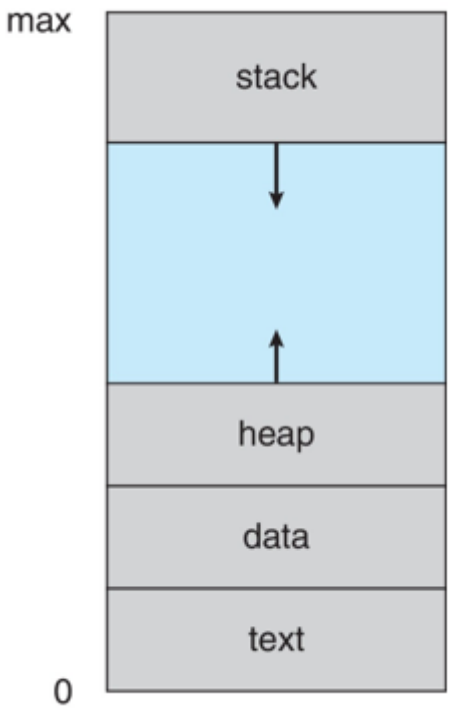
3장 프로세스

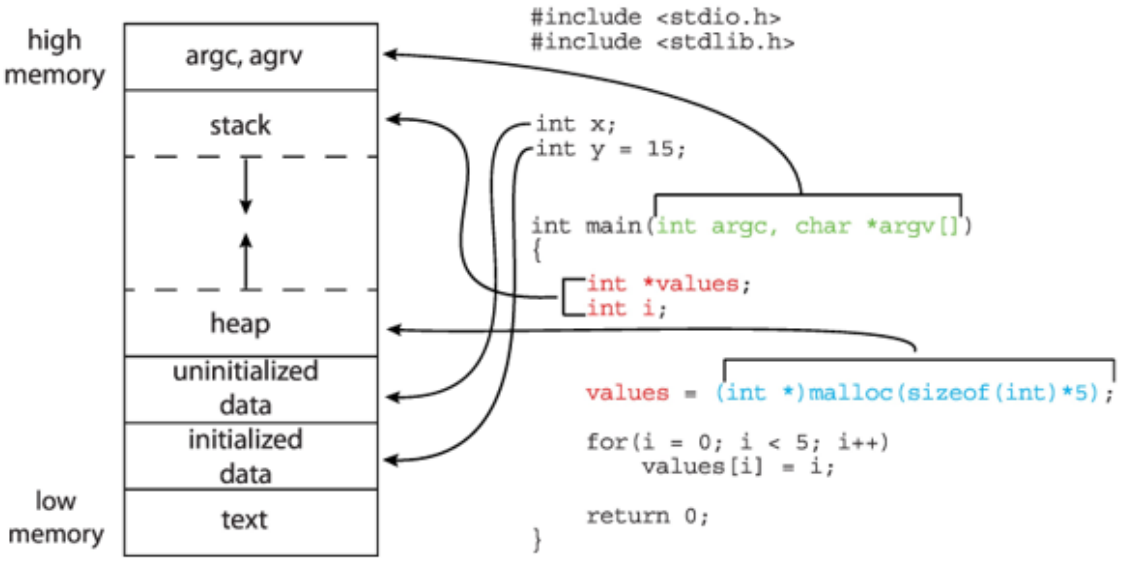
프로세스: OS의 작업단위, OS에서 관리됨

프로세스에 필요한 자원: cpu, memory, file, I/O device

메모리 섹션

- Text: 실행가능한 코드가 위치
- Data: 전역변수들(초기화,미초기화로 나뉨)
- Heap: 런타임시 동적으로 할당되는 메모리
- Stack: 함수,지역변수등을 일시적으로 저장하는 메모리



