Process & Thread

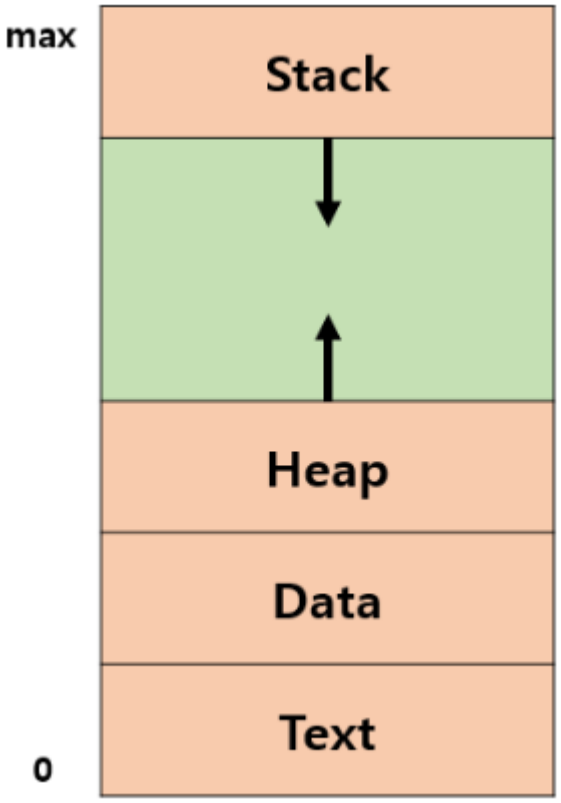
태그	OS
주차	4주차

📌 목차

1. Process

- 실행중인 프로그램
- 운영체제에 의해 관리됨
- 독립적으로 실행되고 자원을 할당 받을 수 있는 단위

1-1. 구조



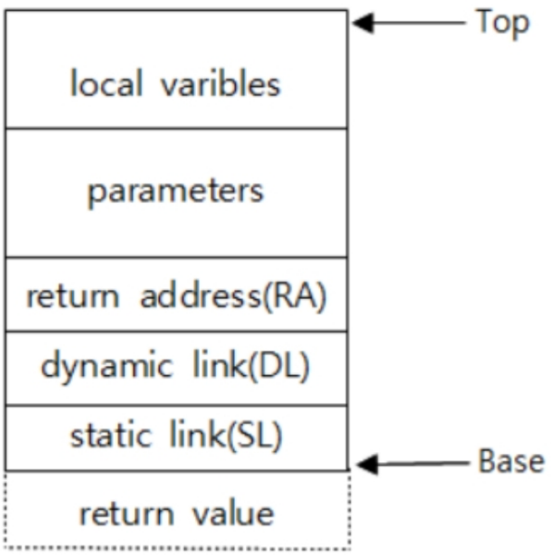
1. Text 영역(Text Segment)
    - 코드가 저장되는 공간
    - 코드는 컴파일되어 0과 1로 변환된 기계어가 저장됨
2. Data 영역(Data Segment)
    - 코드에서 선언된 전역 변수, 정적 변수, 상수 등을 저장

      2-1. BSS Section
        - 초기화되지 않은 데이터
        - 초기화되지 않은 전역변수, 상수 저장

      2-2. Data Section
        - 초기화된 데이터
        - 초기화된 전역변수 및 static 변수 저장
    - 초기화된 변수와 초기화되지 않은 변수들이 나눠서 저장됨
    - 데이터 영역은 프로그램의 시작 시 초기화되며, 프로세스가 종료될 때 까지 유지
3. Heap

- 코드에서 동적으로 생성되는 데이터 구조나 객체 저장
- 데이터가 추가됨에 따라 유동적으로 공간을 늘릴 수 있음
- 프로세스 실행 중 동적으로 메모리를 할당받고 해제하는데 사용
- 프로세스 주소 공간의 나머지 영역에 위치하며, 크기는 동적으로 확장 가능

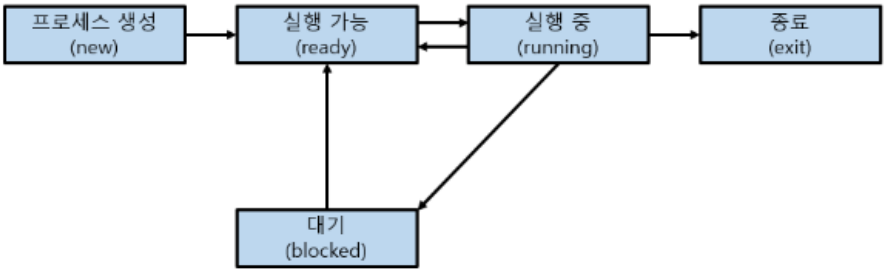
4. Stack
    - 함수 호출 시 stack 공간 생성, 함수 종료시 제거
    - 지역변수, 매개변수, return 주소 등을 저장

