10.Process & Thread

10.Process & Thread

10.1. Instruction

- Process:
  - 실행중인 프로그램
  - 프로그램은 보조기억장치에 저장되어 있으며, 이를 메모리에 적재하고 실행하는 순간 프로세스가 된다.
  - 이 과정을 '프로세스를 생성한다'고 표현한다.
- Foreground process
  - 사용자가 보이는 곳에서 실행되는 프로세스
- Background process
  - 사용자가 볼 수 없는 곳에서 실행되는 프로세스
  - 사용자와 상호작용하는 프로세스와 상호작용하지 않는 프로세스(daemon in unix, service in window)로 나뉜다.

10.1.1. PCB

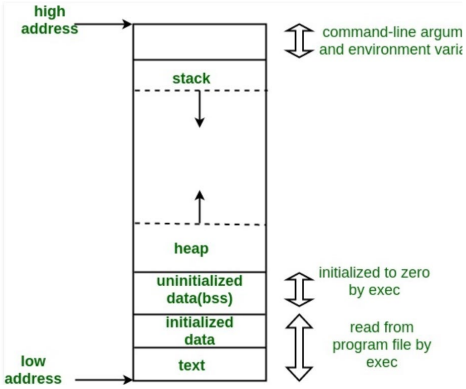
- Process Control Block
- 한정된 CPU 자원을 효율적으로 사용하기 위해 이용.[1]
- 프로세스와 관련된 정보를 저장하는 자료구조
- 프로세스를 식별하는 정보를 저장
- 커널 영역에서 생성된다.
- 프로세스 생성 시에 만들어지고 실행이 끝나면 폐기. 즉, 프로세스 생성 = PCB 생성이며 프로세스 종료 = PCB 폐기다.

- PCB에 저장되는 정보는 다음과 같다.
  - PID: Process ID. 프로세스 식별을 위한 고유 번호.
  - Registers:
    - 복원해야 할 중간값.
    - 이전에 사용했던 프로그램 카운터 등이 저장됨
  - Process state:
    - create/ready/running/waiting/terminated 등을 저장
    - 즉, CPU를 사용하기 위해 대기중인지, 사용중인지 등
  - Priority:
    - CPU scheduling 정보
    - 언제, 어떤 순서로 CPU를 할당받을지에 대한 정보
  - Memory information:
    - 프로세스가 저장된 주소 정보
    - 베이스 레지스터, 한계 레지스터 값, 페이지 테이블 정보 등
  - I/O information
    - 실행 과정에서 특정 입출력 장치, 파일을 사용한 기록.

10.1.2. Context Switch

- Context:
  - 프로세스 수행을 재개하기 위해 기억해야 할 정보
  - 프로세스의 실행 중간정보, 즉 PCB에 저장된 프로그램 카운터, 메모리 정보 등을 말한다.
- 문맥 교환
  - 실행 중 인터럽트 처럼 프로세스를 종료하고 다른 프로세스를 실행해야하는 시점에 이루어지는 과정
  - 이전 프로세스의 실행 정보(문맥)를 PCB에 백업하고 다음에 실행할 프로세스의 문맥을 복원한다.

10.1.3. 프로세스 메모리 영역

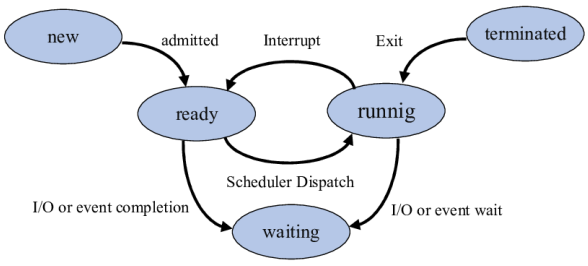


- 프로세스의 사용자 영역에는 크게 코드, 데이터, 힙, 스택 영역으로 구분된다.
- code segment:
  - text segment라고도 부르며, 실행할 수 있는 코드, 즉 기계어로 이루어진 명령어가 저장된다.
  - 수정이 불가능한 read-only 공간이다.
  - 정적 할당 영역.
- data segment:
  - 프로그램 실행 동안 유지할 데이터가 저장되는 공간
  - 대표적으로는 global variable이 있다.
  - 정적 할당 영역.
- heap segment:
  - 동적 할당 영역
  - 프로그래머가 직접 할당할 수 있는 저장 공간
  - 적절하게 반환하지 않는다면 메모리 누수(memory leak)가 발생할 수 있다.
- stack segment:
  - 동적 할당 영역
  - 데이터를 일시적으로 저장하는 공간.
  - 잠시 사용하는 데이터로, 매개변수, 지역변수가 대표적이다.

10.2. 프로세스 상태와 계층 구조

- 운영체제는 프로세스의 상태를 PCB에 기록해, 동시에 실행되는 다수의 프로세스를 계층적으로 관리한다.

10.2.1. Process state



- New: 프로세스 생성. 메모리에 적재되어 PCB를 할당받은 상태.

- Ready: CPU를 할당받을 수 있는 상태. 준비에서 실행단계로의 전환을 dispatch라 한다.
- Running: CPU를 할당받아 실행중인 상태. 할당된 일정 시간 동안만 사용할 수 있으며, timer intterrupt가 발생하면 다시 준비 상태로 돌아가고, I/O 작업을 기다려야 하면 대기상태로 돌아간다.
- Blocked: 입출력 작업을 요청 후 장치의 작업이 마칠 때까지 기다리는 상태. 작업이 완료되면 프로세스는 준비 상태로 전환된다. 입출력 작업 외에도 특정 이벤트를 기다릴 때도 대기 상태로 전환된다.
- Terminated: 종료 상태. 이때 OS는 PCB와 메모리를 정리한다.

