

운영 체제란

컴퓨터 하드웨어 바로 위에 설치 컴퓨터 시스템 자원을 효율적으로 관리
사용자 및 다른 모든 소프트웨어와 하드웨어를 연결하는 소프트웨어 계층

Cpu, Memory, i/o 장치 등의 효율적으로 관리

Disk Scheduling → 순서대로 처리하면 비효율적 이동하는 디스크 헤드가 있다.

기기별 속도 차이

Cpu > Memory > ssd

Cpu와 메모리의 속도 차이

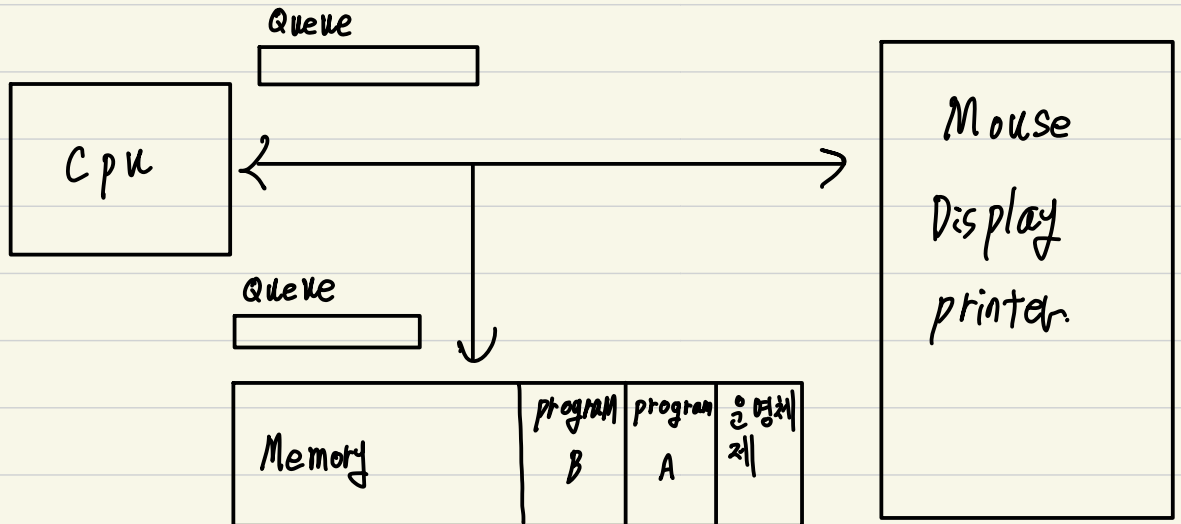
캐싱을 사용하여 중간 속도 차이 완충

Cpu i/o의 속도 차이
캐싱

Cpu의 작동 방식 실행 중인 프로그램들에게

짧은 시간씩 Cpu를 번갈아가며 할당 (동시성)

Computer system 내부 구조



운영체제가 하는 일

CPU 스케줄링 : 한정된 시간동안만 CPU 사용 CPU 사용권을 빼앗음.

메모리 관리

Disk 스케줄링 : disk head 이동 최소화, 입출력 최소화

인터럽트 : CPU가 기계어를 읽은 다음에 하는 것이 인터럽트 체크이다.
(가르쳐다)

CPU에게 인터럽트를 건다. 완료표알림 → CPU 현재작업중지 CPU 일을 처리 및 CPU 재점령

Cpu Scheduling (FCFS) (First-Come First-served) 첫번째 프로세스가 먼저 실행

process	금번 cpu 사용시간	→ Io	→ 다시 cpu Queue
P_1	24		
P_2	3		
P_3	3		

waiting time: $P_1=0$, $P_2=24$ $P_3=27$

Average waiting time $(0 + 24 + 27)/3 = 17$

Cpu Scheduling (SJF) shortest-Job-First

가장 짧은 프로세스를 제일 먼저 스케줄

minimum average waiting time 보장

starvation (기아 현상) 발생 가능

process	금번 cpu 사용시간
P_1	24
P_2	4
P_3	2

프로세스의 도착 순서: P_1 P_2 P_3

Round Robin [RR]

