4. Process Management

프로세스 생성 (Process Creation)

부모 프로세스 (Parent process)가 자식 프로세스 (Children process) 생성

프로세스의 트리(계층 구조) 형성

프로세스는 자원을 필요로 함

- 운영체제로부터 받는다
- 부모와 공유한다

자원의 공유

- 부모와 자식이 모든 자원을 공유하는 모델
- 일부를 공유하는 모델
- 전혀 공유하지 않는 모델
    - **자원 경쟁**

수행 (Execution)

- 부모와 자식은 공존하며 수행되는 모델
- 자식이 종료(terminate)될 때까지 부모가 기다리는 (wait)모델

Copy-on-write (COW)

⇒ 부모와 자식 프로세스가 똑같다면 그냥 복사해서 사용하면 되지만 다른 내용이 들어갈 때 이제 COW를 통해서 복사한 후에 쓰게 됨

주소 공간 (Address space)

- 자식은 부모의 공간을 복사함 (binary and OS data)
- 자식은 그 공간에 새로운 프로그램을 올림

유닉스의 예

- **fork()** 시스템 콜이 새로운 프로세스를 생성
    - 부모를 그대로 복사 (OS data except PID + binary)
    - 주소 공간 할당
- fork 다음에 이어지는 **exec()** 시스템 콜을 통해 새로운 프로그램을 메모리에 올림

⇒ **복사 후에 덮어 씌워야 함 (매번 새로 만드는 것이 X)**

프로세스 종료 (Process Termination)

프로세스가 마지막 명령을 수행한 후 운영체제에게 이를 알려줌 (**exit**)

- 자식이 부모에게 output data를 보냄 (via **wait**).
- 프로세스의 각종 자원들이 운영체제에게 반납됨

부모 프로세스가 자식의 수행을 종료시킴 (**abort**)

- 자식이 할당 자원의 한계치를 넘어섬
- 자식에게 할당된 태스크가 더 이상 필요하지 않음
- 부모가 종료(exit) 하는 경우
  - 운영체제는 부모 프로세스가 종료하는 경우 자식이 더 이상 수행되도록 두지 않는다
  - 단계적인 종료

시스템 콜

fork() 시스템 콜

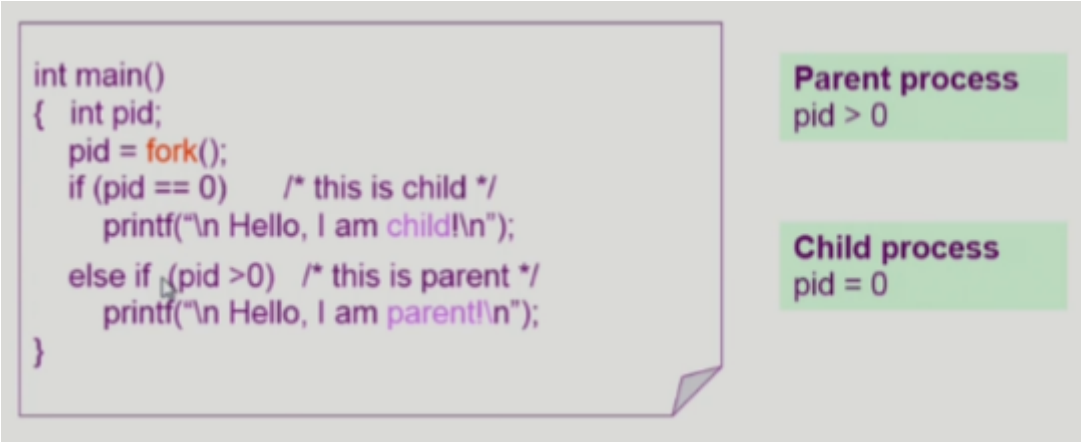
A process is created by `fork()` system call.

- create a new address space that is a duplicate of the caller.

