# 프로세스. · 실행파일이 메모리에 적재되어 실행중인 상태.

"프로그램이 메모리에 올라가면" 프로세스
↓
명령의 나열.

· CPU가 읽고 쓸 수 있음.

· Executable 하다 … 하드에 매핑 됨.

{ · PCB (Process Control Block)의 존재로서 명시되는 것. }

· 프로세스가 할당되는 개체로서 $\underbrace{\text{디스패치}}$ 가능한 단위.
↳ "CPU가 일다 마다"

→ PCB가 있으면.. OS가 관리가능함.

# 다중 프로그래밍. "여러 프로그램을 동시에 실행하자"

· 여러 프로그램 메모리에 위치.

· Multi-instance, 같은 프로그램, 실행할때마다 독립된 프로세스 생성.

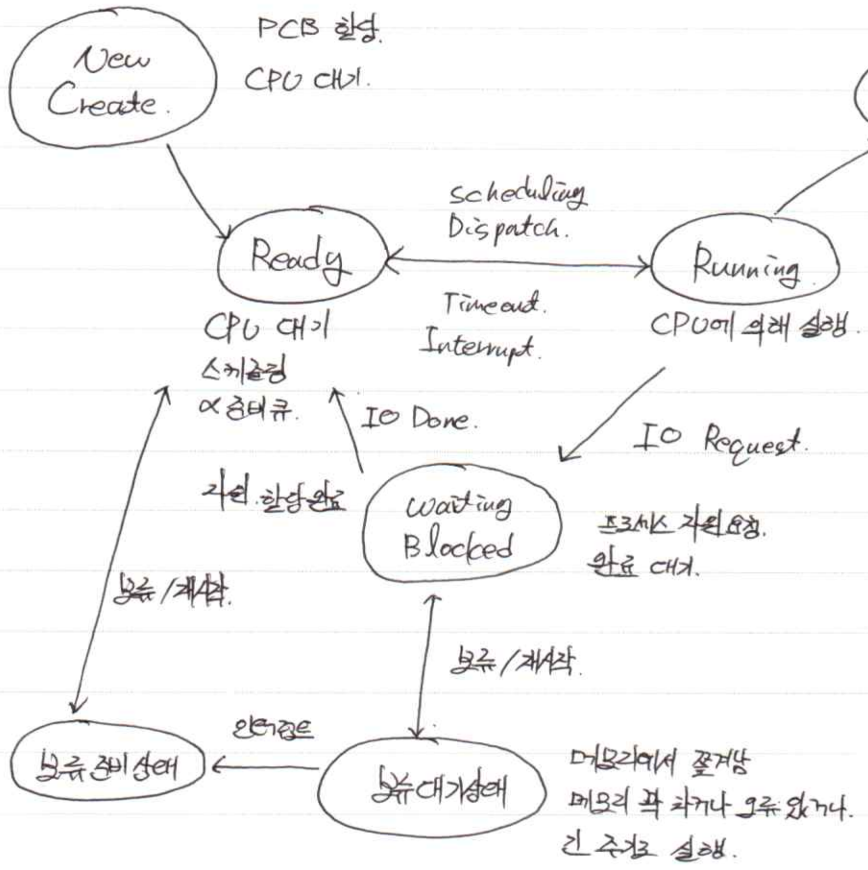
# Loading. 1. Memory Allocation.

2. Linking. (Refer external libraries, Link syscall.)

3. Relocation. : Memory Address … Relative (x+50).
↓
Absolute (x+50).

4. Load to Memory. : 좁은 의미의 로딩.
Loading

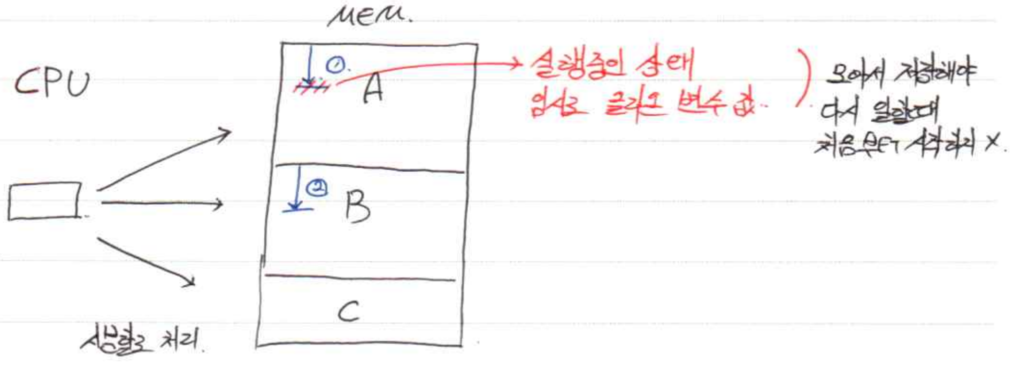
Life Cycle.