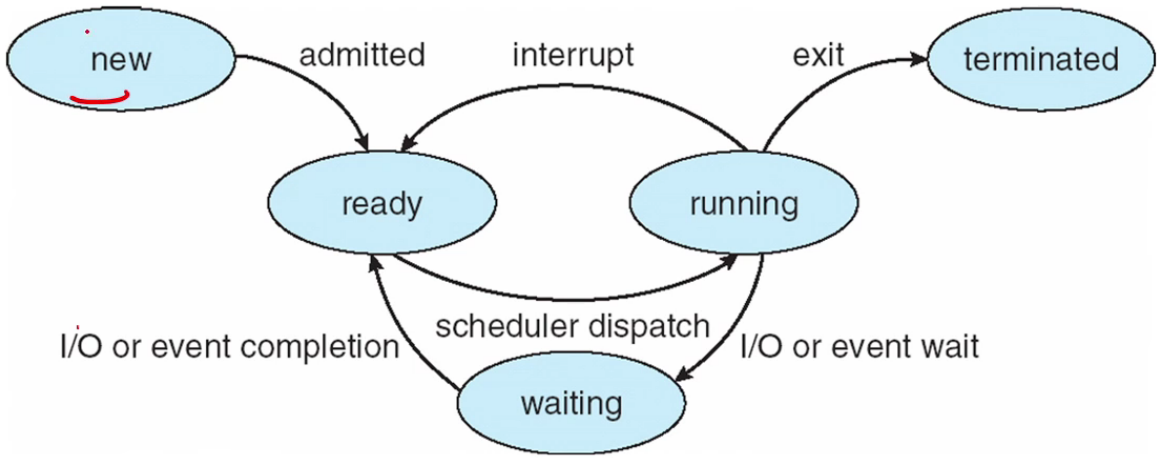


1. 전체코드가 text에 저장
2. main과 인자 argc,argv가 stack에 쌓임
3. mian내 values,i는 stack에 쌓임
4. malloc은 메모리를 확보해서 heap에 할당

프로세스 사이클

프로세스는 5가지의 상태를 사이클 순서에 따라 돌아간다.

- New: 프로세서 생성된 상태
- Running: 프로세스 실행중인 상태
- Waiting: 작업시작하기 전 대기 상태
- Ready: cpu배정 대기 상태
- Terminated: 프로세스 작업 완료된 상태



interrupt: 다른 프로세스를 먼저 실행할 경우

I/O or Event: I/O작업이 필요할 경우

PCB(Process Control Block),TCB(Task Control Block)

PCB: 프로세스를 관리학위한 구조체(struct)

보유중인 정보

- 프로세스의 상태정보
- Program Counter(Fetch 대상인 메모리) 정보
- CPU registers: context? 뒤에서 추가설명 나옴
- CPU 스케줄링 정보
- 메모리 관리 정보
- 사용자 정보
- I/O status 정보

단일 프로세스는 하나의 작업을 실행하는 프로그램

스레드는 프로세스내의 작업을 더 세분화 시킨 단위다.

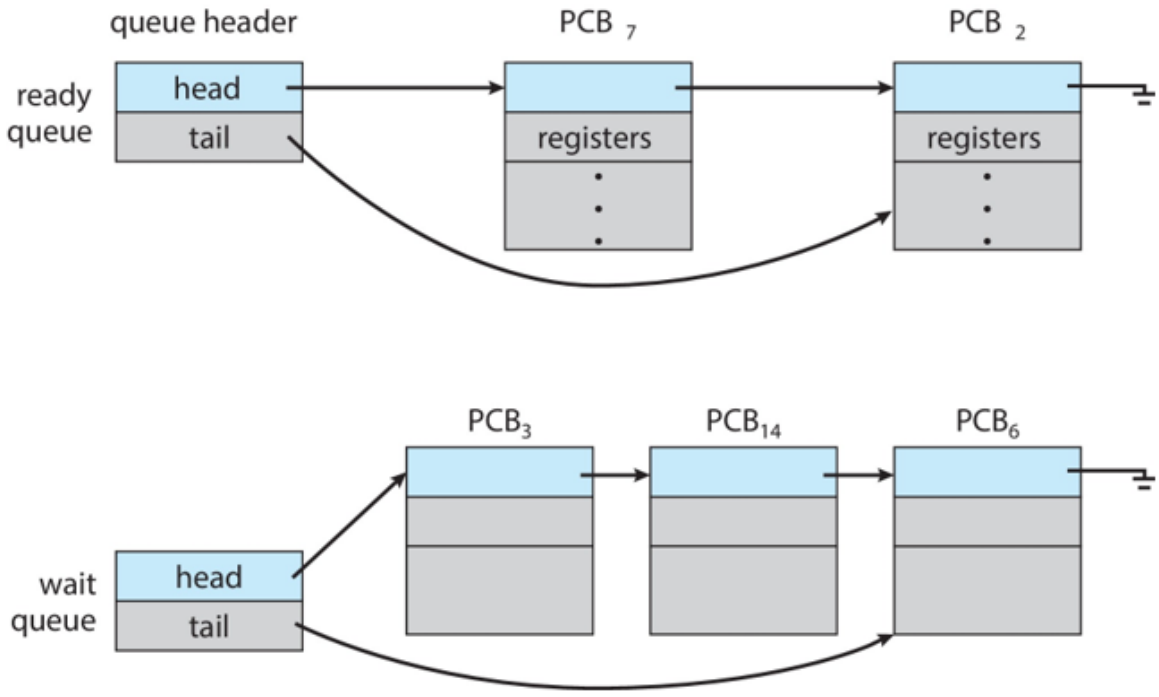
스레드는 프로세스내의 작업을 스케줄링해서 더 효율적이게 처리하기 위함이다.

