4. Process Management

▼ 목차

1. 프로세스의 생성

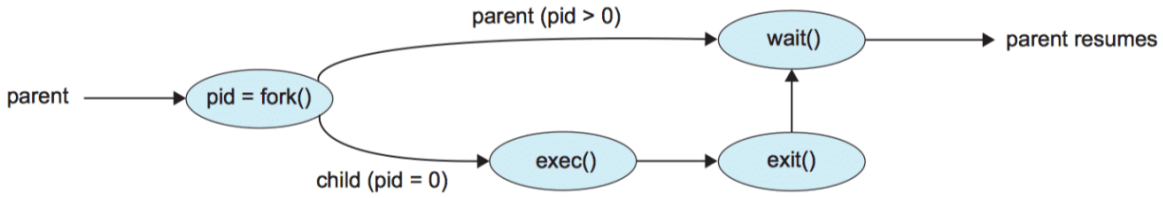


Figure 3.10 Process creation using the `fork()` system call.

운영체제는 부모 프로세스(Parent Process)의 Address space를 '복제'하여 자식 프로세스(Child Process)를 만든다.

- 자식 프로세스에 해당하는 PCB 역시 별도로 생성한다.
- 자식 프로세스는 할당 받은 주소 공간에 새로운 프로그램을 올린다.

프로세스의 생성은 다음의 두 단계로 진행된다.

- 1단계: `fork` **system call** — 부모 프로세스를 복제해서 자식 프로세스 생성
- 2단계: `exec` **system call** — 자식 프로세스가 주소공간에 새로운 프로그램을 올림

→ 두 단계는 서로 '독립적'이다. 즉, 1단계의 fork를 수행하지 않아도 exec을 통해 새로운 프로세스를 생성할 수 있다.

→ system call이라는 단어에서 알 수 있듯이 프로세스의 생성은 사용자 프로그램이 운영체제에 요청을 통해 의해 수행된다. 프로세스를 만들어 달라고 운영체제에 부탁하는 것.