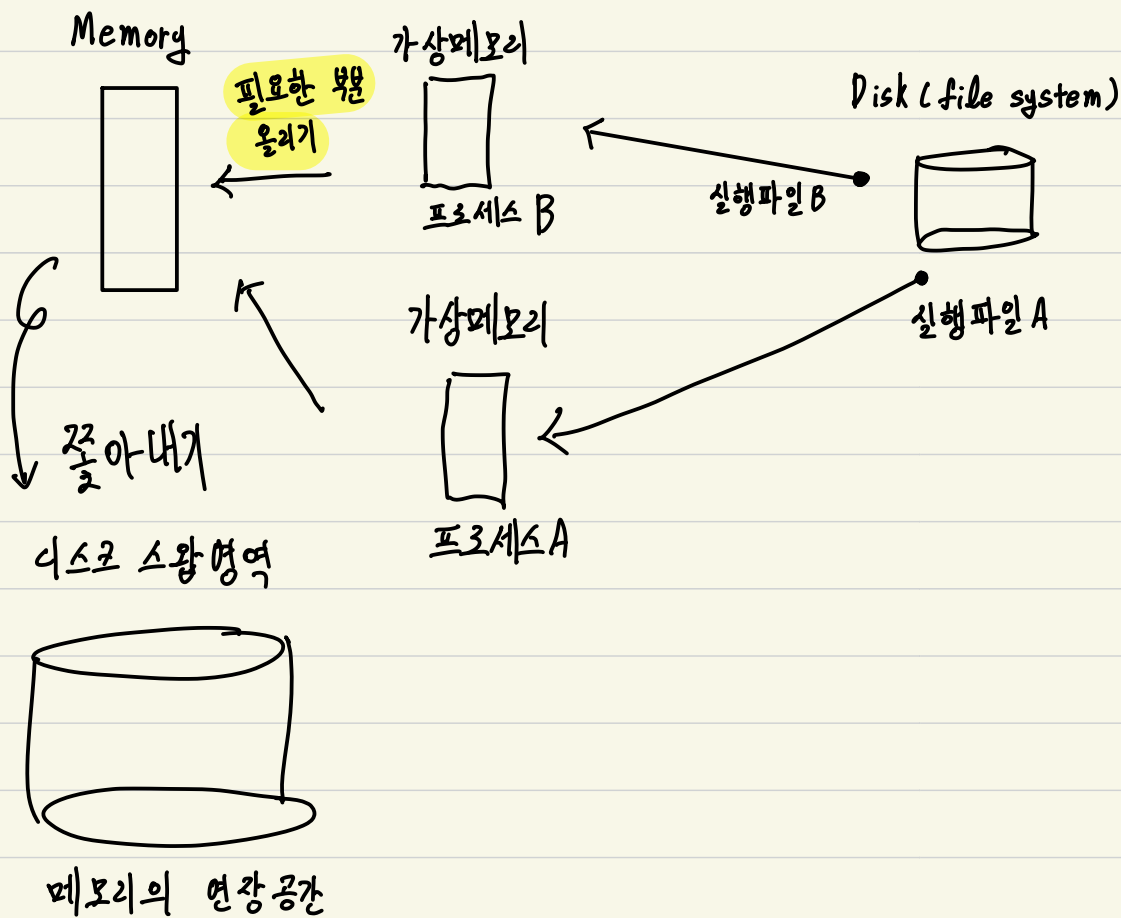
각 프로세스는 동일 크기의 cpu 할당시간을 가진다.

할당시간이 끝나면 인터럽트 발생 프로세스 cpu 빼앗기고

cpu 큐의 제일 뒤에 줄을 섰.

대기시간이 cpu 사용시간에 비례한다.

메모리 관리



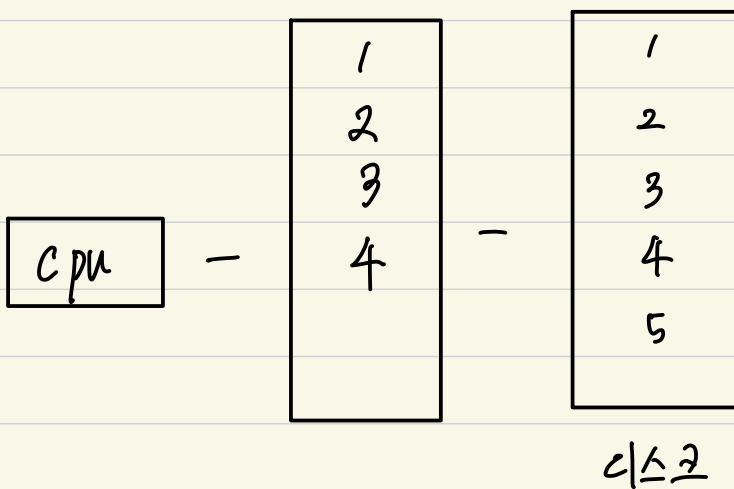
메모리 관리

CPU가 요청한 페이지 순서

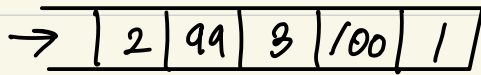
1, 1, 1, 1, 2, 2, 3, 3, 2, 4, 5, ...