프로세스 관리.

- OS에 의해 관리.
- PCB로 관리함.

* Context Switching. "문맥 교환"



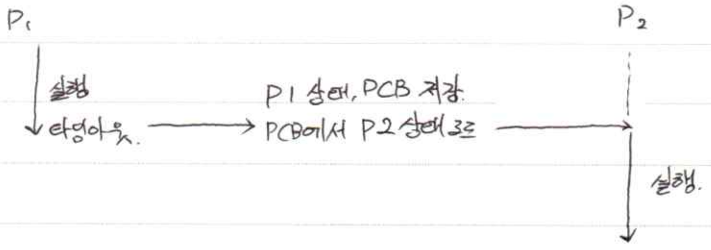
PCB.

·주요 구성. { Pointer.
state.
PID. = Context.
Counter
⋮

*Page 17, Material 4.

Context Switching.

· 다른 프로세스를 CPU에 넘겨줌.



남은 이슈.

· 그 다음 누가 실행?

· 메모리 어디에, 어떻게? allocation.

· 프로그램 크기 크면? Swapping. paging. 가상 메모리.

· Swapping

· 보안.

운영체제 - 4. 프로세스 (프로세스 메모리 주소).

#16.
NO. 24. 03. 20.

#메모리주소.

· 32비트와 64비트 : CPU와 메모리에 연결된 버스 수.

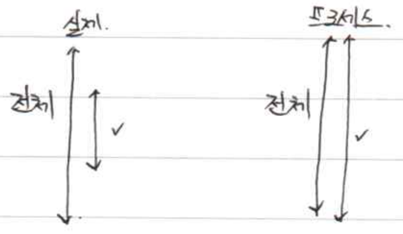
2^{32}까지 주소 할당 가능. 2^{64}까지 주소 할당 가능.

· CPU 주소공간보다 큰 메모리: 있어도 접근 불가.

#프로세스 주소 공간.

· ==가상 주소 공간== : 프로세스 입장... 메모리 독점한 것처럼 처리.

= 프로세스 주소 공간.

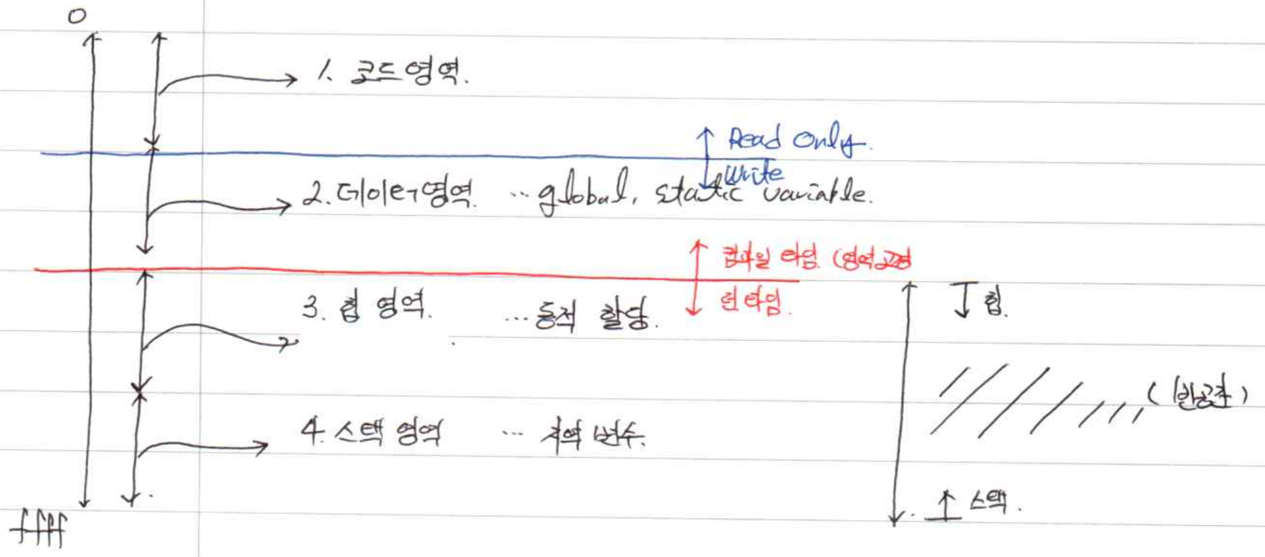


확장성. | · 물리 메모리는 한정적. 가상 메모리는 확장 가능!
| · Page Swap.

