3️⃣

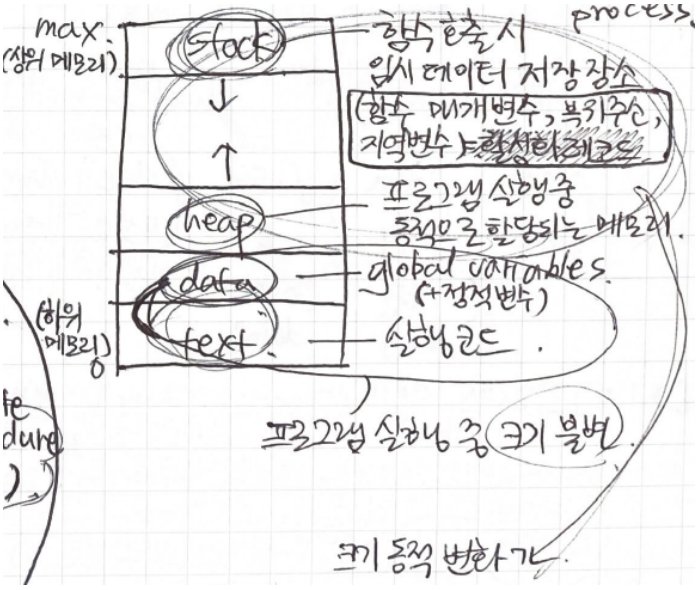
Process

Created time	@April 30, 2024 4:02 PM
Subject	운영체제

3.1 프로세스 개념

3.1.1 프로세스

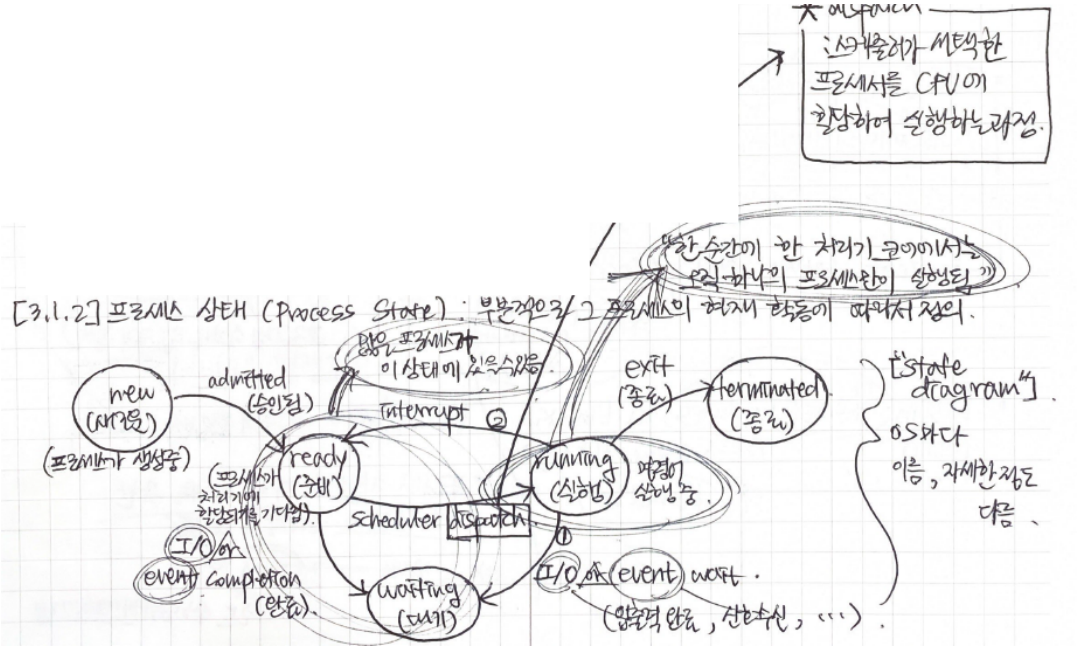
- 프로세스는 실행 중인 프로그램을 뜻합니다.
- 프로세스 메모리는 프로그램 실행 중에 사용되는 코드, 데이터, 실행 환경을 저장하는 공간인데, 배치는 그림과 같습니다.



함수 호출 시 활성화 레코드가 스택에 푸시된다.