LRU: (가장 오래전에 참조된 페이지 삭제) /

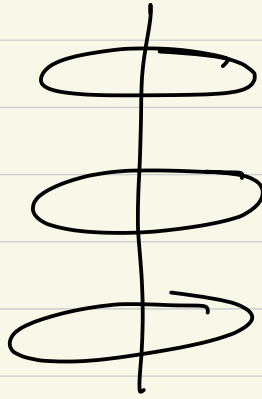
LFU: (참조 횟수가 가장 적은 페이지 삭제) 4



디스크 스케줄링



디스크 Queue



디스크 접근시간의 구성

- 탐색시간 (Seek time) *

• 헤드를 해당트랙으로 움직이는데 걸리는 시간

- 회전 지연

• 헤드가 원하는 섹터에 도달하기 까지 걸리는 시간

- 전송시간

• 실제 데이터의 전송시간

디스크 스케줄링 (Disk Scheduling)

- Seek time을 최소화하는 것이 목표

- $\text{Seek time} \approx \text{Seek distance}$

SSTF (Shortest Seek Time First)

starvation 문제

SCAN

디스크 헤드가 디스크 한쪽 끝에서

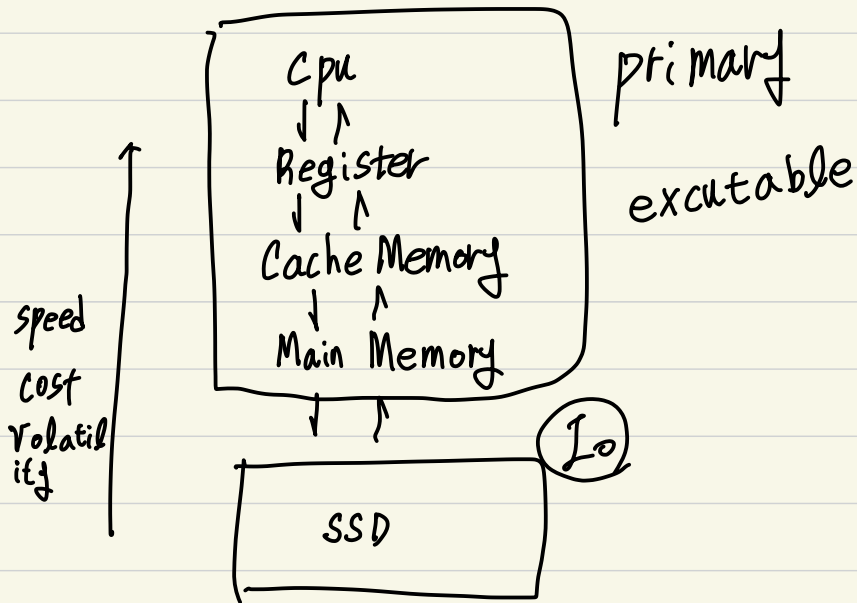
다른쪽 끝으로 이동하며 가는 길목에 있는

모든 요청을 처리

헤드가 한쪽 끝에 도달하면 역방향으로

이동하며 오는 길목에 있는 모든 요청 처리

저장장치 계층구조와 캐싱 (caching) : 속도 차이 완충



Caching: Copying information into faster storage system

Flash Memory

NAND형(스토리지), NOR형(임베디드 코드 저장용)

- 반도체 장비

Low power consumption

shock resistance

small size

Lightweight

Nonvolatile - 비휘발성

쓰기 횟수 제약

bad sector 발생
전하 문제

운영체제

Linux, Android

Window

Mac OS

Ios android — Linux

운영체제

협의의 운영체제 (커널)

- 운영체제의 핵심 부분 메모리에 상주하는 부분

Memory 관리가 중요하다 !!!