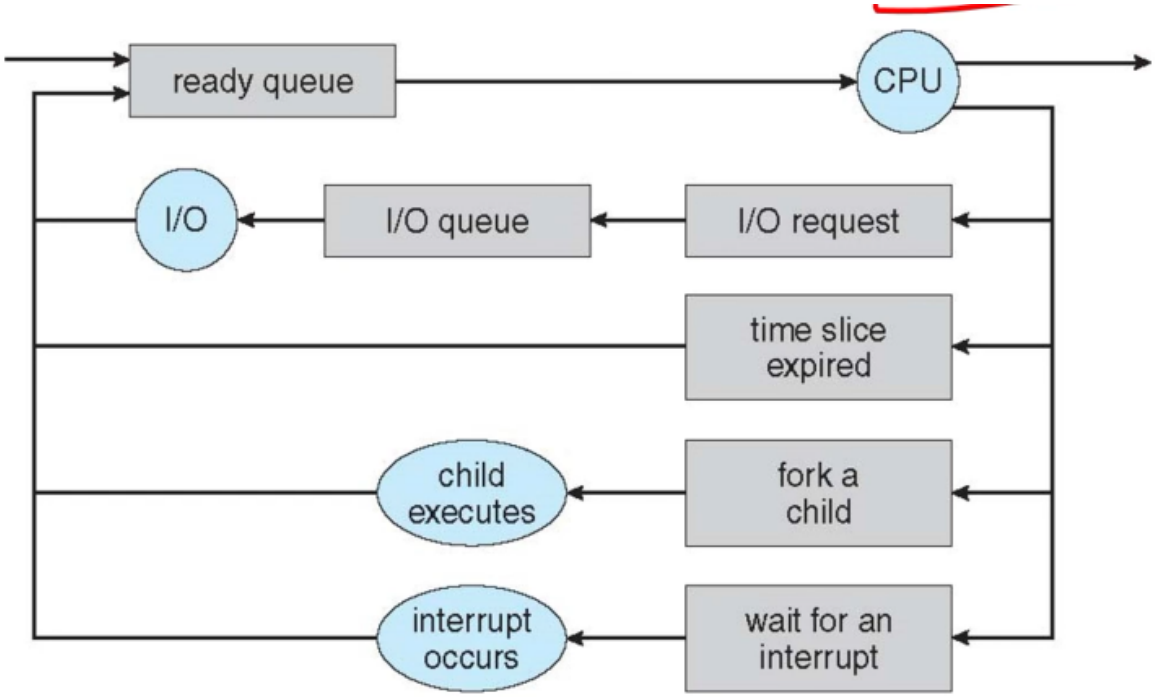
그래서 Multithreading으로 설계되는 경우가 많다. (Chapter4에서 설명)

Process scheduling

- **Multiprogramming**: 프로세스가 항상 돌아가서 CPU활성도를 최대로 하기 위함
- **Time Sharing:** cpu를 점유하는 시간을 프로세스들이 공유, 이를 통해 여러 CPU가 차례대로 실행해서 사용자는 여러 프로그램을 동시에 사용할 수 있다.
- **Scheduling Queues**: FIFO형태로 Ready또는 Waiting상태인 프로세스들을 각 Queue에 저장해 둔다.