fork() 후에, 부모 프로세서는 fork()후의 코드부터 실행함

자식 프로세서도 fork()까지 실행했구나의 기억을 가진채로 복사 → 거기부터 실행

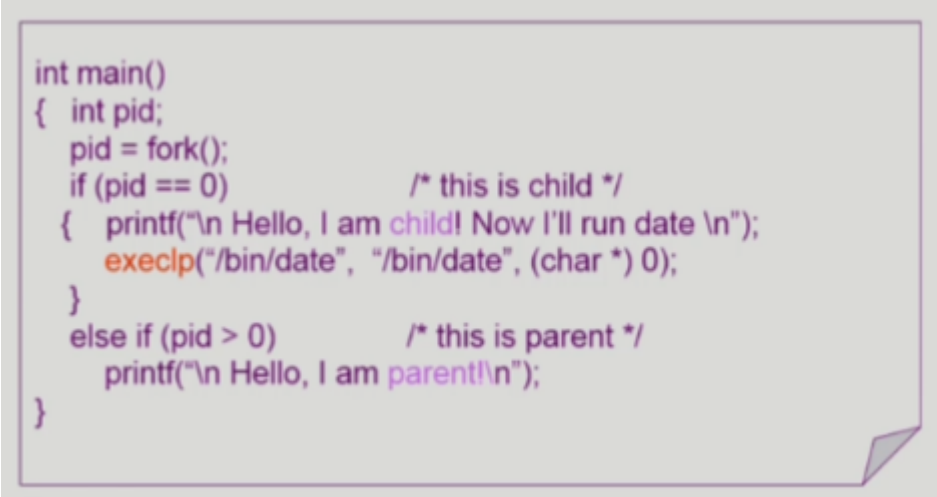
부모는 fork를 했기 때문에 결과값이 양수임. 부모와 자식은 fork()값으로 구분할 수 있음



exec() 시스템 콜

A process can excute a different program by the `exce()` system call.

- replaces the memory image of the caller with a new program.



```
int main()
{   int pid;
    pid = fork();
    if (pid == 0)                    /* this is child */
    {    printf("\n Hello, I am child! Now I'll run date \n");
         execlp("/bin/date",  "/bin/date", (char *) 0);
    }
    else if (pid > 0)                /* this is parent */
         printf("\n Hello, I am parent!\n");
}
```

exec() 시스템 콜을 만나면 처음부터 다시 → 완전 새로운 프로그램으로 덮어씌움

그 프로그램이 끝나면 거기서 끝!

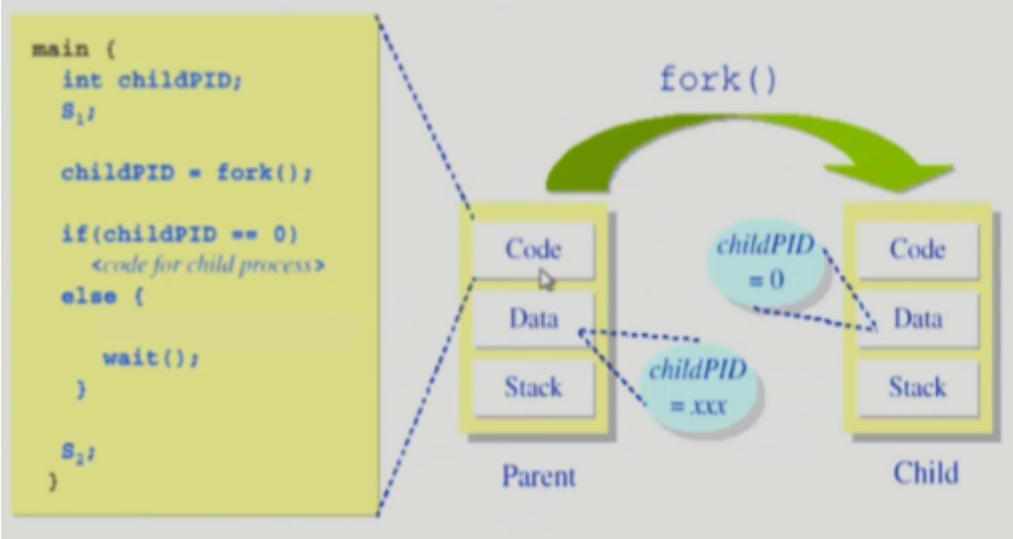
한번 exec하면 다시 돌아올 수 없음. 그냥 완전 새로운 프로그램

exec 이 꼭 자식 프로세스에만 할 수 있는건 아님 그냥 프로그래밍 중에 사용할 수 있음

wait() 시스템 콜

프로세스 A가 wait() 시스템 콜을 호출하면

- 커널은 child가 종료될 때까지 프로세스 A를 sleep시킨다 (block 상태)
- Child process가 종료되면 커널은 프로세스 A를 깨운다 (ready 상태)



wait() 는 자식이 종료될 때까지 기다리는 모델. ⇒ 자식과 경쟁하지 X

exit() 시스템 콜

프로세스의 종료

- 자발적 종료
  - 마지막 statement 수행 후 exit() 시스템 콜을 통해
  - 프로그램에 명시적으로 적어주지 않아도 main 함수가 리턴되는 위치에 컴파일러가 넣어줌
- 비자발적 종료
  - 부모 프로세스가 자식 프로세스를 강제 종료시킴
    - 자식 프로세스가 한계치를 넘어서는 자원 요청
    - 자식에게 할당된 태스크가 더 이상 필요하지 않음
  - 키보드로 kill, break 등을 친 경우
  - 부모가 종료하는 경우
    - 부모 프로세스가 종료하기 전에 자식들이 먼저 종료됨
    - 부모가 자식보다 먼저 죽으면 그 아래에 있는 자식의 자식의 .. 자식들 모두 종료