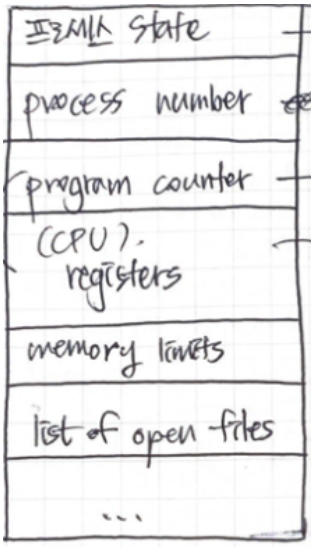
3.1.2 프로세스 상태 (Process State)

- 프로세스는 실행되면서 그 상태가 변하는데, 그림에서 다양한 상태를 확인할 수 있습니다.



3.1.3 Process Control Block = PCB

- 각 프로세스는 OS에서 PCB로 표현되는데, 이 PCB에는 특정 process에 관련된 여러 정보가 수록되어 있습니다.



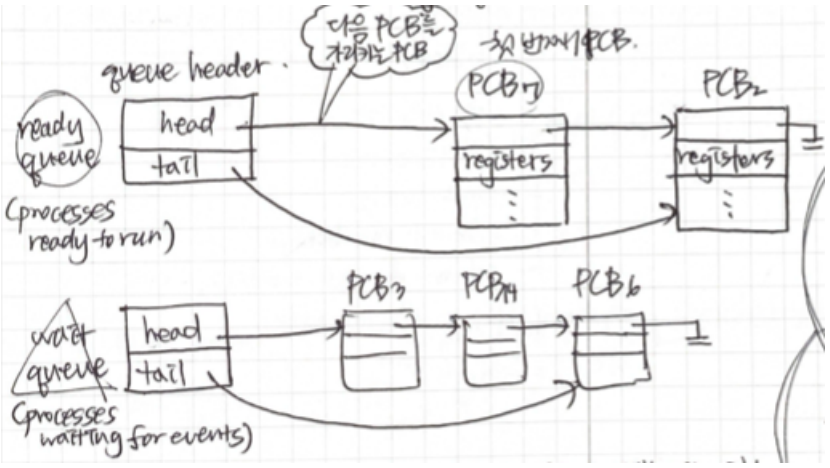
- 맨 위쪽의 프로세스 state는 앞서 언급했던 프로세스의 여러가지 상태들 중 현재 상태가 기록됩니다.
- Process counter에는 이 프로세스가 다음에 실행할 명령어의 주소가 기록됩니다.
- registers는 나중에 프로세스가 다시 스케줄될 때 계속 올바르게 실행되도록 하기 위해서 interrupt 발생 시 저장되는 부분입니다.
- 이외에도 CPU-scheduling information, memory-management information, ... 등 프로세스에 관련된 다양한 정보가 PCB에 저장되게 됩니다.

3.2 프로세스 스케줄링

각 단일 CPU 코어는 한 번에 하나의 프로세스를 실행할 수 있기에, 이러한 경우 '프로세스 스케줄러'는 실행 가능한 여러 프로세스 중에서 하나를 선택합니다.

만약 다중 코어 시스템일 경우, 한 번에 여러 프로세스가 실행될 수 있으며, 코어보다 많은 프로세스가 있는 경우 초과된 프로세스는 코어가 사용가능해질 때까지 기다려야 합니다.

3.2.1 스케줄링 큐(Queue)

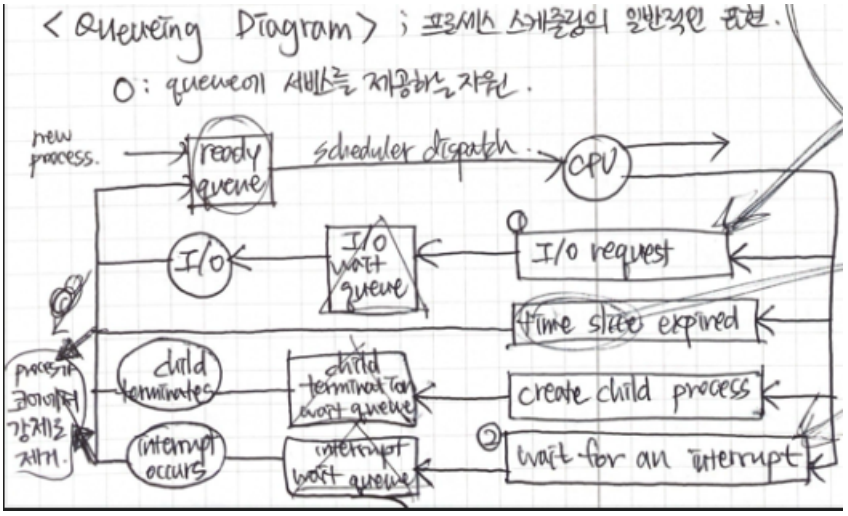


- 프로세스가 시스템에 들어가면 준비 큐에 들어가서 준비 상태가 되어 CPU 코어에서 실행되기를 기다립니다. 이 큐는 일반적으로 그림의 위쪽과 같이 연결 리스트로 저장됩니다.