아래 그림은 프로세스의 과정을 State에 따른 큐와 PCB로 저장된 형태를 사용해서 보여준다

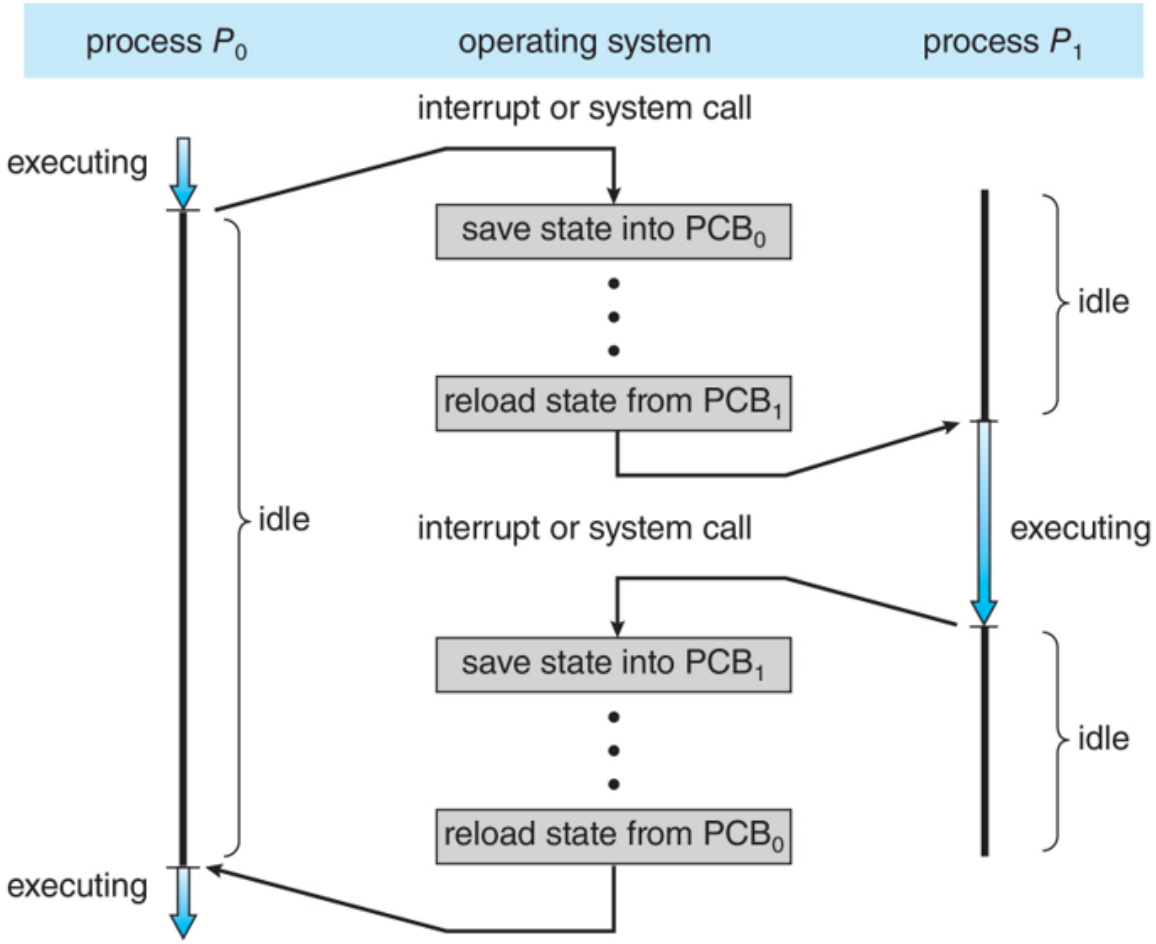




Context Switch

- **context**: 프로세스가 가용되고 있는 상태 즉 **PCB**자체를 뜻하기도 한다.
- **Context Switch**: CPU가 프로세스를 바꿔가며 실행하기 위해 프로세스의 State를 저장하고 복원

아래의 그림을 보면 P0와 P1의 작업을 병행하며 작업한다. 이때 하나의 프로세스의 context를 잠시 PCB에 저장하고 다른 프로세스의 context를 다시 불러들여 중단했던 지점부터 다시 필요한 작업을 실행하게 된다.