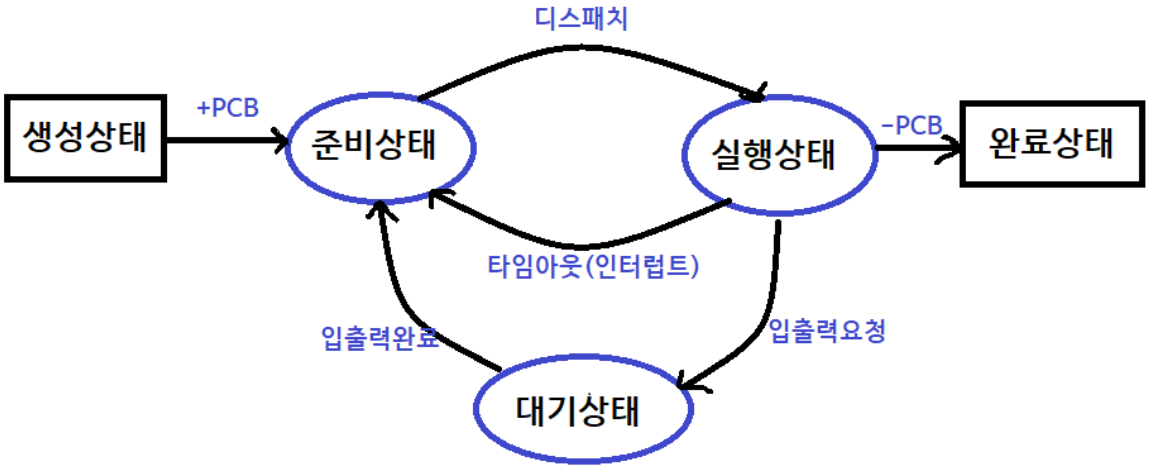


스택의 구조

- Base/Top: 해당 함수의 스택에서의 시작 주소와 끝 주소. Top 주소는 SP(stack pointer) 레지스터에 저장됨
- return address(RA): 스택 종료 후 return될 함수의 주소 (PC값)
- dynamic link(DL): 이전에 호출된 함수의 Base를 가리킴
- static link(SL): 전역변수가 저장된 스택 주소를 가리키며, 전역변수 사용 시 참조

1-2. 상태





1. 프로세스 생성(new)
    - 프로세스가 생성된 상태
    - 실행을 위한 자원을 할당받지 못한 상태
2. 실행 가능(Ready)
    - 실행을 기다리는 상태
    - PCB에 올라감
    - 필요한 자원을 모두 할당받았지만 CPU를 할당받지 못한 상태
    - CPU를 할당받기 위해 스케줄링 대기열(Queue)에 들어감
3. 실행 상태(Running)
    - 코드를 실제로 실행하는 상태
    - 작업을 처리하고 결과를 만들어 냄
4. 대기(Blocked)

- 프로세스 처리 중 작업 시간이 초과되거나 자원 사용을 위해 대기해야 하는 이벤트가 발생해 프로세스가 잠시 멈춘 상태
- CPU를 사용하지 않음
- 특정 자원을 사용할 수 있을 때 까지 대기열(Ready Queue)로 들어가며, 처리 가능 상태가 되면 실행(Running)상태로 변경

5. 종료 (Terminated, Exit)
    - 실행이 완료되어 종료된 상태
    - PCB에서 삭제
    - 할당된 자원이 해제되고, 메모리 공간은 OS에 반환

1-3. 프로세서 처리를 위한 레지스터

1) Program Counter (PC)

- 실행 할 명령어의 주소를 가리키는 레지스터
- 명령어를 순차적으로 실행하며 PC값을 증가시켜 다음 명령어를 찾음
- 프로그램의 흐름을 제어하는 역할. 명령을 실행하다 분기나 점프 명령어를 만나면 PC값을 분기된 주소로 변경하여 해당 명령어 실행
- 프로세스가 중단되거나 인터럽트가 발생했을 때 현재 실행 중이던 명령어의 주소를 저장하고, 이후 다시 프로세스가 재실행될 때 주소를 찾아가 재실행 시킬 수 있는 역할을 함

2) Stack Pointer (SP)

