3. Processes and Threads

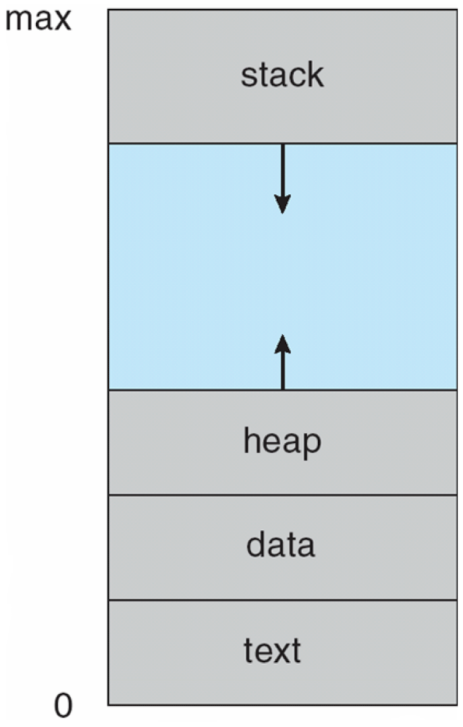
Process Concept

process는 기본적으로 순차적으로 실행

프로세스 포함정보

1. text (말 그대로 프로그램 코드)
2. stack
3. data (static, global 변수같은 것들, 주로 compile 타임에 결정되어 변동이 없는 것들)
4. heap

Process in Memory



이거 MAX값이 운영체제가 지원하는 bit수에 따라 달라짐

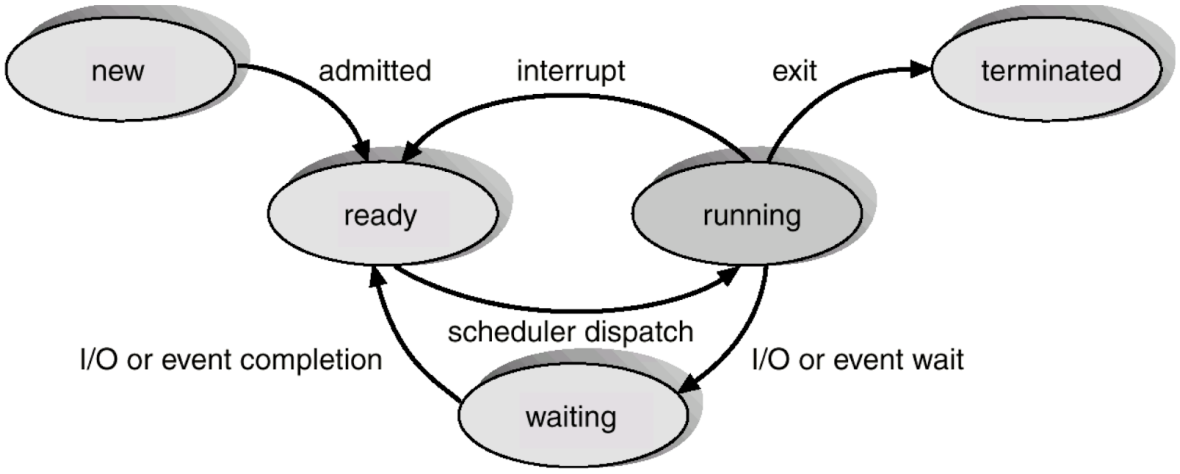
예를 들어 32bit 운영체제면 2147483647이 MAX값이 되는거고

64bit면 2^64 - 1이 MAX 값이 되겠죠?

그래서 여기서는 stack과 heap만 가변길이

data, text는 메모리에 올라온 순간 고정돼

Process State



new: process가 만들어지기만 하고 자원할당은 안된 상태

ready: 언제든지 CPU를 할당받을 수 있는 상태

running: cpu가 현재 실행 중인 process

waiting: event가 발생해서 잠시 대기중인 프로세스

terminated: 종료된 프로세스

그러면 running에서 ready로 가는 경우가 뭐가 있을까?