~~장점~~ 보안장점 | · 다른 프로세스 신경x.
| · 프로세스 간 메모리 격리.

* RTOS 등에는 없을 수도. (속도 확보 전략).

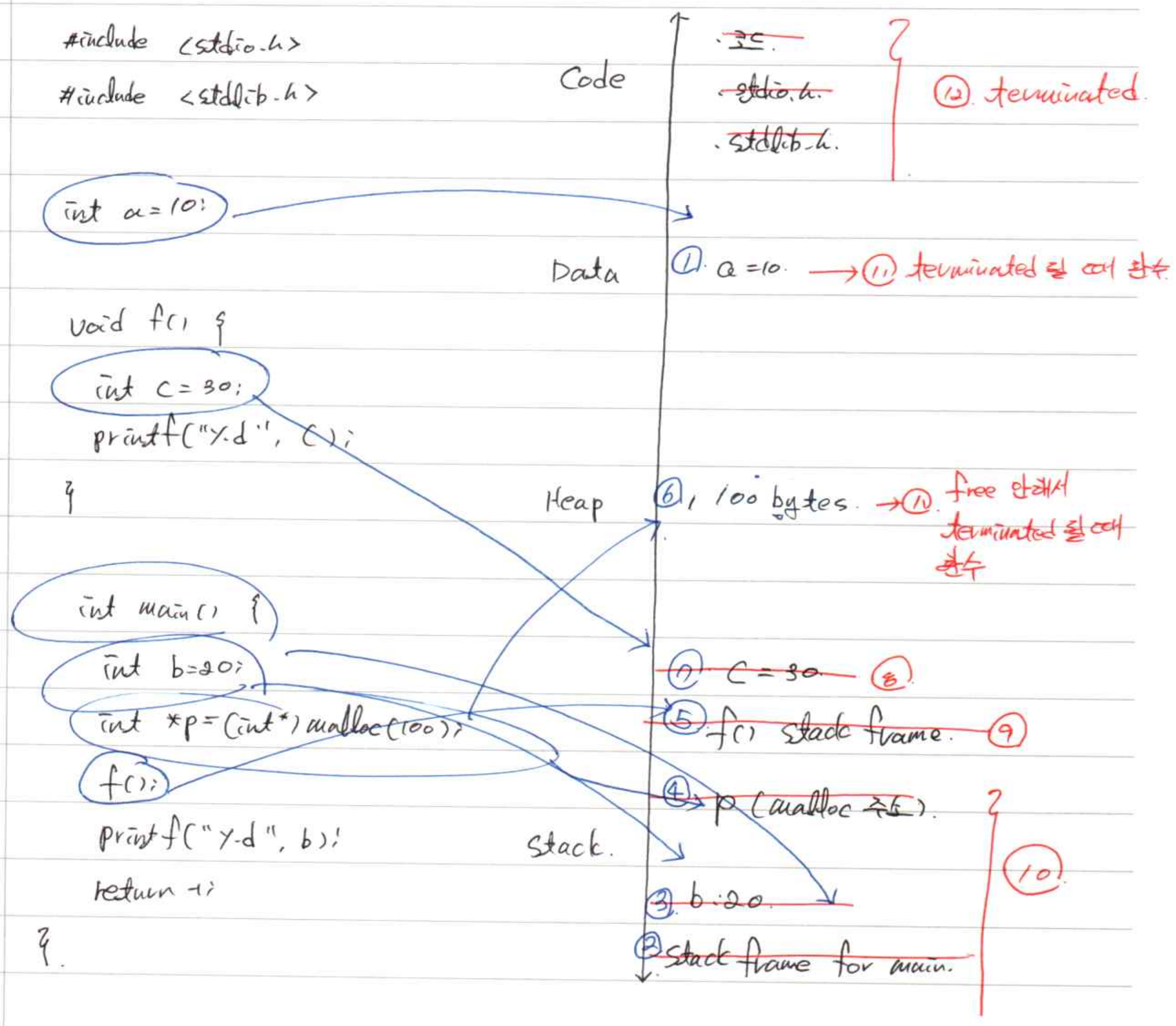
프로세스가 메모리 올라갈 때.



1. 코드영역.
2. 데이터영역. ... global, static variable.
3. 힙 영역. ... 동적 할당.
4. 스택 영역 ... 지역 변수.

* 왜? Page 29. Material 4.