Process Creation

프로세스가 다른 프로세스를 생성할 수 도 있다

- Parent Process: 프로세스 생성자
- Child Process: 프로세스 피생성자

이런 프로세스의 프로세스 생성으로 트리형태의 프로세스관계가 형성된다.

이런 트리형태의 프로세스 관계를 **Tree of Processes**라고 부른다.

Child 프로세스가 생성되면

둘중 한가지 방식으로 진행

- Parent가 Child와 병행되어 실행
- Child가 실행되기까지 대기

둘중 한가지 방식으로 메모리 사용

- Parent와 똑같은 코드라면 Parent를 복제
- 새로운 프로그램(코드)라면 새로운 메모리에 적제

UNIX의 프로세스 관련 명령어

- fork(): 새로운 프로세스 생성
- exec(): 프로세스 실행(보통 생성된 child를 parent보다 먼저 실행할때)
- wait(): 프로세스 대기(child를 먼저 실행하고 parent를 대기시키기 위함)
- exit(): OS에게 프로세스 종료 및 자원 회수
- getpid(): 현제 프로세스의 pid값 리턴

Zombie: child인데 parent가 wait 안하고 계속 실행한 경우

Orphan: child인데 parent가 wait 안하고 종료 해버린 경우

위 두가지는 Daemon 혹은 Background 프로그램을 만들때 사용되기도 함

The contents below still need to be revised.

UNIX계열의 OS에서 fork()되면

- child에게는 pid=0이 전달
- parent에게는 child의 0이 아닌 pid가 전달

보통 parent를 먼저 하고 child를 실행해서

for()의 process생성 대상 예제

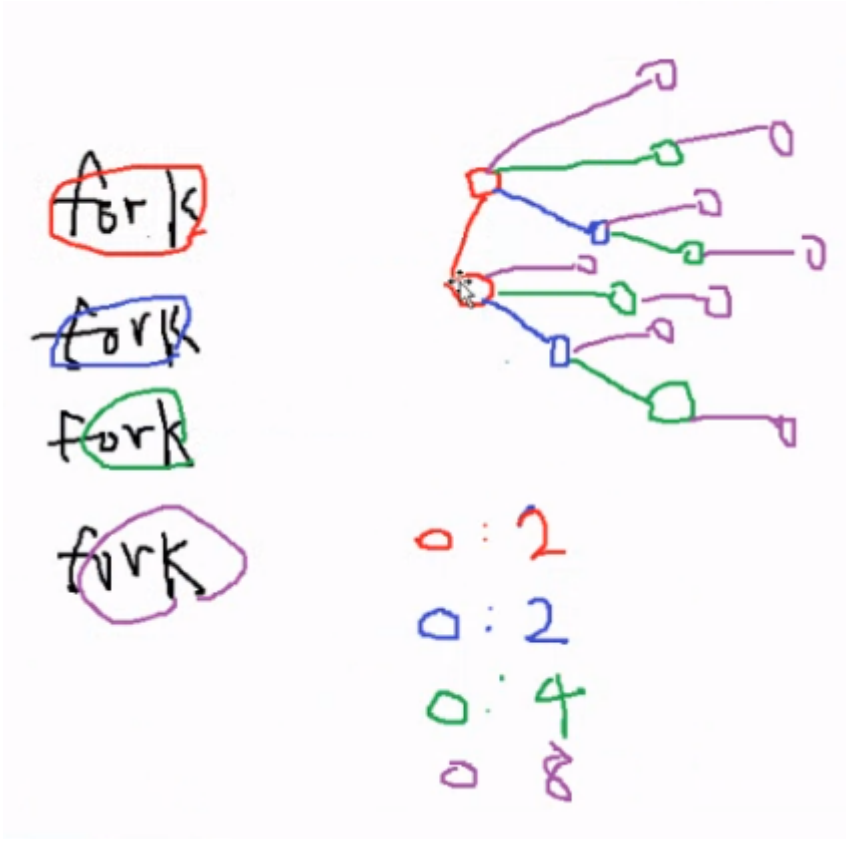
```
#include <stdio.h>
#include <unistd.h>

/*
 * How many processes are created?
 */

int main()
{
```

```
  int i;

  for (i=0; i < 4; i++)
    fork();
  return 0;
}
```