→ 현재 프로세스가 할당받은 시간을 다 소모한 경우

Process Control Block

특정 프로세스와 연관된 여러 정보를 수록한다

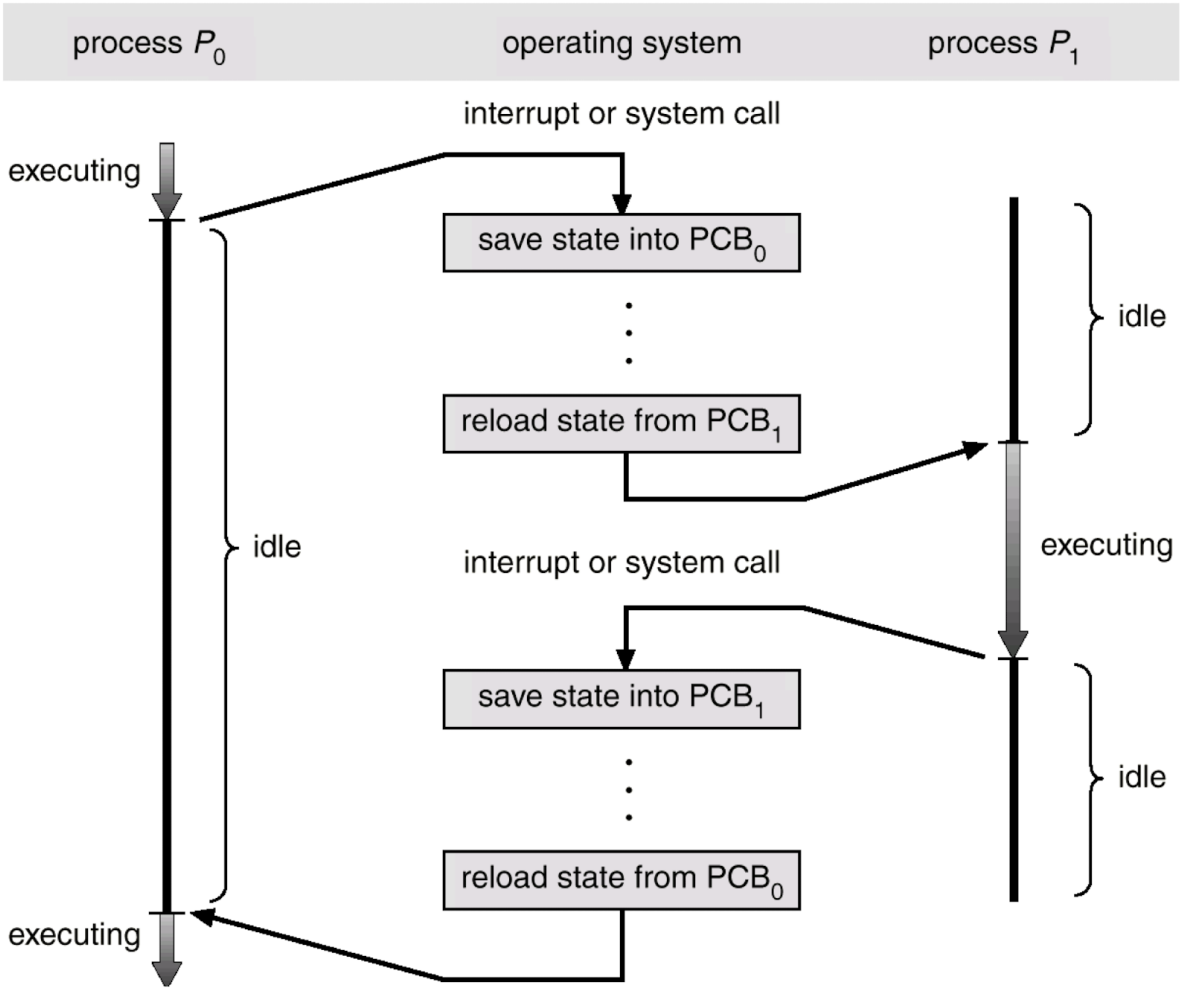
그래서 갖고 있는거

1. Process state
2. Program counter
3. CPU registers
4. CPU scheduling information
5. Memory-management information
6. Accounting information
7. I/O status information

근데 솔직히 말해서 이거 갖고 있는 게 뭔지는 몰라도 된다고 생각합니다…

그냥 process 를 들고 나르는 애가 PCB다 정도만 알면 되지 않을까요?