- I/O 완료와 같은 특정 이벤트가 발생하기를 기다리는 프로세스는 아래와 같이 '대기 큐'에 삽입되게 됩니다.

[프로세스 스케줄링을 나타내는 큐잉 다이어그램]

새 프로세스는 처음에 준비 큐에 놓이고, 프로세스에 CPU 코어가 할당되고 실행 상태가 되면, 여러 이벤트 중 하나가 발생할 수 있음을 나타내는 다이어그램입니다. 여기서 원은 해당 큐에 서비스를 제공하는 자원을 나타냅니다.



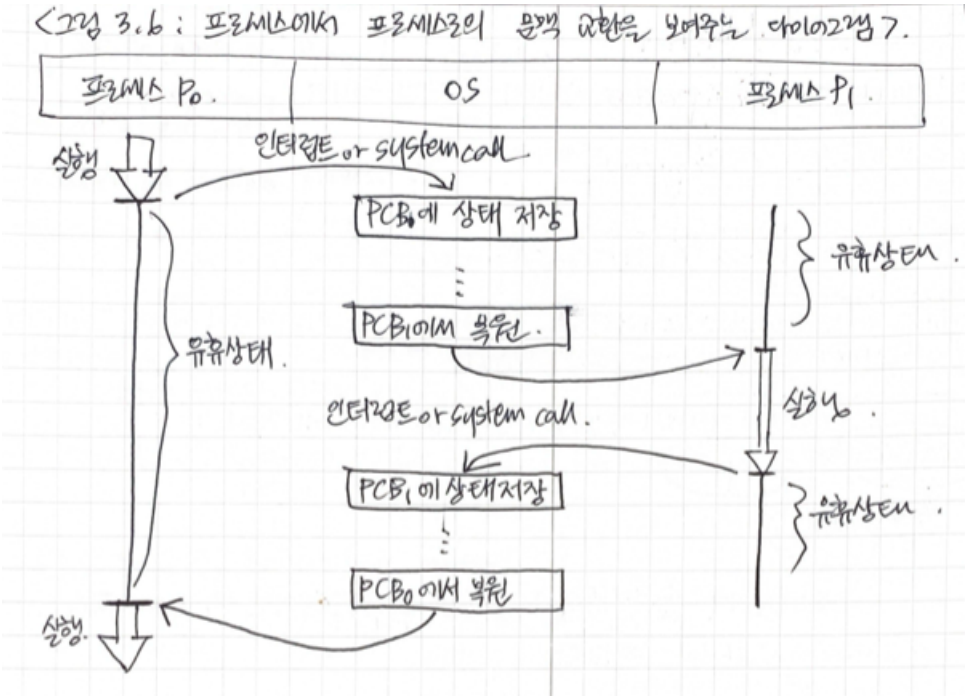
3.2.2 CPU Scheduling

준비 큐에 있는 프로세스 중에서 선택된 하나의 프로세스에 CPU 코어를 할당하는 과정을 'CPU Scheduling'이라고 합니다.

3.2.3 문맥 교환 (Context Switch)

CPU 코어를 다른 프로세스로 교환하려면 이전의 프로세스의 상태를 보관하고 새로운 프로세스의 보관된 상태를 복구하는 작업이 필요한데, 이 작업을 '문맥 교환'이라고 합니다.

문맥 교환이 일어나면, 커널은 과거 프로세스의 문맥을 PCB에 저장하고, 실행이 스케줄된 새로운 프로세스의 저장된 문맥을 복구하게 됩니다.