Interprocess Communication

프로세스간의 통신

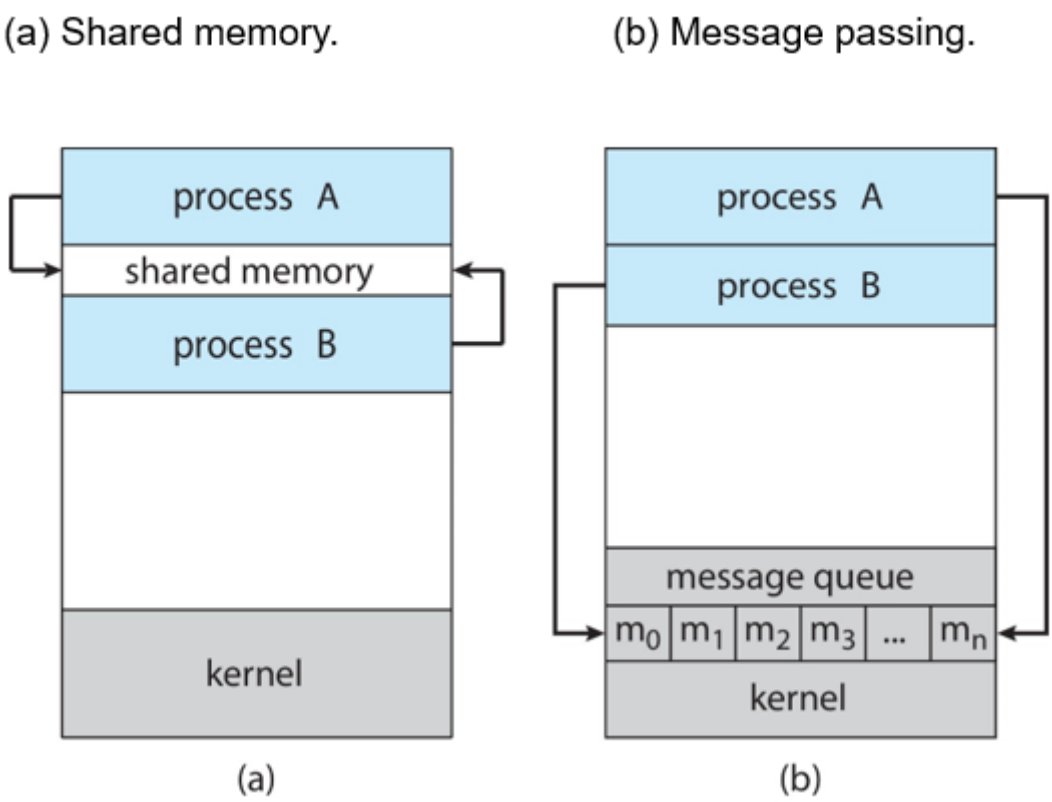
프로세스는 각자 독립적

데이터 공유하는 프로세스는 협업 프로세스

프로세스들은 커널영역 공유

IPC Models

- Shared Memory
- Message Passing



Producer-Consumer Problem

producer는 정보를 생성 consumer는 정보 소비

ex. web server가 생성 web browser가 소비

Shared-Memory

buffer라는 공간을 통해 정보 공유

producer가 채우고 consumer가 비움

buffer공유로 Prod-Consum 문제 해결

문제: 프로그래머가 buffer를 통해 공유하도록 구현해 주어야 함(번거로움)

Message-Passing

OS가 제공하는 기능

send() received()를 통해 프로세스간 정보통신

Communicaton LInks

- Physical:
  - Shared memory
  - Hardware bus
  - Network
- Logical:
  - Direct or indirect
  - Synchronous or asynchronous
  - Automatic or explicit buffering

Blocking NoN-blocking

Blocking: 상대가 수신했는지 확인해야 작업을 지속(Synchronous)

Non-Blocking: 상관없이 작업 지속 (Asynchronous)

POSIX shared memory