최초의 프로세스 하나부터 시작해서 프로세스의 트리 구조를 형성

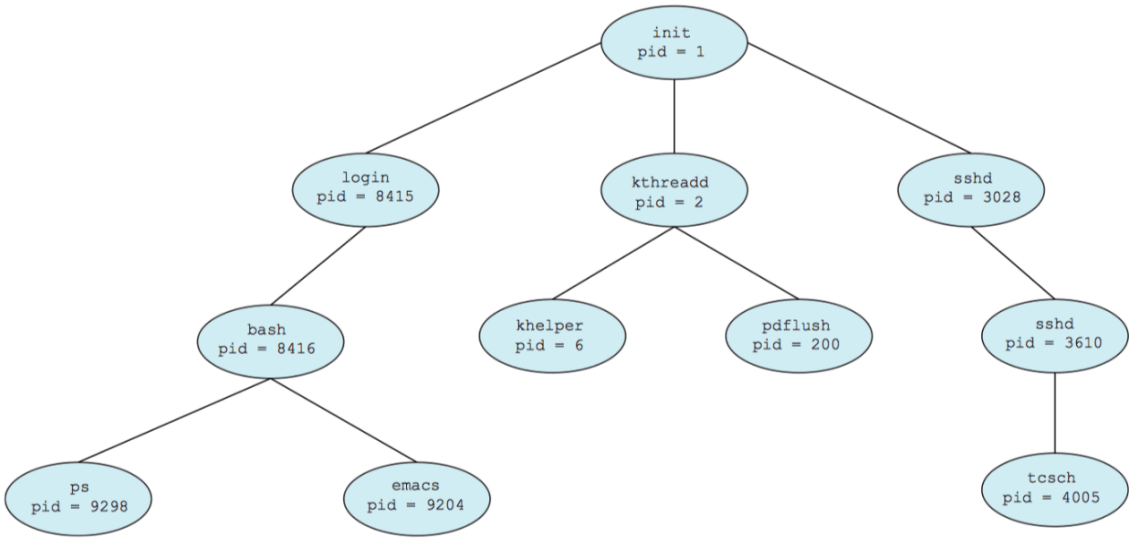


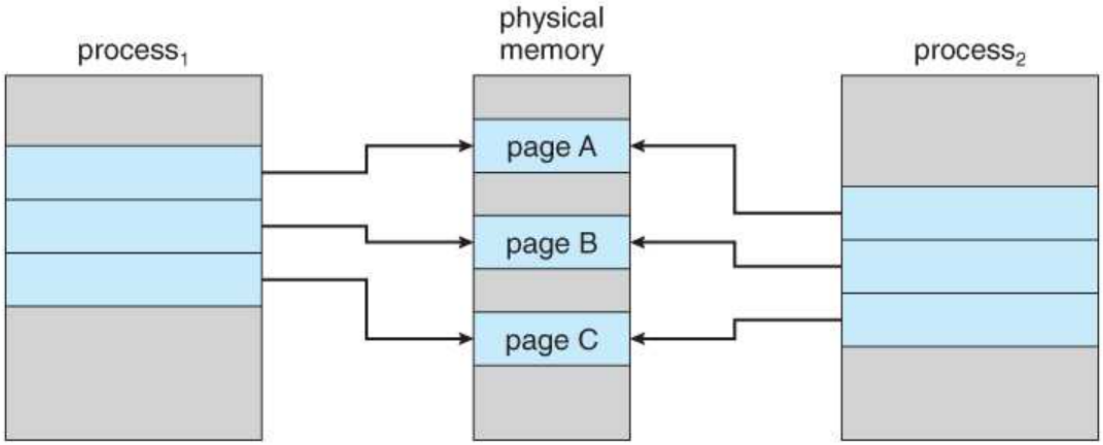
Figure 3.8 A tree of processes on a typical Linux system.

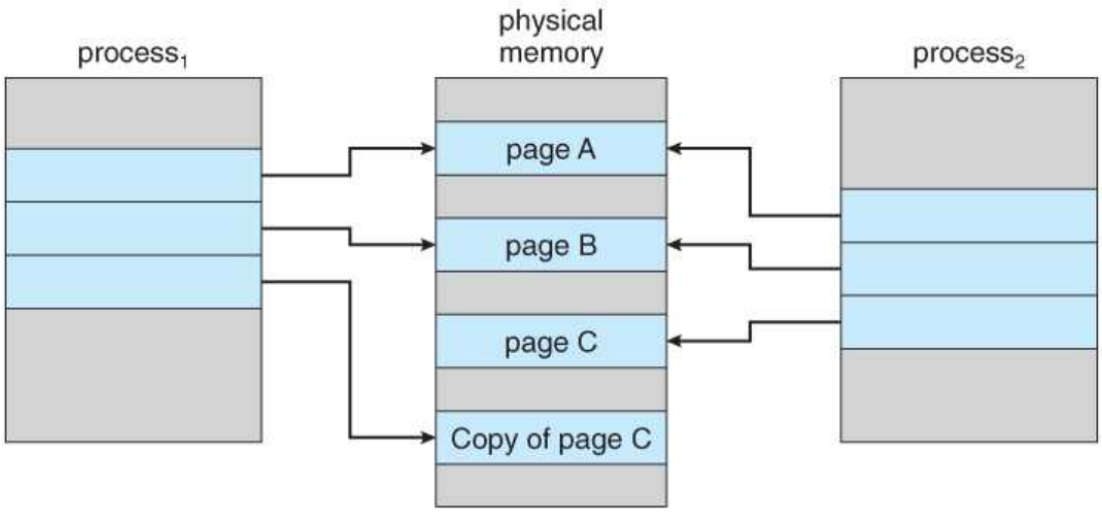
자원을 공유하는 형태

1. 부모와 자식이 모든 자원을 공유하는 모델
2. 일부를 공유하는 모델
3. 전혀 공유하지 않는 모델

→ 부모 프로세스와 자식 프로세스는 서로 독립적으로 존재하며 CPU 제어권을 두고 경쟁하는 관계
→ 자원을 공유하지 않는 경우가 일반적

- 하지만 리눅스를 비롯한 일부 모델에서는 부모 자식 프로세스가 자원을 공유하다가
- 부모 또는 자식 프로세스에서 변경사항이 생겼을 때 해당 부분만 복제해서 물리적 메모리에 할당하는 **Copy-on-Write(COW)** 기법을 사용한다.
- Copy-on-Write는 write가 발생해서 내용이 수정됐을 때 copy를 한다는 의미이다. COW 기법을 사용하면 메모리를 보다 효율적으로 사용할 수 있다.