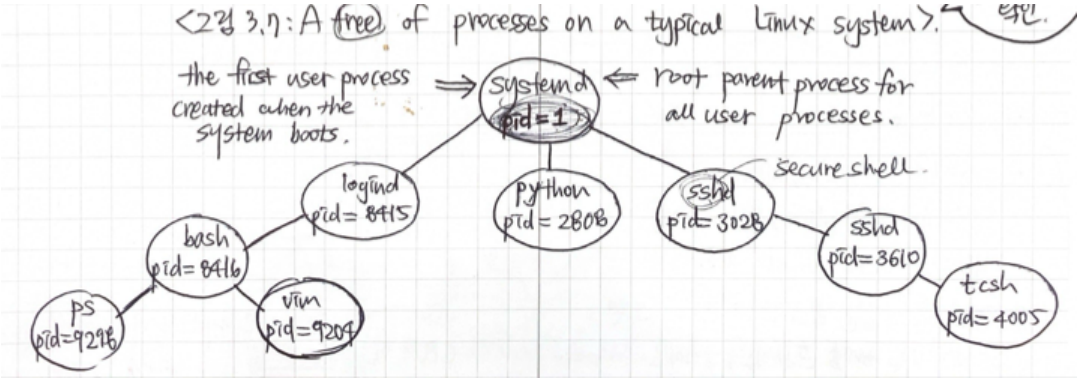


3.3 프로세스에 대한 연산

대부분 시스템 내의 프로세스들은 병행 실행될 수 있으며, 반드시 동적으로 생성되고 제거되어야합니다.

3.3.1 프로세스 생성

- 생성하는 프로세스를 부모 프로세스라고 부르고, 새로운 프로세스는 자식 프로세스라고 부릅니다.
- 이 새로운 프로세스들은 각각 다시 다른 프로세스들을 생성할 수 있으며, 그 결과 그림과 같은 프로세스의 트리를 형성합니다.
- 프로세스를 구분하는 데에는 '프로세스 식별자(pid)'가 쓰이게 됩니다.



- UNIX 운영체제에서는 새로운 프로세스가 fork() 시스템 콜로 생성됩니다. 새로운 프로세스는 원래 프로세스의 주소 공간의 복사본으로 구성되게 됩니다.

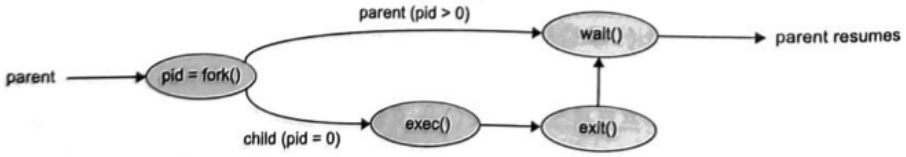


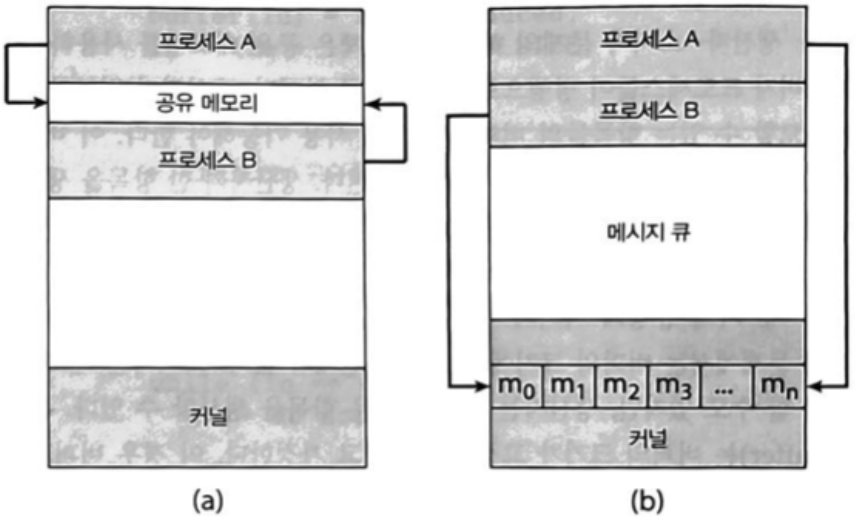
그림 3.9 `fork()` 시스템 콜을 사용한 프로세스 생성

3.3.2 프로세스 종료

- 프로세스는 exit() 시스템 콜을 사용하여 운영체제에 자신의 삭제를 요청하면 종료됩니다.
- 부모 프로세스는 자식이 자신에게 할당된 자원을 초과하여 사용하는 경우, 자식에게 할당된 테스크가 더 이상 필요없는 경우 등 다양한 경우에 자식 중 하나의 실행을 종료할 수 있습니다.

3.4 프로세스 간 통신

- 프로세스가 시스템에서 실행 중인 다른 프로세스들에 영향을 받는 경우에는 '협력적인 프로세스'라고 합니다.
- 협력적 프로세스들은 데이터를 서로 주고받을 수 있는 '프로세스간 통신(interprocess communication, IPC)' 기법이 필요합니다.
- IPC는 두 가지 방식이 있는데, 공유 메모리와 메시지 전달입니다.