프로세스와 관련한 시스템 콜

- `fork()` create a child (copy)
- `exec()` overlay new image
- `wait()` sleep until child is done
- `exit()` frees all resources, notify parent

프로세스 간 협력

독립적 프로세스 (Independent process)

프로세스는 각자의 주소 공간을 가지고 수행되므로 원칙적으로 하나의 프로세스는 다른 프로세스의 수행에 영향을 미치지 못함

협력 프로세스(Cooperating process)

프로세스 협력 메커니즘을 통해 하나의 프로세스가 다른 프로세스의 수행에 영향을 미칠 수 있음

프로세스 간 협력 메커니즘 (IPC : Interprocess Communication)

메세지를 전달하는 방법

message passing : 커널을 통해 메세지 전달

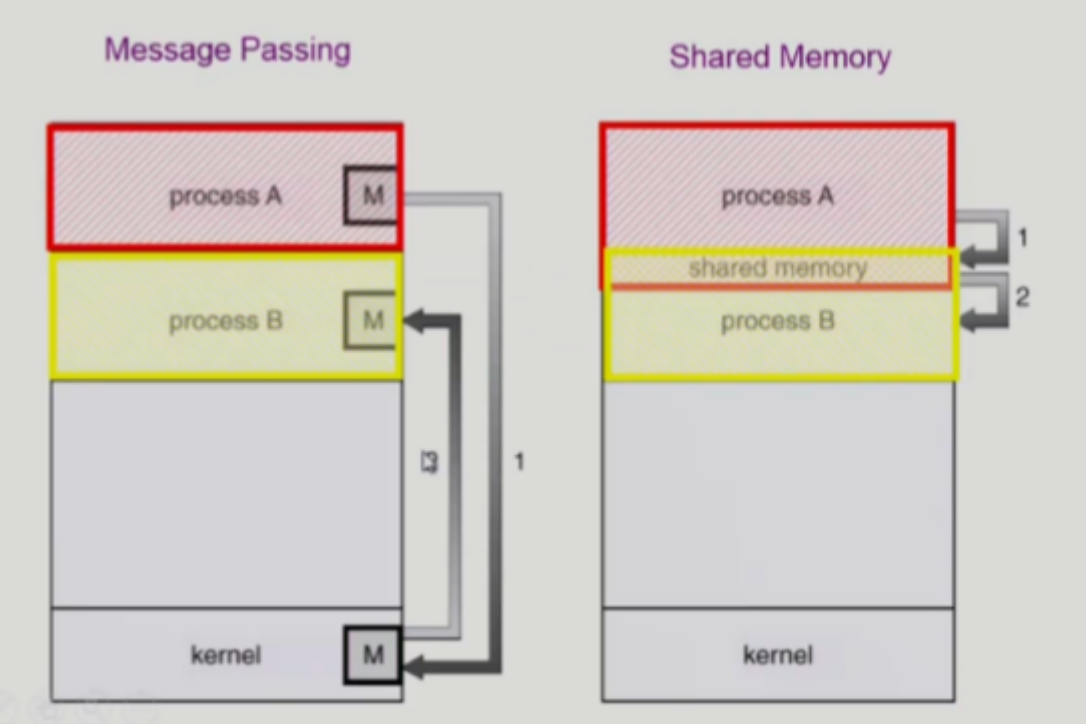
주소 공간을 공유하는 방법

shared memory : 서로 다른 프로세스 간에도 **일부 주소 공간을 공유**하게 하는 shared memory 메커니즘이 있음

- 일부 주소 공간을 공유해야 하므로 두 프로세스는 서로 신뢰할 수 있는 관계여야함

* **thread** : thread는 사실상 하나의 프로세스이므로 프로세스 간 협력으로 보기는 어렵지만 동일한 process를 구성하는 thread들 간에는 주소 공간을 공유하므로 협력이 가능