10.2.2. 계층 구조

- 프로세스는 실행 도중 다른 프로세스를 생성할 수 있으며, 생성한 프로세스를 parent process, 생성된 프로세스를 child process라 한다.
- 부모와 자식은 서로 다른 PID를 가지며, 일부 운영체제에서는 자식 프로세스의 PCB에 부모 프로세스의 PID(PPID: Parent PID)를 기록하기도 한다.
- 자식 프로세스도 자신의 자식 프로세스를 생성할 수 있기에, 이 과정을 반복하므로써 계층 구조가 형성되며, 이 구조 속에서 OS는 프로세스를 관리하게 된다.

10.2.3. 프로세스 생성 기법

- `fork`:
  - 부모 프로세스는 `fork`를 통해 자신의 복사본을 자식 프로세스로 생성한다.
  - 메모리 내의 내용, 열린 파일 목록 등의 자원이 복사된다.
- `exec`:
  - 만들어진 복사본(자식 프로세스)은 `exec`로 자신의 메모리 공간을 다른 프로그램으로 교체한다.
  - 코드 및 데이터 영역의 내용이 실행할 프로그램의 내용으로 바뀌고 나머지 영역은 초기화된다.
  - 자식 프로세스 중 이 시스템 콜을 사용하지 않는 경우는 부모 프로세스와 같은 코드를 병행하여 실행하는 프로세스다.
- Network에서 socket을 관리할 때 사용되기도 한다. [2]

10.3. Thread

- 프로세스를 구성하는 실행의 흐름 단위

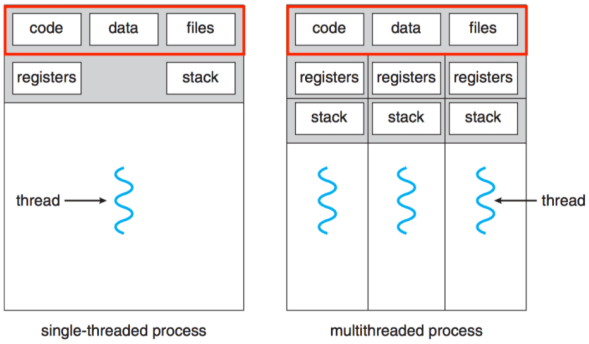
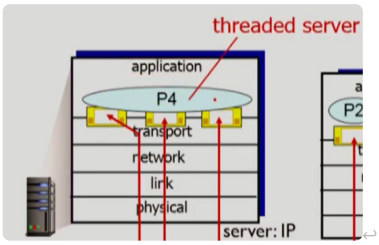


Figure 4.1 Single-threaded and multithreaded processes.

10.3.1. 프로세스와 스레드

- 단일 스레드 프로세스
  - 실행의 흐름 단위가 하나인 프로세스
  - 하나의 프로세스에 하나의 스레드, 즉 한 번에 하나의 명령어만 실행한다.
- 멀티 스레드 프로세스
  - 실행의 흐름 단위가 여러 개인 프로세스
  - 프로세스에서 한 번에 여러 명령어를 실행할 수 있다.
  - 각각의 스레드는 각기 다른 스레드 ID, 레지스터 값, 스택 등 최소한의 정보만 유지한 채 **프로세스의 자원을 공유**[3] [4] 하며 실행된다.
- 프로세스가 실행되는 프로그램이라면, 스레드는 프로세스를 구성하는 실행의 흐름 단위.
- 최근의 운영체제에서는 처리할 작업을 프로세스 단위가 아닌 스레드 단위로 전달하기도 한다.
-

1. OS는 timer interrupt(하드웨어 인터럽트로, 클럭 신호를 발생시키는 장치에 의해 주기적으로 발생)에 따라 프로세스에 CPU를 넘겨주는 등 프로세스의 실행 순서와 자원의 분배를 담당한다.↩
2. server app은 여러 process를 두고 있다. python에서 client contact가 들어올 때 socket만 새로 만들어 줬다. 반면, 실제 C로 app을 만들 때는 socket 뿐 아니라 socket으로 data를 주고받을 process도 `fork()`한다. app의 자식 process를 만들어 해당 socket으로 데이터를 주고받는 일만 담당하게 만든다. 다만 process가 늘어나면 overhead가 커지기 때문에 thread로 서로 다른 socket을 관리하기도 한다.



↩

3. 이 때문에 단일 스레드 프로세스를 여러 개 돌리는 것 보다 메모리 효율이 좋다. 단일 스레드 프로세스는 각기 다른 코드, 데이터, 힙, 파일 등의 영역이 필요하기 때문에 메모리 낭비가 심하다. 즉, `fork()` 시 부모와 자식 프로세스는 서로 다른 메모리 영역을 차지하는 반면, 스레드는 동일한 메모리 영역 내에서 명령어를 수행하고 이런 특징 때문에 협력과 통신에 유리하지만 안정성 문제를 가지고 있다. ↩
4. 프로세스 간 통신도 물론 가능하다. 이를 IPC(Inter Process Communication)이라 하며, 동일한 파일 속 데이터를 주고 받거나, 공유 메모리(shared memory)를 통해 데이터를 주고받을 수 있다. 이 외에도 소켓, 파이프 등으로 통신 가능하다.↩