광의의 운영체제

- 커널 뿐 아니라 각종 주변 시스템 유틸리티
- 효율적으로 자원관리

운영체제의 분류

단일 작업 (Single-tasking)

한번에 하나의 작업만 처리

Ms Dos 프록프트 상에서는 한명령만 실행됨.

다중 작업 (Multi-tasking)

한번에 여러개의 작업을 처리

Windows, Linux, UNIX

일괄 처리 (batch processing)

- 작업 요청의 일정량을 모아서 한꺼번에 처리
- 작업이 완전히 종료될때 까지 기다려 한다.

시분할 (time sharing)

- 여러 작업을 할 때 컴퓨터 처리 증속을 일정한 시간 단위를 분할
- 일괄 처리 시스템으로 짧은 응답시간을 가짐
- Interactive한 방식

실시간 (Real time Os) - dead line을 넘기면 안된다.

정해진 시간안에

메모리관점

Multiprogramming 메모리안에 여러 프로그램

Time sharing은 CPU의 시간을 분할하여 나눠 사용

하나의 컴퓨터에 CPU (processor)가 여러개 붙어있다.

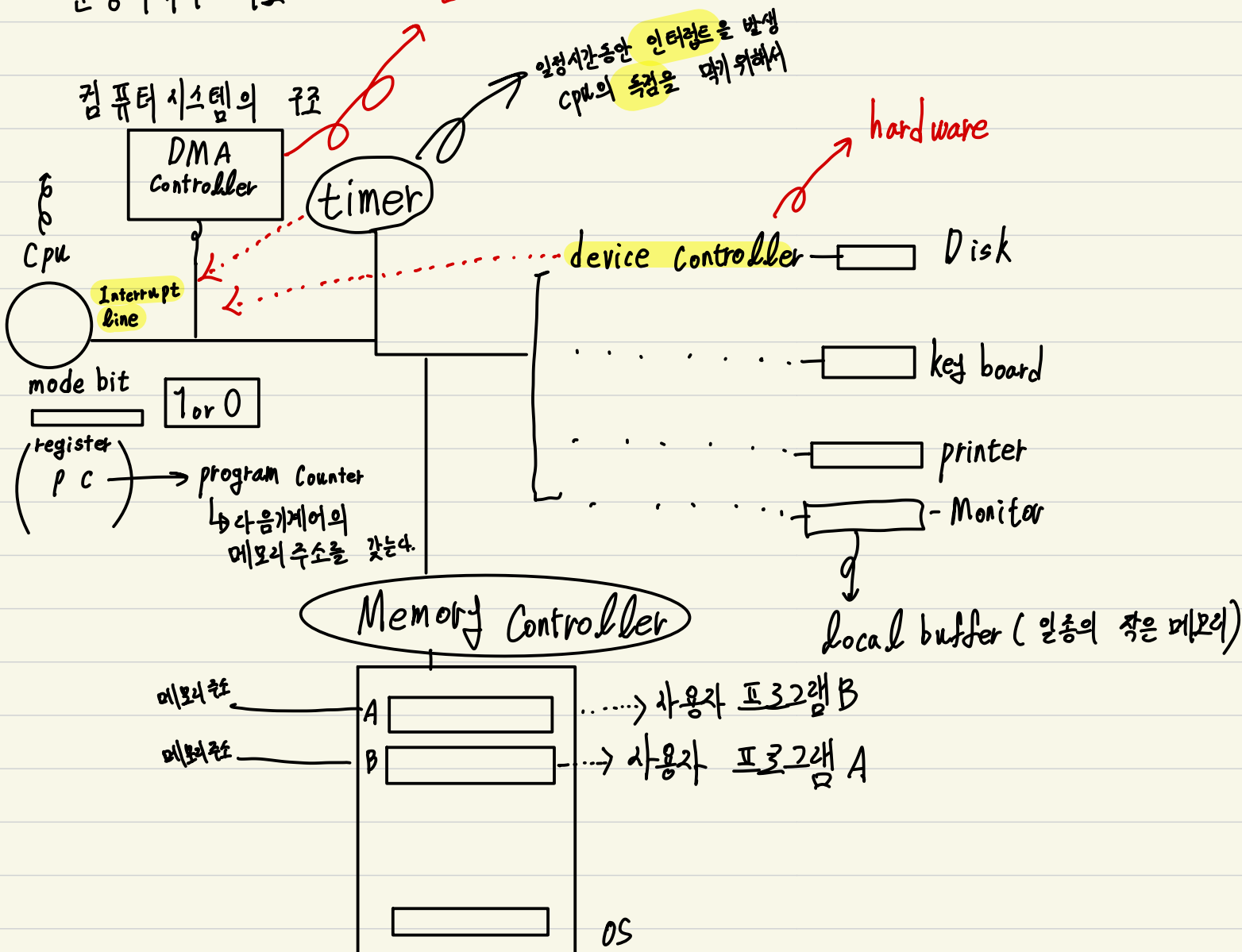
유닉스 (UNIX)

