 있으며, 시작하기 전 대기하고 있는 요청들을 우선적으로 처리하고, 처리하는 과정 중에 요청이 들어오는 것들은 이후에 모아서, 반대방향으로 진행할 때 서비스한다.

⑧ 에션바흐(Eschenbach)기법

- 부하가 매우 큰 항공 예약 시스템을 위해 개발
- 탐색 시간과 회전 지연 시간을 최적화하기 위한 최초의 기법

Section 7. 환경변수와 로그 파일

1. 환경변수

(1) 환경변수의 개념

- 프로세스가 컴퓨터에서 동작하는 방식에 영향을 미치는 동적인 값들의 모임
- OS에서 프로세스들을 생성할 때 참조하는 '변수'

(2) UNIX/Linux 환경변수

- env, set, printenv 명령어들을 사용하여 환경 변수와 그에 따른 모든 값을 볼 수 있다.
- export 명령을 이용하여 사용자 환경 변수를 전역 변수로 설정할 수 있다.
- 환경변수 종류

환경변수	설명
$PATH	디렉터리의 경로
$HOME	사용자의 홈 디렉터리
$LANG	기본 지원되는 언어
$USER	사용자의 이름
$TERM	로그인 터미널 타입
$PS1	1차 명령 프롬프트 변수
$HISTFILE	히스토리 파일의 절대 경로
$MAIL	도착한 메일이 저장되는 경로
$TMOUT	로그인 후 일정시간 작업을 하지 않을 경우 로그아웃 시키는 시간
$UID	사용자의 UID
$PWD	사용자의 현재 작업 디렉토리
$DISPLAY	X 윈도우에서 프로그램 실행시 출력되는 창
$OSTYPE	운영체제 타입

(3) Windows 환경변수

- 제어판 > 시스템 및 보안 > 시스템 > 고급 시스템 설정 > 환경변수 존재
- 커맨드 창에서 set 명령으로 확인
- 환경변수 종류

환경변수	설명
%HOMEDRIVE%	로그인한 계정 정보가 들어있는 드라이브
%HOMEPATH%	로그인한 계정의 폴더
%SYSTEMDRIVE%	윈도우가 부팅된 드라이브
%SYSTEMROOT%	부팅된 운영체제가 들어있는 폴더
%PROGRAMFILES%	기본 프로그램 설치 폴더
%TEMP%	임시 파일이 저장되는 폴더
%COMSPEC%	기본 명령 프롬프트 프로그램
%USERDOMAIN%	로그인한 시스템의 도메인 명
%USERNAME%	로그인한 계정 이름
%USERPROFILE%	로그인한 유저의 프로필이 들어있는 폴더명
%ALLUSERPROFILE%	모든 사용자 프로필이 저장된 폴더
%APPDATA%	설치된 프로그램의 필요 데이터가 저장된 폴더
%PATH%	실행 참조용 폴더 지정 목록

2. 로그 파일

(1) 로그의 개념

- 시스템의 모든 기록을 담고 있는 데이터

(2) 로그 데이터 정보

- 외부로부터의 침입 감지 및 추적
- 시스템 성능관리
- 마케팅 전략으로 활용
- 시스템의 장애 원인 분석
- 시스템 취약점 분석

(3) 로그 데이터 중요성

- 시스템에서 발생하는 모든 문제에 대한 유일한 단서
- 시스템에서 발생한 오류 및 보안 결함 검색이 가능
- 잠재적인 시스템 문제를 예측하는데 사용
- 장애발생 시 복구에 필요한 정보로 활용
- 침해사고시 근거 자료로 활용

(4) 리눅스 로그 종류

종류	설명
messages	시스템 로그 파일
secure	보안인증에 관한 메세지 로그파일
maillog	메일 로그 파일
xferlog	ftp 로그파일
dmesg	부팅 시의 시스템 로그
wtmp	시스템에 로그인 기록이 저장되는 파일(전체 로그인 기록)
utmp	시스템에 로그인 기록이 저장되는 파일(현재 로그인 사용자에 대한 기록)
lastlog	각 계정들의 가장 최근 로그인 기록