프로세스의 수행(Execution) 형태

1. 부모와 자식이 공존하며 CPU 제어권을 두고 경쟁하는 모델
2. 자식이 종료(terminate)될 때까지 부모가 기다리는(blocked, wait) 모델

2. 프로세스 종료 (Process Termination)

- 프로세스가 마지막 명령을 수행한 후 운영체제에게 이를 알려준다.

  : `exit` system call (자발적 종료)

  - 프로세스의 세상에서는 항상 자식 프로세스가 먼저 종료되며, 자식 프로세스는 종료 시점에 `wait` system call을 통해 부모에게 output data를 전달한다.
  - 프로세스의 각종 자원들이 운영체제에게 반납된다.
- 다음의 3가지 경우에는 부모 프로세스가 자식 프로세스의 수행을 종료시킨다.

  : `abort` system call (비자발적 강제 종료)

1. 자식이 할당 자원의 한계치를 넘어서는 경우
2. 자식에게 할당된 태스크가 더 이상 필요하지 않은 경우
3. 부모 프로세스가 먼저 종료(exit)되는 경우

> 운영체제는 부모 프로세스가 종료되는 경우 자식 프로세스가 수행되도록 두지 않는다.자식이 생성한 모든 자손 프로세스들을 종료시킨다. 트리의 가장 깊은 레벨에 위치한 자손부터 단계적으로 종료시킨 후 맨 마지막에 부모 프로세스를 종료시킨다.

---

3. 프로세스와 관련된 시스템 콜

(1) fork (2) exec (3) wait (4) exit

---

(1) fork

fork 시스템 콜은 자식 프로세스를 생성할 때 사용된다.

아래 코드는 부모 프로세스가 fork 시스템 콜을 통해 프로세스 생성을 요청하는 내용을 담고 있다.

(2) exec

exec는 기존 코드를 새로운 코드로 덮어쓸 때 사용된다.

exec 시스템 콜과 fork 시스템 콜은 독립적이기 때문에 반드시 fork를 해야만 exec을 할 수 있는건 아니다. 또한 자식 프로세스 뿐만 아니라 부모 프로세스 역시 exec을 사용할수 있다.

중요한건 exec을 한 번 실행하면 되돌릴 수 없다는 점이다. exec을 실행해서 코드를 새롭게 덮어쓰게 되면 새로운 코드의 실행이 끝난 뒤 덮어쓰기 전의 기존 코드로 돌아오는게 아니라 프로세스가 아예 종료된다. 따라서 기존 코드의 `execlp()`(= exec 실행 함수) 이후의 코드는 실행 할 수 없다.

(3) wait

프로세스는 항상 자식 프로세스가 부모 프로세스보다 먼저 종료된다고 했었다. **wait 시스템 콜은 자식 프로세스가 종료될 때 까지 기다릴 때 사용**된다.

(4) exit

exit 시스템 콜은 프로세스를 종료할 때 사용되며 아래 소스 코드와 같이 프로그램 내에서 명시적으로 호출할 수도 있고, 컴파일러가 알아서 필요한 시점에 (e.g. 코드 실행 종료 시점)에 호출하기도 한다.

