3. 프로세스와 프로세스관리

프로그램: 하드디스크등 실체

☆프로세스: 프로그램이 메모리에 적재되어 실행중인 상태

- 운영체제는 프로그램 적재 → 프로세스로 다룸
- 〃 는 프로세스에 필요한 메모리할당 / 프로세스가 입출력(변수) 프로세스 아이디
- 프로세스 관련정보 - 커널영역을 통해관리 ☆
- 〃 실행 - 대기 - 잠자기 - 대기 - 실행 - 종료 (생명주기

⇒ 프로세스는 상호 독립적인 메모리공간에서 실행 (침범X)

Ex) 다중프로그래밍이니까 (여러프로그램을 메모리에 적재)

- 한 프로그램을 여러번 실행시키면?

(+) 프로세스는 서로 침범하지 않은 메모리공간에서 실행되며, 실행시마다 독립된 프로세스로 실행

→ 다중 ~~프로세스~~ 인스턴스

CPU 주소공간 : CPU가 주소선을 통해서 액세스할수있는 전체 메모리공간.

공간크기 - CPU 주소선의 수에 결정.

32비트 CPU → 32개의 주소선 → 2^{32} 개의 주소

→ 2^{32} 바이트 ⇒ 4GB !

1GB = 1024MB = 2^{10}

☆ CPU 주소공간 < 큰메모리 : 액세스불가능

☆ CPU 주소공간 > 작은메모리 〃 가능.

→ CPU가 설치된 메모리의 주소영역을 넘어가면 시스템오류.

<프로세스 구성>

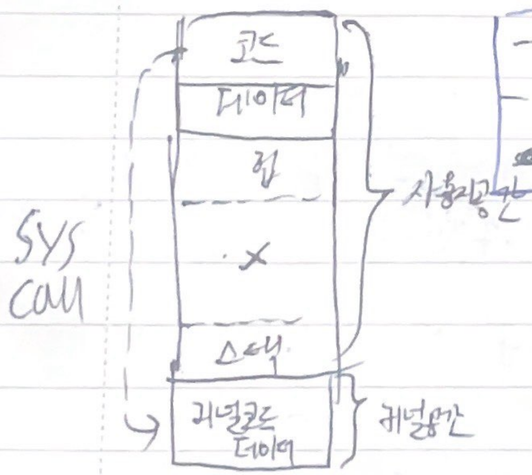
- 코드영역
- 데이터 ∞
- 힙 ∞
- 스택 ∞

① 코드: 실행될 프로그램 코드 적재
② 데이터: ~~…~~ 에 고정적으로 쓰는 변수공간 (전역/정적/라이브러리)
③ 힙: 프로세스 실행 중, 동적 사용 할당
④ 스택: 함수가 실행될 때, 사용될 데이터를 위해 할당
// 호출된 프로세스에서 필요시 사용가능

<프로세스 주소공간> (사용자+커널)

프로세스 실행 중 접근할 수 있도록 허용된 주소의 최대범위
☆ 논리공간(가상공간) 즉, 물리X 모두 사용
프로세스주소공간크기 = CPU주소공간크기

프로세스의 코드와 데이터영역은 실행파일에 결정된 상태로 적재
→ 실행중 크기 변동X
프로세스는 사용자공간의 최대범위까지 동적할당 받으며,



- 프로세스마다 각각 사용자공간 있음
- 시스템 전체에 하나의 커널공간
→ 모든 프로세스는 커널공간 공유

가상주소의 데이터는 물리메모리에 분산되어
어느 물리 번지에 있는지 모름.

즉, 연속적인 것처럼 보여도, 실제 물리메모리에서 적용X

프로세스의 주소공간은 각 프로세스마다 제공(별개)하고, 주소공간은 겹쳐 X
- 가상주소가 실제주소로 매핑. 물리메모리 겹침 X.

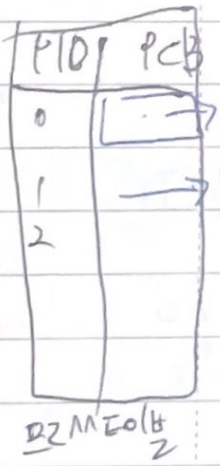
P.36 [5] 프로세스의 구성요소 ★ 힙은 동적이다

2 커널의 프로세스 관리

프로세스 테이블: 시스템의 프로세스를 관리하기 위한 표. 시스템당 하나. 운영체제마다 구현방식 다름.

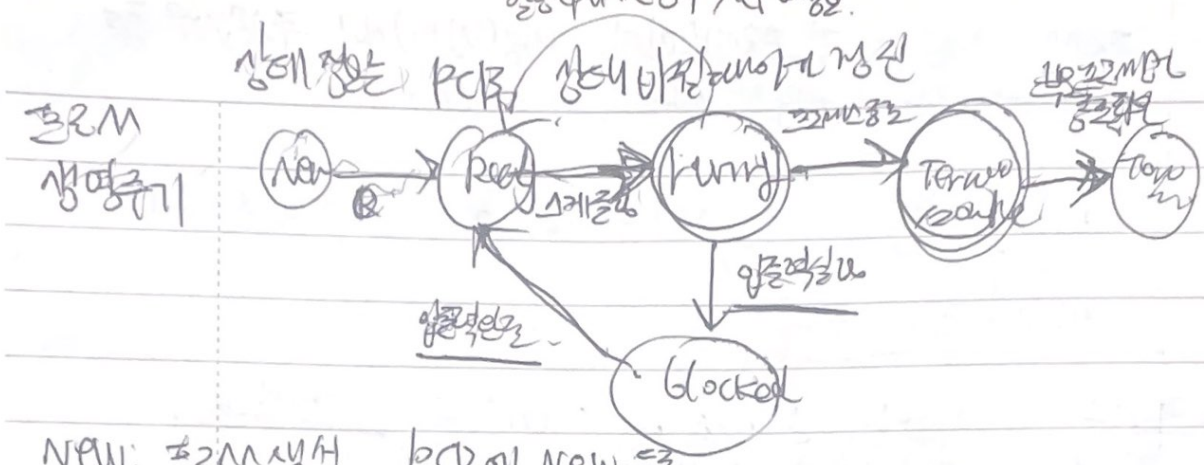
프로세스 제어블록 PCB: 프로세스에 관한 정보를 저장하는 구조체. 프로세스당 하나
프로세스 생성시 PCB 생성 → 프로세스 종료시 PCB 삭제.