3.5 공유 메모리 시스템에서의 프로세스 간 통신

공유 메모리 시스템에서는 공유되는 메로리 영역을 구축하는 단계에서만 시스템 콜이 필요하고, 구축된 후에 발생하는 모든 접근은 일반적인 메모리 접근으로 취급되어 커널의 도움이 필요 없다는 것이 장점입니다.

메시지 전달 시스템은 항상 시스템 콜을 사용하므로 커널 간섭 등의 부가적인 시간 소비 작업이 필요하기에 공유 메모리 모델이 더 빠르다는 장점이 있다는 것입니다.

3.6 메시지 전달 시스템에서의 프로세스 간 통신

메시지 전달 모델은 충돌을 회피할 필요가 없기 때문에 적은 양의 데이터를 교환하는 데 유용합니다.

그리고 메시지 전달은, 분산 시스템에서 공유 메모리보다 구현이 쉽다는 것이 장점입니다.

3.7 IPC 시스템의 사례

실제 다양한 IPC 시스템이 있는데, 그 중에서 '파이프'에 대해 설명하겠습니다.

3.7.4 파이프(Pipes)

파이프는 초기 UNIX 시스템에서 제공하는 IPC 기법 중 하나이며, 두 프로세스가 통신할 수 있게 하는 전달자 역할입니다.

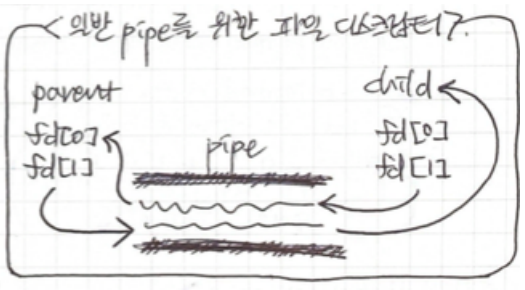
파이프를 구현하기 위해서는 '단방향 통신인지, 양방향 통신인지', '부모-자식과 같은 특정 관계가 존재해야만 하는지' 등 다양한 문제를 고려해야합니다.

UNIX에서 사용되는 파이프에는 두 가지 유형이 있습니다.

3.7.4.1 일반 파이프

첫 번째는 '일반 파이프' 입니다. 일반 파이프는 한쪽으로만 데이터를 전송할 수 있으며 오직 단방향 통신만을 가능하게 합니다.

그리고, 일반 파이프는 파이프를 생성한 프로세스 이외에는 접근할 수 없어, 일반적으로 프로세스가 파이프를 생성하고 fork()로 생성한 자식 프로세스와 통신하기 위해 사용됩니다.



3.7.4.2 지명 파이프

지명 파이프는 양방향으로 통신이 가능하며, 부모-자식 관계가 필요하지 않다는 것이 장점입니다.

그래서 지명 파이프가 구축되면 여러 프로세스들이 이 지명 파이프를 사용하여 통신할 수 있게 됩니다.

3.8 클라이언트 서버 환경에서 통신

앞에서 언급한 통신 방법들은 프로세스 간의 통신입니다.

클라이언트와 서버가 통신하는 방법은 두 가지가 있는데, 소켓과 원격 프로시저 호출입니다.

3.8.1 소켓

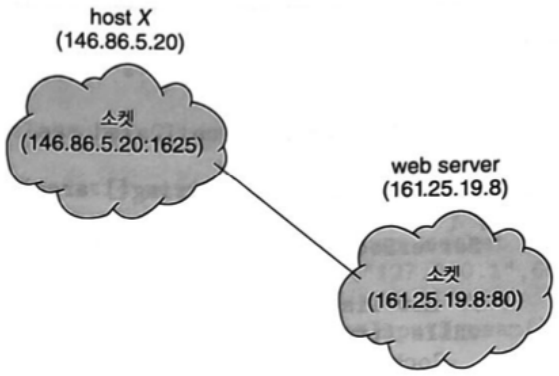


그림 3.26 소켓을 사용한 통신

소켓은 통신의 극점을 뜻하며, 두 프로세스가 네트워크 상에서 통신하려면 총 두 개의 소켓이 필요하게 됩니다.

서버는 지정된 포트에 클라이언트 요청 메시지가 도착하기를 기다리게 되고, 요청이 수신되면 서버는 클라이언트 소켓으로부터 연결 요청을 수락함으로써 연결이 완성됩니다.

3.8.2 원격 프로시저 호출 (Remote Procedure Calls = RPC)

IPC와 유사하며, IPC 기반 위에 만들어지지만, IPC와는 달리 RPC 통신에서 전달되는 메시지는 구조화되어있고, 따라서 데이터의 패킷 수준을 넘어서게 되는 것이 특징입니다.