---

4. 프로세스 간 협력

원칙적으로 프로세스는 각자의 주소공간을 가지고 수행되므로 하나의 프로세스가 다른 프로세스의 수행에 영향을 미치지 못하지만(= 독립적 프로세스, Independent process)

협력 메커니즘을 통해 하나의 프로세스가 다른 프로세스의 수행에 영향을 미칠 수 있다.(= 협력 프로세스, Cooperating process)

프로세스 간 협력 메커니즘 (IPC: Interprocess Communication)

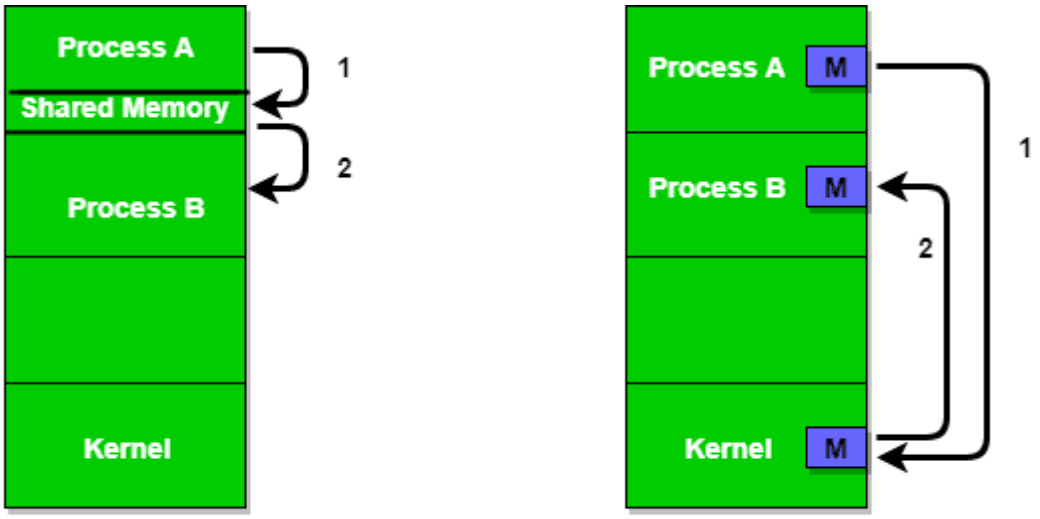


Figure 1 - Shared Memory and Message Passing

---

1. `shared memory`
   - 서로 다른 프로세스 간에 일부 주소공간을 공유
   - shared memory를 mapping할때만 시스템콜 날림 → **faster**

   스레드는 사실상 하나의 프로세스이므로 프로세스 간 협력으로 보기는 어렵지만, 동일한 프로세스를 구성하는 스레드들 간에는 주소공간을 공유하므로 협력이 가능함

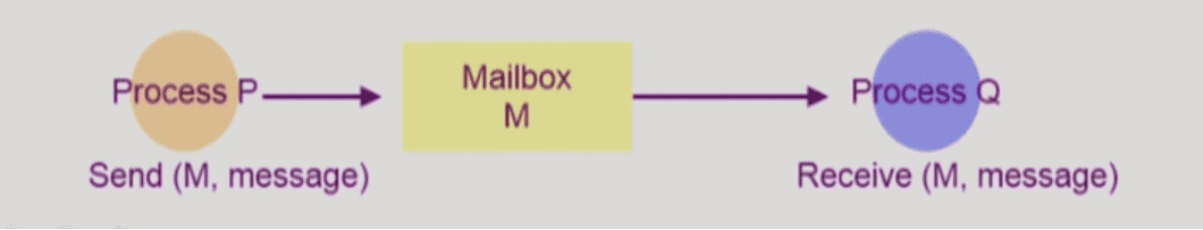
2. `message passing`
   - 프로세스 사이에 공유 변수(shared variable)을 일체 사용하지 않고 커널을 통해 통신하는 방법
   - 메시지를 보낼 때마다 시스템콜을 날림 → **slower**
   - 통신하려는 프로세스의 이름을 명시적으로 표시하는지 여부에 따라 direct / indirect로 구분

Direct Communication

– 통신하려는 프로세스의 이름을 명시적으로 표시

Indirect Communication

– mailbox(또는 port)를 통해 메시지를 간접적으로 전달



메시지를 메일박스에 넣어놓기만 하고 어떤 프로세스가 받을지는 명시하지 않음 (아무나 받아라)