CPU Switch from Process to Process(Context Switching)

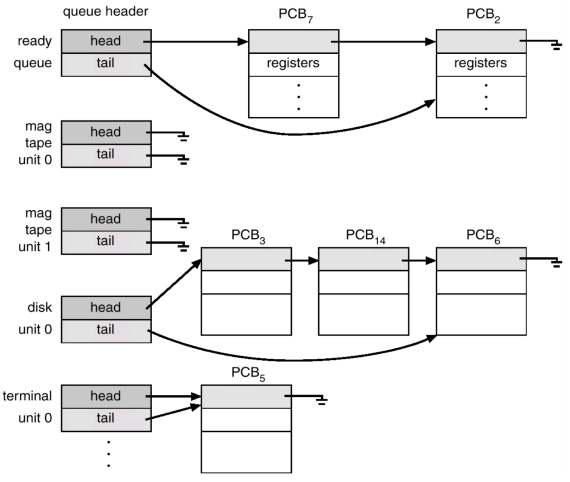


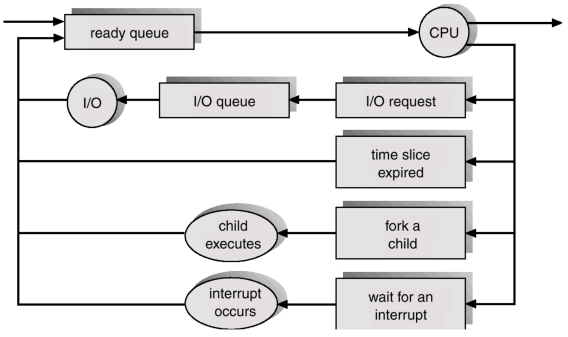
save state into PCB_0 ~ reload state from PCB_1 사이에 CPU가 바보됨

저 상황이 딱 위에서 running→ready로 바뀌는 상황인데 이 때 자원소모가 좀 심해

(CPU가 일을 안하니까)

그래서 자원소모가 심하니까 최대한 필요할 때만 context switch가 발생하도록 Process Scheduling을 해





대충 이런식으로 관리하는 거만 알면 될듯?

Scheduler

1. Long-term scheduler (or job scheduler) **INFREQUENT**

   new → ready 를 결정하는 스케줄러

   말 그대로 process에 자원을 할당하는 과정 (process 생성)

   이게 프로세스의 개수를 결정(the degree of multiprogramming)

2. Short-term scheduler (or CPU scheduler) **FREQUENT**

   ready 상태에 있는 process 중에서 누구에게 CPU를 할당할건지 결정

   I/O 작업은 interrupt가 발생하는 거라서 이거도 CPU 할당과 관련 있음

   그래서 I/O interrupt 발생시 running → waiting으로 변경되는 건 short-term scheduler가 해줘

3. Medium-term scheduler

   disk → memory

   memory → disk

   이렇게 메모리 영역에서 프로세스를 관리하는게 Medium-term scheduler

I/O bound process (키보드 입력 하는 것들…)

- CPU 사용은 짧게 하는데, 빈번하게 발생

CPU bound process

- 일단 한번 잡으면 CPU를 길게 사용해, 대신 잘 안잡아

나중에 이거 관련해서 우선순위 결정하는 거 나옴

Context Switch (Process 전환)

현재 프로세스가 돌고 있는 곳 ⇒ Register

PCB ⇒ memory에 저장됨 (생각해보면 당연한게 new 로 인해서 프로세스가 생성되면 그걸 medium-term scheduler가 memory 영역으로 불러오니까)

그래서 현재 프로세스와 대기중인 프로세스(ready 상태의 프로세스)를 교환

store: Register → PCB

Restore: PCB → Register

근데 기본적으로 이거 되게 overhead가 커

1. memory로 들어가는 순간 속도가 압도적으로 느려져 (Register, cache가 빠르니까)
2. context-switch 시간 동안 CPU가 바보가 돼
3. context-switch 이후에 cache flushing 발생
   a. 이게 한번에 발생되는 건 아니고, 새로운 프로세스가 오면서 cache는 변하지 않아
   b. 근데 새로운 프로세스가 요구하는 건 cache에 없어
   c. 그래서 이 프로세스가 필요로 하는 데이터를 메모리에서 가져와야 되는데 이래서 초반에 cache miss가 연달아 발생해

Process Creation (process가 process 생성)

이거 fork 써서 생성, exec으로 이미지 변경

fork 쓰면 자식 프로세스는 pid가 0, parent와 child가 독립적으로 실행(병렬적으로), 그리고 모든 값을 그대로 복사하기 때문에 해당 code line부터 병렬적으로 실행

exec 쓰는 순간 자식이 부모로부터 독립

각각의 process들은 pid를 갖게 되는데 이걸 통해서 resource sharing 권한을 조절할 수 있어

(부모-자식 간의 관계에서 권한 조정 가능)

기본적으로 parent는 child process가 종료될 때까지 기다림

근데 parent를 강제로 종료시켜버리면? ⇒ 이게 zombie process

```
#include <stdio.h>

void main (int argc, char *argv[])
{
  int pid;

  /* fork another process */
  pid = fork();

  if (pid < 0) { /* error occurred */
    fprintf (stderr, “Fork Failed\n”);
    exit(-1);
  }
  else if (pid == 0) { /* child process */
    execlp(“/bin/ls”, “ls”, NULL);
  }
  else { /* parent process */
    /* parent will wait for the child
       to complete */
    wait(NULL);

    printf(“Child Complete\n”);
    exit(0);
  }
}
```

결과가 뭐가 나올까요?