코드 대부분을 C 언어로 작성

높은 이식성 (open source)

최소한의 커널 구조 (효율적)

운영체제의 구조



System call (시스템 콜)

사용자 프로그램이 운영체제의 서비스를 받기 위해 커널 함수를 호출하는 것

Mode bit - 사용자 프로그램이 잘못된 수행으로 다른 프로그램 및 운영체제에 피해 X 하는 보호장치 필요

Mode bit (① 사용자모드 : 사용자 프로그램 수행 (안전한 기계어만 사용가능) If 권한 명령어 실행시
② 모니터모드 : OS 코드 수행 (특권 명령어) -> 운영체제만 사용 가능 ↓ 권한이 넓어짐.

장치 Controller 에서 인터럽트가 발생되면 CPU의 Mode bit 0으로 되며 Interrupt에 대응 되는 일을 한다.

(13분 52초) → 컴퓨터 시스템의 구조

인터럽트

인터럽트 당한 시점의 레지스터와 program counter save

CPU의 제어를 인터럽트 처리 루틴에 넘긴다.

Interrupt (넓은 의미)

Interrupt (하드웨어 인터럽트) : 하드웨어가 발생시킨 인터럽트

Trap (소프트웨어 인터럽트)

- Exception: 프로그램의 오류를 범한 경우
- System call: 프로그램이 커널 함수를 호출한 경우

인터럽트 용어

인터럽트 Vector : 해당 인터럽트 처리 루틴 주소를 가지고 있음

인터럽트 처리 루틴 (= Interrupt Service Routine, Interrupt 핸들러)

- 해당 인터럽트를 처리하는 커널 함수

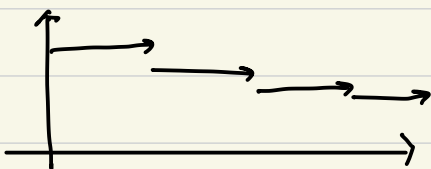
device controller

I/O 장치 유형 관리하는 작은 CPU

제어 정보를 위해 control register, status register를 가짐

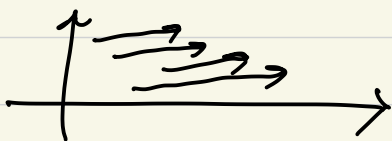
동기식 VS 비동기식 입출력

동기식 (Synchronous)



I/O 요청 후 입출력 작업이 완료된 후 제어가 사용자 프로그램으로 넘어감.

구현 방법 1) · I/O가 끝날 때 까지 CPU를 낭비

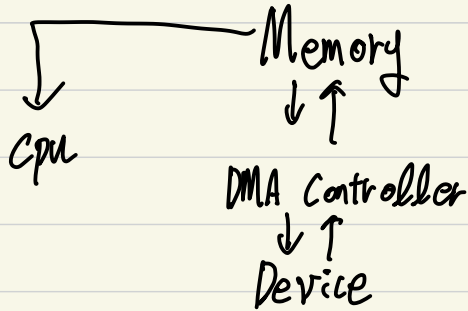


I/O 요청 후 출력하기 전에 다른 작업에 CPU 작업을 넘김

DMA (Direct Memory Access)