메모리의 할당.



```
#include <stdio.h>
#include <stdlib.h>

int a=10;

void f() {
  int c=30;
  printf("%d", c);
}

int main() {
  int b=20;
  int *p=(int*)malloc(100);
  f();
  printf("%d", b);
  return 1;
}
```

프로세스 생성.   Page 41. Material 04.

fork.   Page 43. Material 04.

- fork() (linux)
- CreateProcess() (win).

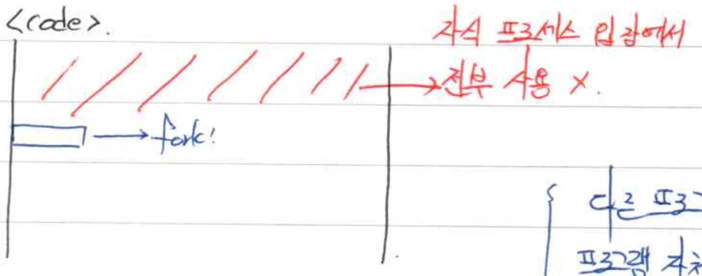
· 맨땅 헤딩보다 있는거 복사해서 바꿀거 바꾸자.

· 부모-자식 관계 파악.

· (PID, PPID, CPID.) 변경되는 정보들...

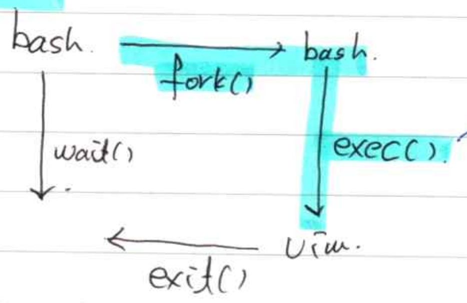
· 일반적으로 부모 프로세스는 자식 프로세스 완료될때까지 대기.
... `wait`

· fork의 장단 ... 빠름. / 컨텍스트 복사본 생성...비효율.
추가 작업 x 자원 상속. / pid=0 외에는 동작 불가.
효율적인 시스템 관리 / ↳ exec` :: 새 프로그램 실행 함수.



다른 프로그램 실행은
프로그램 자체를 복사하는 것으론
어려움.

· 새 프로세스의 실행.



→ "기존 task를 템플릿처럼 사용."

return @ main

```
int status;
fork().
wait(&status);
```

← 자식 프로세스가 0 전달하면 status로 0 할당. (포인터로 호출, 전달하는 것도 이것때문).

return / exit.

· 부모가 자식의 종료를 확인해야 프로세스가 정말로 종료

프로세스의 종료과정

① 모든 자원 반환.

② PCB … Terminated.

종료 코드 저장.

③ 자식프로세스, init에게 입양.

④ 부모에게 SIGCHLD 신호 전송.

… 부모: wait로 종료코드 읽음.

자식 프로세스 PCB 제거됨.

→ 못하면 좀비 프로세스가 됨.

wait과 WEXITSTATUS

```
pid = fork(); int status;
  ⋮
zombpid = wait(&status);
WEXITSTATUS(status);
```

→ 사실상 리턴이 두개.

→ 종료된 task의 pid (=cid) 리턴 (=준 리턴)

&status … Status 변수에 종료된 task의 상태 저장

→ 이 매크로는 가독성을 높이면서 status에 저장된 값 파싱하기 위해 존재.

idle과 kthreadd

* idle: 프로세스 1개도 안돌때 처리 난이도 < 억지로 1개라도 굴릴 때 처리 난이도.

* kthreadd: 커널 공간에서 실행되는 프로세스들의 조상. (커널 프로세스).

#셸. · 사용자와 OS(커널) 사이 인터페이스 역할 프로그램 / 환경.
(UI, 바탕화면 포함).

#세션. · 터미널을 사용해 시스템에 로그인했을 때 활동 상태 관리 단위.

(제어하는 인터페이스들 단위..)

· pts/4 , pty/0. ⇒ 세션 식별에 사용.
`ps ajx` → TTY 열 내용.
SID : 세션 ID.

· 세션 닫힘 ⇒ 세션에 부모를 둔 모든 task 다 종료.
→ ssh: timeout 존재, heartbeat 주기적 전송.
nohup 명령어. ⟶ SIGHUP 무시토록.
고아 프로세스화 → init으로 입양.
↳ 바로 죽는게 아니라 SIGHUP 신호 시 처리.

#파이프라이닝 cat in.in | python app.py. · 두개의 프로세스가 동시 작동함.

#프로세스 그룹. · PGID가 동일함.

· ex) ps ajx | cat | cat | cat | cat.
↳ 프로세스 5개, 모두 동일 PGID.