

## CP, 4 스레드 & 멀티태스킹

### 1. 프로세스 문제점

멀티태스킹: 컴퓨터 시스템내 여러 태스킹 동시에 실행.

- ↓ 1 :
- ① 프로세스 생성 큰 오버헤드
  - ② 프로세스 컨텍스트 ~~switching~~ 스위칭 //
  - ③ 프로세스 사이 통신 어려움

과거 운영체제의 실행단위는 프로세스,

오늘날 // 스레드를 대상으로 스케줄링!

→ 프로세스는 스레드에게 공유자원을 제공하는 컨테이너 역할

### 2. 스레드 개념

스레드 ① 프로세스를 사용하는 문제점을 해결 위해 고안. 더 작은 단위.

- 실행단위이며, 스케줄링 단위.

- 운영체제에게 실행단위 / CPU에게 스레드 단위

- 코드 / 데이터 / 힙 / 스택을 가진 실행체

- 스레드마다 정보를 저장하는 구조체 **TCB**

프로세스 스레드 컨테이너, 반드시 1개 이상 스레드로 구성

스레드 ② • **프로세스**는 스레드들이 공유공간 제공

- 스레드는 함수로 작성 (스레드코드는 함수로 만들림)

- 스레드로 만든 함수 종료 → 스레드 종료

- 스레드 종료 → TCB 등 관련정보 제거


- 프로세스에 속한 **스레드** 종료 → 프로세스 종료

- 스레드마다 하나씩 TCB


# 멀티스레딩

실행중인 스레드

① 동시성 : 1개의 CPU에서 2개 이상의 스레드가 동시에

concurrency 

② 병렬성 : 2개 이상 스레드가 다른 CPU에서 같은 시간에 동시 실행

parallelism 

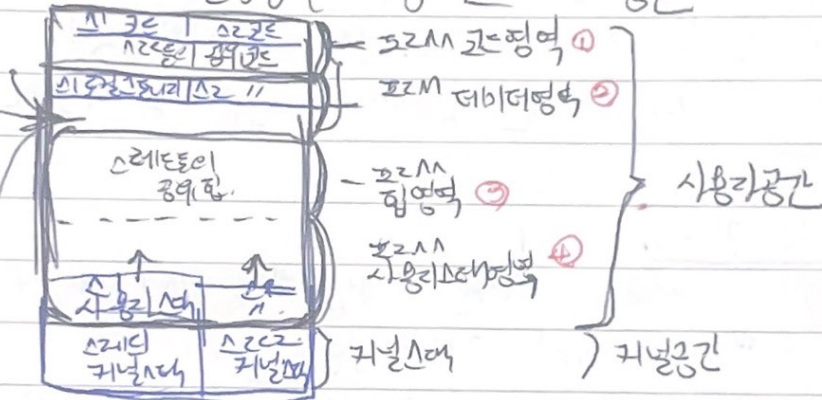
## 3. 스레드 주소 공간과 레지스터

스레드 스레드가 실행중에 사용하는 메모리 공간

주소 공간

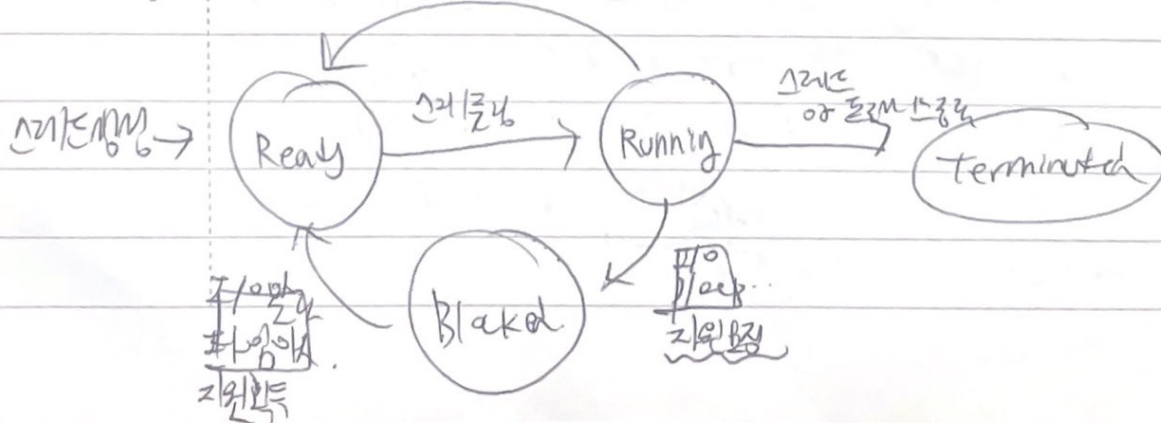
P53

프로세스 영역 (모든 스레드 공유)



- ① 스레드가 실행할 작업이 함수, 프로세스 코드 영역의 디펜던스 때문
- ② // 사용하는 모든 메모리 공간
- ③ 모든 스레드가 공유한다.
- ④ 스레드가 생성될 때마다 프로세스의 사용자 영역의 일련된 메모리

프로세스의 주소 공간에  
스레드 주소 공간을 할당한다.