▼ 답?

일단 process 가 진행

fork 호출되는 순간 독립적인 프로세스가 2개가 됨 (부모, 자식)

pid == 0 인 애가 자식 프로세스

pid < 0 ⇒ 일단 이건 고려 안함, 정상적으로 잘 되겠지

pid == 0 ⇒ 자식 프로세스

exec가 호출되면서 ls 실행하러 떠나고 이 때부터 부모 자식 독립(이거 부모 자식 간의 관계는 유지되는데 독립이라는게 데이터가 달라진다는거, pid 값 같은 거

parent는 자식이 끝날 때까지 기다려 (이 때 parent는 block, 우리 입출력할 때 block 그거 맞음)

ls 작업이 끝나고 parent가 Child 자원을 회수하면서 프로그램 exit

그러면 만약에 wait(NULL) ⇒ 이 코드가 없다면

child, parent 중 뭐가 더 먼저 끝날까?

Interprocess Communication

이거 분산시스템에서 많이 쓰이는 기법이라고 하는데 소신발언하면 잘 모르겠습니다…

그냥 다른 프로세스끼리 소통할 수 있는 방법이 2가지 있는데

그 중 하나가 Shared memory 고 다른 하나가 Message passing 인데

분산시스템은 후자를 많이 채택한다고 들었습니다

Producer-Consumer Model(Shared Memory)

C++ 기준으로 전역 변수 쓰면 공유 가능합니다.

자바 기준으로는 아마 static이 아닐까 싶은데요? (그래서 static 변수들은 멀티스레드 환경에서 안전하게 처리하려고 synchronized 키워드 쓰는걸로 아는데 자세히 아시는분?))

전역변수로 배열 하나를 두고 Producer가 배열을 채워넣음

그럼 반대로 Consumer가 배열에서 값을 빼먹음

근데 배열 크기가 100인데 이미 100을 다 채우면 producer는 어떻게 해야할까?

⇒ 이게 bounded-buffer problem

Message Passing

다시 말씀드리지만 아는게 없습니다.. 그냥 메시지 주고 받는다 정도만 알아요…

Synchronization

1. Blocking

   이거는 말그대로 프로세스가 정지하는 거

   예를 들어 자바에서 입력 받을 때 BufferedReader나 Scanner 쓸 때 입력을 받기 전(엔터를 치기전) 까지 프로세스가 멈춤 입력이 처리되는 순간 CPU에 interrupt를 걸어서 원래의 process를 처리하도록 만듦