빠른 입출력 장치를 메모리에 가까운 속도로 처리하기 위해 사용

CPU의 중재 없이 device controller가 device의 buffer storage의 내용을 메모리에 block 단위로 전송

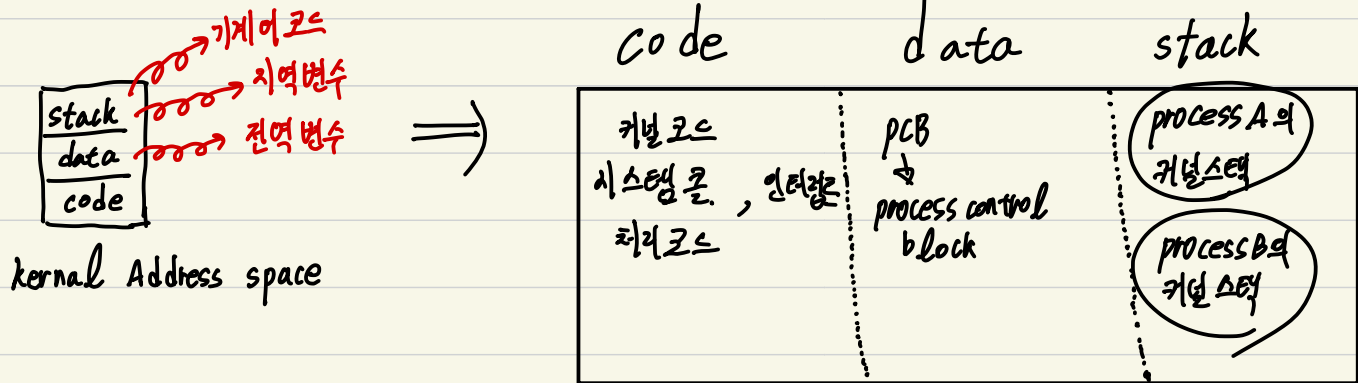


캐싱: 많이 사용하는 데이터 일시 저장 빠른 리승광역

프로그램의 실행 (메모리 load)

physical memory

virtual memory



사용자 프로그램이 사용하는 함수

사용자 정의 함수 (자신의 프로그램에 정의한 함수)

라이브러리 함수 (가장다 쓰는 함수)

커널함수 (운영체제 프로그램의 함수)

kernel 함수의 호출 = 시스템콜

프로세스를 스케줄링하기 위한 Queue

- Job Queue: 현재시스템내에 있는 모든 프로세스의 집합
 - Ready Queue: 현재메모리 내에 있으면서 CPU를 잡아서 실행되기를 기다리는 프로세스
 - Device Queue: I/O의 처리를 기다리는 프로세스 집합
- * 프로세스들은 **각큐**를 오가며 수행

process 실행 중인 프로그램
 ↳ 문맥 context
 과거의 CPU 사용량 메모리의 양

stack
 data
 code

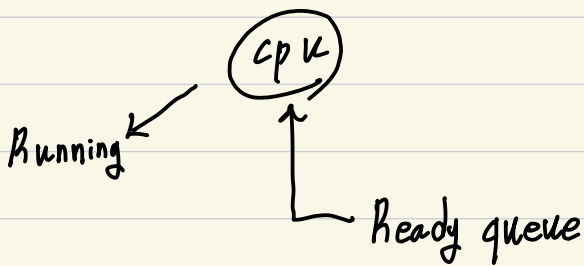
· program Counter
 · 각종 register } 하드웨어 문맥

프로세스

프로세스 관련 커널 주소

- PCB (process control block)
- kernel stack

프로세스의 상태



process의 state

Running : CPU를 잡고 instruction을 수행
 Ready : CPU를 기다리는 상태
 Blocked (wait, sleep) → 오래 걸리는 프로그램
 CPU를 할당 주어도 running은 실행 X.

Running

Ready

Blocked : 적극적으로 일을 하는 상태

suspended (stopped)

↳ 외부에서 프로세스 수행이 정지 X

resume → 다시 계속

process control block

PCB : 운영체제가 각 프로세스를 관리하기 위해 프로세스당 유지하는 정보
(구조체로 유지) 링크드리스트 자료구조의 포인터

(1) OS가 관리상 사용하는 정보

process state, process ID, scheduling information
priority

→ 부속이 코드 (캐시 메모리)

문맥교환 (context switch) 사용자 프로세스끼리 변화

CPU를 한 프로세스에서 다른 프로세스로 넘겨주는
과정

CPU가 다른 프로세스에게 넘어가면 그 프로세스 정보를
PCB에 저장한다.

운영체제의 커널에 들어가는 것은 문맥교환X

스케줄러 (scheduler)

프로세스가 바로 ready queue로 가게 X
long term scheduler가 admitted 시켜준다.