이벤트에

↑  
스레드들

응용프로그램이 스레드에 대해 하고 있는 응용프로그램,

- ① 스레드 생성 ② 스레드 종료 ③ 스레드 대기 ④ 스레드 실행  
구현.

스레드  
컨텍스트

스레드의 실행 중인 상태 정보  
PC, SP, 상태 등

TCB

스레드로 실행하는 데 따라 스레드에 대한 정보를 담은 관계  
개념 명칭이 만들어진 개념이다. 스레드.  
스레드 실행 → TCB 자동 생성.

ready  
준비 상태  
블록 //  
block.

준비 상태에 있는 스레드들이 TCB 연결하여 관리는 리스트  
블록 // // //

스레드  
컨텍스트 스위칭

→ 현재 실행 중인 스레드 중단, 다른 스레드에 CPU 할당

이유 4가지

(스레드 실행)

- ① 스레드가 자발적으로 양보.
- ② // 시스템 호출을 실행하여 블록.
- ③ // 의 타임슬라이스(시간 할당량) 소진.
- ④ I/O 부러 인터럽트 발생

스레드 스케줄링

- 1) 현재 실행 중인 스레드를 TCB-A에 저장  
TCB-B는 CPU 장치. (CPU 장치 저장하기)
- 2) 스레드 상태 정보 처리. 커널 정보 수정

TCB-A 저장  
오버헤드

- ① 모든 CPU 작업은, CPU 시간 필요.  
민감도 관리가 없으면, 처리를 저하.
- ② 동일 우선순위에 다른 스레드로 스케줄링  
다른 // // //

# 4. 케널레벨 12C와 사용자레벨 12C

12C 스케줄링 주체에 따라 2가지

① 케널레벨 12C: 케널내에 스케줄링되는 스레드

② 사용자레벨 12C: 스레드 외부에 스케줄링되는 스레드

① 응용프로그램 실행 중 ① 생성

멀티프로세스  
병렬성

TTC를 케널공간에 생성 및 소유함. 케널내에 스케줄  
스레드 주소공간(12C 주소 레지스터) : 사용자공간

② 11 라이브러리 호출 ③ 생성

멀티프로세스  
병렬성

X ③

멀티프로세스  
직접

12C 라이브러리가 스레드 생성(U-TCB)을 사용자공간에 생성 & 소유  
- 케널 사용자레벨 스레드 존재 안함  
- 11 케널에 스케줄  
스레드 주소공간(11) : 사용자공간

스레드  
12C

부정확한 케널의 가용성으로 스레드  
스레드의 주소공간 : 케널공간

## 중요 개념

← 리눅스

① 사용자레벨 12C에 커널스레드가 존재함

- 케널 11 : 케널내 리소스 소유

- 사용자 11 : 케널 소유

② 이식 및 활용성 - 사용자레벨 스레드에 호환성 있음

③ 실행 방법 - 사용자 11의 경우, blocked 상태일 때,

같은 응용프로그램에 다른 모든 사용자레벨 스레드가

④ 병렬성 - 사용자레벨 12C는 멀티프로세스 실행됨



사용자: 커널

## 5. 멀티스레드 구현

멀티스레드 응용에서 작성한 스레드가 시스템에서 ~~주~~ 실행/종료 구현

구현: ① N:1 ② 1:1 ③ N:M

P61항 ① N:1 : 운영체제 수준 프로그램으로 단일 스레드 프로그램 다음  
프로그램 당 1개의 커널레벨 스레드 생성 인터피

커널레벨 스레드의 스레딩 - 사용자 스레드는 스레드 라이브러리에 의해 스레딩

\* 비평! 사용자 스레드는 해당 커널 루틴이 스케줄되어야 실행 가능하도록 되어 있음

장점: 단일 호어 CPU에서 실행 속도 빠름.