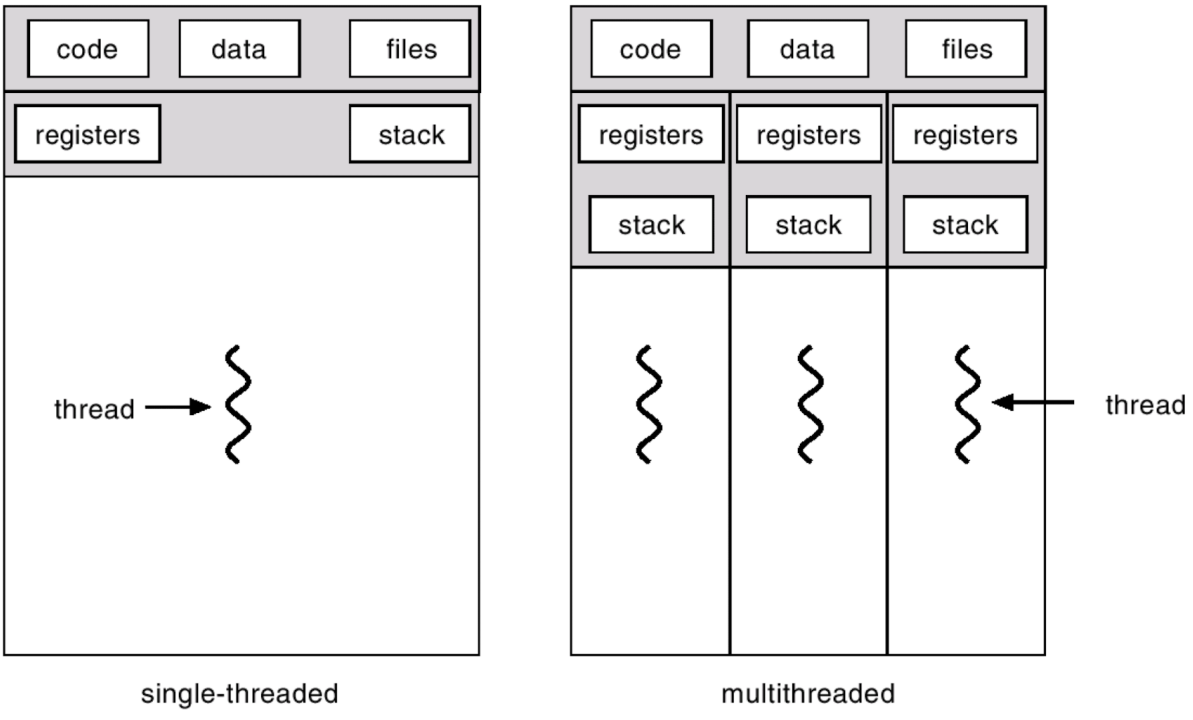
2. Non-blocking

   이건 주로 웹에서 프론트에서 많이 쓰이지 않나? 그 ajax, axios 같이 데이터 비동기 통신할 때 많이 쓰이는 방법이라고 아는데

   데이터가 올 때까지 기다릴 수는 없으니까 일단 할 수 있는 다른 것들을 먼저 하고 데이터가 도착하면 그걸 비동기적으로 처리하는거

Threads (Light Weight Process)

프로세스랑 다를게 없는데 그냥 좀 더 가벼워진 프로세스 라고 생각하시면 됩니다



이 그림에서 알 수 있겠지만

기본적으로 하나의 스레드는 각각 레지스터와 스택, PC(Program Counter)를 별도로 가져

대신 code, data, file은 서로 공유 (resource는 공유)

그래서 Thread의 장점이 뭔가요? RRES

1. Responsiveness
2. Resource Sharing
3. Economic
4. Scalability

그럼 단점은?

⇒ 하나의 스레드로 인해 모든 스레드가 블록될 수 있음

생각해보면 간단한게 스레드도 결국 프로세스라 user-level의 main thread 하에 관리 되는데 이 main thread가 죽어버리면 답이 없음…

그러면 kernel thread는요? block 안해요

Threading Issues

fork에는 2가지 버전이 있어

1. fork 직후에 exec이 바로 호출되는 경우

   이 경우에는 아예 새로운 프로그램을 만들겠다는 소리여서 모든 스레드를 복사할 필요가 없어

   그냥 main 스레드만 복사하고 그걸로 새로운 프로세스 생성하러 가면 돼
2. fork 만 호출되는 경우

   이건 말 그대로 복제본을 만들겠다는 소리라 (병렬 실행) 모든 스레드 싹 다 복사해야돼

   (stack, pc, register …)

Thread cancellation

1. Asynchronous cancellation

   원래는 스레드가 중단되면 자원회수(wait)해야되는데 그거 안하고 즉시 중단

   ⇒ 이걸로 발생할 수 있는 문제

   1. Resource De-allocation problem

      동적으로 할당된 메모리를 해제하지 않고 스레드 종료시키는거 이러면 프로그램 내에 메모리가 계속 잡혀서 손해 봄

      lock 관련 ⇒ 만약 스레드가 Lock을 먹고 종료돼버리면 이 lock은 누가 풀어줌? ⇒ 바로 데드락

      그럼 이거 왜 씀?

      역설적으로 데드락 같은 상황이 발생할 때 스레드 강제종료 시켜버릴 수 있음 + 빠르게 종료할 수 있어
   2. Shared data integrity problem

      이거는 공유되는 데이터 무결성이 깨지는 거

      좀 구체적으로 가보면 스레드 총 10개가 있고

      각각의 스레드가 10씩 증가 시키기로 했는데 값을 확인하러 가보니 100이 아니라 95같이 데이터가 어긋나는 상황
2. Deferred cancellation

   정상적인 종료하는거

   당연히 자원의 신뢰성이 보장되겠지?

   단점이라면 느리다는 거