Interprocess Communication



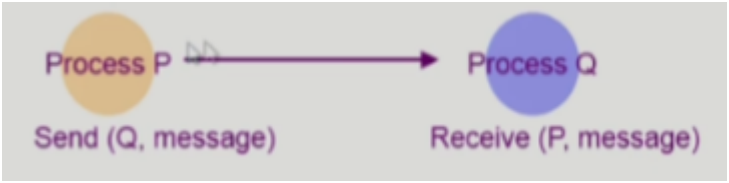
Message Passing

Message system

프로세스 사이에 고유 변수 (shared variable)를 일체 사용하지 않고 통신하는 시스템

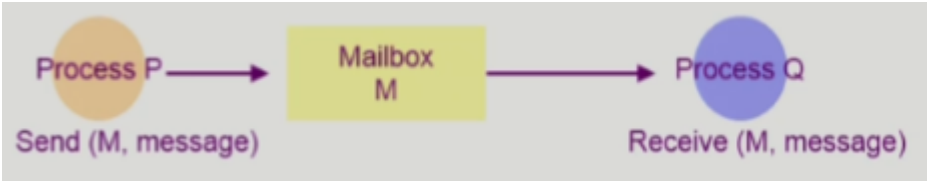
Direct Communication

통신하려는 프로세스의 이름을 명시적으로 표시



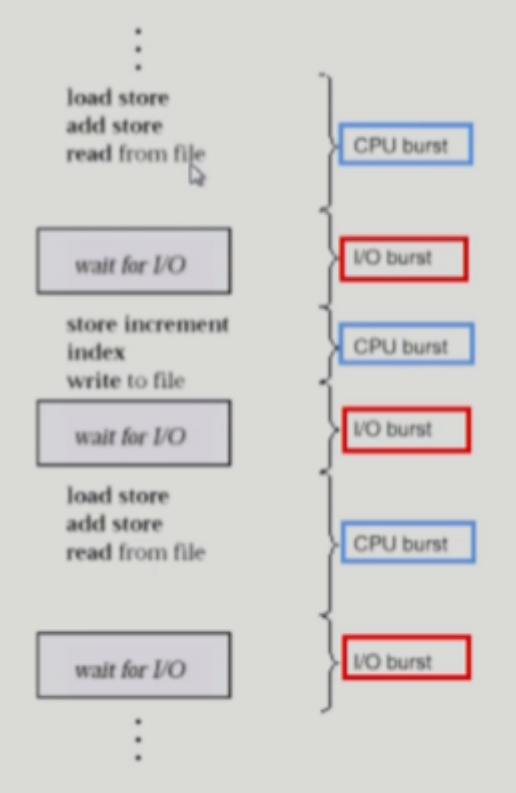
Indirect Communication

mailbox(또는 port)를 통해 메시지를 간접 전달



CPU Scheduling

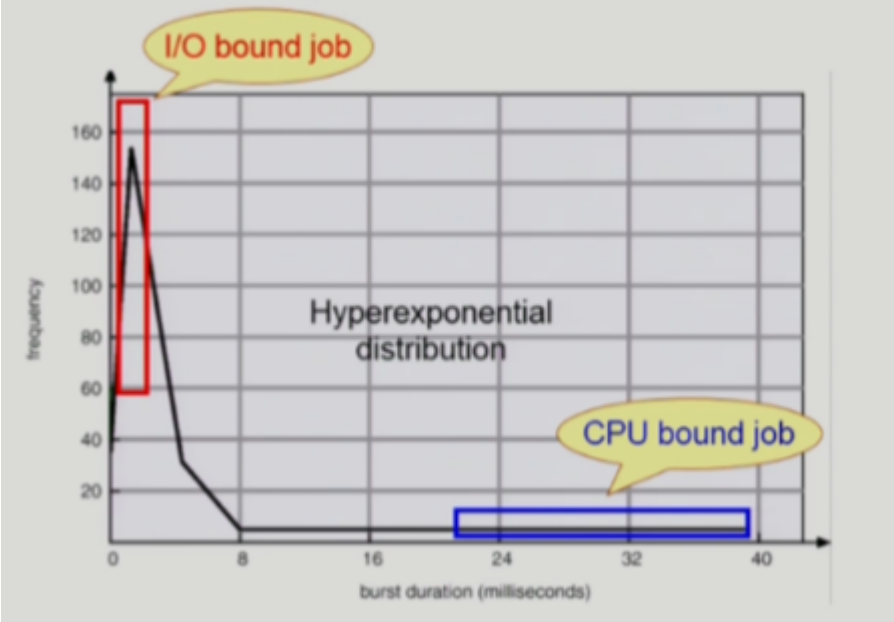
CPU and I/O Bursts in Program Execution



→ I/O가 자꾸 끼어듦.

CPU bursts와 I/O bursts 가 짧아짐

CPU-burst Time의 분포



여러 종류의 job(=process)이 섞여 있기 때문에 CPU 스케줄링이 필요하다

- Interactive job에게 적절한 response 제공 요망
- CPU와 I/O 장치 등 시스템 자원을 골고루 효율적으로 사용

CPU가 job을 좀 오래 잡고 있으면 I/O 와 같은 Interactive job에서 사용자는 답답함을 느끼게 됨

프로세스의 특성 분류

프로세스는 그 특성에 따라 2가지로 나뉜다.

I/O-bound process

- CPU를 잡고 계산하는 시간보다 I/O에 많은 시간이 필요한 job
- many short CPU bursts

CPU-bound process

- 계산 위주의 job
- few very long CPU bursts

출저 : kocw 운영체제 - 반효경 교수님 (이화여자대학교)

http://www.kocw.net/home/search/kemView.do?kemId=1046323