- 현재 실행중인 프로세스의 Stack 최상단을 가리키는 레지스터
- 가장 최근에 호출한 함수의 위치
- 스택 프레임(Stack Frame, 스택 공간)의 시작주소를 가리키며, 스택에 데이터를 저장하거나 불러올 때 사용
- 함수 호출 시 SP가 감소하여 새로운 스택 프레임을 생성하고, 반환되면 SP가 증가하여 이전 스택 프레임으로 되돌아 감
- SP는 프로세스의 스택 영역을 관리하고, 스택 오버플로우(Stack Overflow)와 같은 문제를 방지하기 위해 제한된 메모리 영역을 사용하는 등의 역할을 수행

1-4. Process Control Block (PCB)

- 운영체제가 프로세스를 관리하기 위해 사용하는 데이터 구조
- Kernel의 데이터 영역에서 각 프로세스의 상태, CPU 사용 정보, 메모리 사용 정보 등의 각종 자원 관리를 위한 공간
- 저장되는 정보
  1. **프로세스 상태(Process State)**
  2. **프로그램 카운터(Program Counter, PC)**
  3. **레지스터 (Registers)**
     - 프로세스가 사용중인 레지스터 값 저장
     - 프로세스가 일시 중단되고 다시 실행될 때 레지스터 값들을 복원하는 데 사용
  4. **스케줄링 정보 (Scheduling Info.)**
     - 프로세스의 우선순위, 할당 된 CPU 시간, 스케줄링 알고리즘 등 스케줄링에 필요한 정보
  5. **메모리 관리 정보 (Memory Management Info.)**
     - 프로세스가 사용하는 메모리 공간의 주소 범위, 페이지 테이블, 메모리 할당 등 메모리 관리에 필요한 정보
  6. **입출력 상태 (I/O state)**
     - 프로세스가 사용중인 입출력 장치 정보
     - 어떤 요청을 보냈는지, 어떤 파일을 열었는지 등의 정보

2. Thread

- CPU에 작업 요청을 하는 실행 단위

2-1. Process와 Thread의 차이

- 프로세스
  - 운영체제 입장에서의 작업의 단위
  - 작업이 다른 프로세스에 영향을 미치지 않음 (독립성)
- 쓰레드
  - CPU 입장에서의 작업의 단위
  - 작업이 다른 쓰레드에 영향을 미침

2-2. Java에서의 Thread

- JVM이 프로그램 시작점인 main() 메소드를 찾아 메인 스레드 생성
- 메인 스레드의 run() 메소드를 통해 해당 main() 메소드 실행
  - 메인 스레드는 싱글 스레드
- JVM 시작 시 데몬 스레드도 시작
  - 데몬 스레드
    - 다른 스레드에 도움을 주는 스레드로, GC(가비지 컬렉터)가 있음
    - 메인스레드 종료시 같이 종료
- 싱글 스레드 애플리케이션에서는 메인 스레드가 종료되면 프로세스도 종료됨

2-3. Thread를 사용하는 이유

- 메모리 절약을 위함
  - 프로세스는 실행시 평균적으로 32~64MB 물리메모리 점유
  - 그러나 쓰레드는 1MB 이내의 메모리만 점유 ⇒ 경량 프로세스
- 프로세스 Context Switching에 비해 오버헤드 절감
  - 스케줄러가 기존 실행 프로세스를 우선순위 때문에 미루고 새 프로세스로 교체해야 할 때 프로세스 상태 값을 교체하는 작업
  - 쓰레드는 생성될 때 마다 스택이 새롭게 생성될 뿐, 힙, 데이터는 공유
- 작업들 간 통신 비용 절감
  - IPC에 비해 쓰레드는 데이터를 공유하므로 쓰레드간 통신 비용이 적음

2-4. Multi Thread

- 하나의 프로세스 내에서 둘 이상의 스레드가 동시에 작업을 수행하는 것

1) Context Switching

- 컴퓨터에서 동시에 처리할 수 있는 최대 작업 수는 CPU의 코어 수와 같음
- CPU의 코어 수보다 많은 스레드가 실행되면 각 코어가 정해진 시간동안 여러 작업을 번갈아가며 수행
- 이 때 Context Switching 발생

    - 현재까지의 작업 상태나 다음 작업에 필요한 각종 데이터를 저장하고 읽어오는 작업
- Context Switching의 오버헤드가 커질 수록 멀테 스레딩의 효율 저하
    - 단순한 계산은 싱글 스레드가 더 효율적
    - 따라서 많은 스레드를 실행하는 것이 언제나 좋지는 않음