Long term scheduler (장기 스케줄러 or job scheduler): 일종의 Memory scheduler
degree of Multi programming을 제어

시작 프로세스들 이면것들을 ready queue로 보낼지 결정

timesharing system에는 보통 장기 스케줄이 없다. (무조건 ready 상태) → 일반적인 window

short-term scheduler (cpu 스케줄러)

cpu에 프로세스에 번갈아 주는 코드

충분히 빨라야 된다. (milliseconds 단위)

Medium-term scheduler (중기 스케줄러 or Swapper)

메모리가 부족할때 메모리에서 디스크로 통째로 쫓아낸다.

프로세스에게서 memory를 뺏는 문제

Running 상태

프로세스가 자기 코드를 실행한다.

usermode 간주

시스템콜 kernel mode.

운영 체제 Running CX)

(시스템콜) 소프트웨어
→ 소프트웨어

hardware interrupt.

스케줄러: 운영체제가 메모리를 관리

메모리에 올라간 프로세스의 수 (degree of multiprogramming)

Suspended 상태는 메모리에서 프로세스가 내려온 상태 단, I/O 작업은 가능하다.

suspended 상태에서 \rightarrow suspended ready \rightarrow ready

kernel의 도움이 필요하거나 kernel의 프로세스가 실행이되도 기존 프로세스는 running

시스템콜 (운영체제 \rightarrow 디스크) \rightarrow 소프트웨어 인터럽트

i/o 인터럽트 (디스크에서 운영체제로) \rightarrow 하드웨어 인터럽트

프로세스: stack, data, code

스레드: CPU 수행단위를 가지고 있다

프로세스 a \rightarrow 프로세스 b로 가는 것 Contextswitch 오버헤드가 많은 작업

thread 1에서 thread 2번 으로 가는 것은 오버헤드가 큰 작업이 x

스레드의 구성

프로그램 카운터

레지스터 셋

스택 공간 (함수호출)

스레드가 동요 스레드와 공유하는 부분 = task

코드 섹션

데이터 섹션

OS 자원

별도로 갖는 부분

registers

stack

heavy weight process - 하나의 thread를 가지고 있는 task

스레드 이점

· 응답성이 빠르다.

· 자원을 공유한다.

· 병렬성을 추구한다.

프로세스 1개보다 스레드 1개 만드는데

30배 효율적

프로세스 생성 → 부모 프로세스 → 자식 프로세스

복제 생성 (system call)

`fork()` 시스템콜 \rightarrow creates process

프로세스는 created 된다. by fork()에 의해서

V creates new address space caller과 똑같은 것을 복제

프로세스는 트리 (계층 구조) 형성

프로세스는 자원 필요

은영체제로 부터 받는다.

부모와 리소스를 공유한다. (원칙적으로 공유X)

수행 → 부모와 자식이 공존

자식이 종료될 때까지 부모가 기다리는 wait 모델

→ blocked.

Address Space - 자식은 부모공간을 복사 binary And OS data.

유닉스의 예

fork() 시스템 콜이 새로운 process를 생성!

fork 다음에 이루어지는 `exec()` system call을 통해

새로운 프로그램을 메모리에 올림.

Multiprocessor scheduling

load sharing 일부 프로세서에 job이 몰리지 X

Symmetric Multiprocessor (SMP)

각 프로세스가 알아서

Asymmetric → 하나의 processor system 공유 몇몇은 책임 나머지는 기계에 맡김

리얼 Time scheduling 빨리 처리 X deadline 지켜야 됨

soft - real-time 일반 프로세스보다 우선순위

최상 24 디코딩 동영상 스트리밍

soft real time task

Thread cpu 수행 단위

Local scheduling: 프로세스 내부에 thread

global scheduling
운영체제의 스케줄러

↳ 운영체제가 스레드의 존재를 알음
이런 스케줄러가 어떤 스레드를 스케줄

코딩 모델

이론

서버 요청 처리 → 도착

큐

빈도

확률 분포



Measuremen €

$$\boxed{0 - 6}$$

$$0 + 6 + 3 + 9 / 4 =$$

$$\odot \quad n-1 \\ 6$$

10

$$p1 \rightarrow p4 \rightarrow p3 \rightarrow p5 \rightarrow p2$$

$$\odot \quad \begin{array}{ccc} 10-3 & 14-2 & -4 \\ 7 & 12 & \end{array}$$

$$\text{반환 시간} = 12$$

8

$$14 - 4$$

$$\text{프로세스 순서} \odot \quad \text{순서} (12)$$