단점: 멀티코어 CPU(현제) 비효율, - 병렬 처리 불가

프로그램이 스레드로 실행됨 - 하나의 사용자 스레드 블록, 프로그램 전체 블록

② 1:1 : 사용자 스레드 당 1개의 커널 스레드 (TCB) 생성  
매핑된 커널 스레드 스케줄링, 사용자 스레드 실행

장점: 개념 단순 → 구현 용이

높은 병렬성 / 블록되지 않게끔 X: 성능

단점: 커널에 접근 부담

사용자 스레드 많아지면, 모두 바뀔어 부담

③ N:M : N개의 사용자 스레드와 M개의 커널 스레드 - 비평

장점: 커널 인터디 접속하여 커널의 부담 ↓

단점: 구현 복잡. 라이브러리 의존.

대형 애플리케이션  
fork()를 통한 스레드

## 6. 멀티스레딩 이슈

장점: 높은 실행 성능 (병렬 실행), 응답성 향상, 자원 사용 효율성, 응용 프로그램의 신뢰성, 동시

주의점: 여러개의 스레드 중 한 스레드가 fork() 호출

한 스레드가 exec() 호출 - 현재 프로세스의 모든 프로세스 종료, 새로운 응용 프로그램의 실행

스레드 사이 동기화 문제 → 동기화 방법 이해

## 7. 프로세스 간 통신(IPC)

프로세스 자원들이 분리되어있고, 여러프로세스가 동시에 여러작업을 하고, 데이터로 주고받아야함: 통신!

p. 65

각각의 프로세스에서 동일함의 변수사용  
→ A프로세스의 x와 B프로세스의 x가 같지않다!  
fork() 하는 순간, 자식프로세스를 만든다  
= 별개의 프로세스.

IPC  
Inter  
Process  
Communication

운영체제가 제공하는 프로세스 간 통신방법  
→ 라이브러리있음

① 신호 ② 공유데이터 ③ 파일

## 8. 신호(Signal)

신호. 가장 오래된 IPC 방법. Bell Lab의 유닉스에서.

특징 ① 자기자신으로부터 수신 ② 다른 프로세스로부터 수신  
③ 커널로부터 수신 (방법명)

신호전달과정

① 호출 시리 신호핸들러를 등록.  
② 호출 → 호출 신호발송  
③ 호출가 등록해둔 신호핸들러(비사용자정의함, 커널코드가 실행)  
Signal이 - 신호 핸들러로 등록한 시스템호출  
Kill이 - 신호를 보낸 시스템호출

신호의 종류

모든 신호는 커널보다 고위이름(SIG...)을 가짐. 하위커널  
커널은 신호발생시, 수신호출자에게 신호전달 & 신호핸들러실행



반역 신호 전달과 더불어 이동성, 디폴트 행동이

신호의 종류

이름

코어덱트

p 66

신호의

디폴트 행동

1) 코드 : 신호를 수신할 프로세스 강제적으로

2) 코어덱트 :

시그널, 코어파일 (종료)

실행

3) 중지 : 이 실행을 일시 중지

4) 무시 : 신호를 무시하고 아무런 일도 하지 않음.

상대적  
에러상태를  
기록

이러한 문제들이라는 비동적 사건.

프로세스는 사용자 모드에서 실행을 계속함.

신호전파 :

신호발생 : 목적 프로세스를 찾고, PCB의 신호에 도착했음을 알리고 리턴

(1) 신호종류 : signal() 이용. PC에서 커널에 보내기

구조체

프로세스 신호를 처리 할 때, 신호마다 신호처리 함수의

(2) 발생 : kill() 시스템호출 사용. PID 번호로 프로세스에

signal() 신호를 보내는 것.

- kill()의 실행이 커널로 수신 프로세스 구조체 PCB에

신호 발생만 표시 후 리턴

전파 - 이이동성 신호 PCB에 이처가 되게 하기 - 그냥 리턴

(3) ~~신호~~ : 수신 프로세스에 도착해야만 신호를 처리할 수 있음 X.

- 구조체나 사용자 모드에서 신호를 처리함

- 실행이 커널에서 시작되면, 사용자 모드로 올라가면 신호를 처리함

신호 전달은 수신 프로세스가 스케줄될 때만 실행

// 사용자 모드에서 실행

커널직할 : 신호를 전달하고 실행되게 하는 것.

신호 수신 프로세스에 직접 :

→ 발생한 신호 커널의 버퍼에 저장 기록되어.

미처리 상태를 존재함,

프로세스가 신호 재처리 시 처리