프로세스테이블과 프로세스 제어블록 PCB는 커널공간, 커널모드만 액세스



★ 저장정보

(1) PID 프로세스 번호 : 0 이상의 정수. 유일. 이것로 프로세스구분
(2) PPID 부모번호 : 부프로세스의 PID
(3) 프로세스 상태정보 : 준비 - 실행중 - 블록(입출력완료대기) 등
(4) CPU 컨텍스트
PC : 프로세스가 선택되면 실행할 다음 프로세스 명령주소
- 사용자모드에 있는 경우 - 사용자공간의 명령주소
- 그 외 " - 커널공간의 명령주소
(5) 스케줄링 정보 우선순위, nice, 스케줄러
(6) 프로세스 종료코드
(7) ... 등.
(8) 메모리관리정보 : 페이지테이블의 물리메모리 주소.



New: 프로세스생성 PCB에 New등록.

Ready 준비: 프로세스가 스케줄링을 기다리는 준비상태.
" 는 준비큐에 대기.
스케줄링되면 Running, CPU 에디바이실행

Running 실행상태: 프로세스가 CPU에 의해 정상실행
- CPU의 사간할당량(타임슬라이스)가 지나면, Ready로 바뀌고, 준비큐에삽입.
- 프로세스가 입출력을 시행하면, 커널은 프로세스를 Blocked로 만든 대기큐

Blocked/wait: 프로세스 자원요청 or 입출력요청 인터럽트처리 입출력 완료시 프로세스는 Ready, 준비큐삽입

Terminated/Zombie: 프로세스가 본인의 종료된 상태 (좀비상태)
- 프로세스가 처리하고 완료 메모리나 할당자원 커널에의해반환
- 프로세스데이블항목과 PCB가 여전히 시스템에서 제거되지않은상태
- 프로세스가 부모프로세스(PCB에있음)을 부모프로세스가 알아차리지않아 완전히 종료되지 않은 상태.

Terminated/Out: 프로세스 종료시에 할당자원(PCB)을 부모프로세스가 회수 완전종료
시스템에서 완전 종료

3. 프로세스 계층 구조

프로세스는 일반적으로 부모-자식 관계임
#0 최초 프로세스, 부모 프로세스는 여러개 자식 프로세스 OK
~~자식~~ 모든 프로세스는 부모 허락에 생성가능

#0 프로세스 | Unix - swapper
리눅스 - idle - 우선순위가 가장 낮다.

시스템 호출
fork() - 자식 프로세스 생성
exit() - 현재 프로세스의 종료를 커널에 알리는 시스템 호출
wait() - 부모가 자식 프로세스 종료를 기다리고 확인하는 시스템 호출

P41 자식이 부모보다 먼저 종료, 그럼 자식은 좀비!!
(ex) 종료시 부모확인해야하는데, PCB가 남아있어

~~좀비 프로세스 제거~~
고아 프로세스: 부모가 먼저 종료한 자식 프로세스.
→ 커널은 자식 프로세스 확인하고, init 프로세스에 입양.
혹은 자식 // 도 종료

CPU 집중 프로세스: 대부분 시간을 CPU 작업만 함
I/O 집중 프로세스: // 입출력 //
우선순위는 I/O! (ex) I/O 동안 다른 프로세스에 CPU 할당가능하니까

```
pid = fork() //자식프로세스 생성
if (pid > 0) {//부모프로세스}
else if (pid == 0) {//자식 프로세스}
else {//에러}
```

P44 자세한 코드 3-5

* 언제 enzomble 로 요르지?

exec()

* 프로세스 오버레이	현재 실행 프로세스의 주소공간에, 새 응용프로그램을 적재실행
	PID 변경 X, fork()에 의해 생성된 자식프로세스는 바로 exec()로 실행
* exit()	시스템콜로 프로세스 종료마킹
	① 프로세스 모든 자원 반환.
	② PCB-프로세스상태를 terminated 변경. PCB 종료코드 저장
	③ 자식있으면, init 프로세스 입양.
	④ 부모 프로세스에게 SIGHLD 전송
* 종료코드	. 프로세스 종료 상태 & 이유를 부모에게 전달함.
	-255 값 없음. / -1 리턴시, 종료코드로 255 전달
	1111 1111 2의 보수.

* 커널 remind

(1) 커널은? 스스로 실행 X

↳ systemcall & ISR 에 의해 호출되어 실행

(2) 커널은 실행 중 X (프로세스도 스레드도 아님?)

↳ 응용프가 시스템콜해서 커널코드 실행

↳ ~~ISR~~ 인터럽트발생 → ISR이 실행.

(3) ~~커널~~ 스택&힙 커널거 아님.

↳ 프로세스 & 스레드가 가지는 주체임.

↳ 프로세스 생성시 사용자공간 & 커널공간에 각각 스택 조성

프로세스(&스레드) - 사용자코드실행 - 사용자공간스택활용

// - 시스템콜/커널코드실행 - 커널스택활용