#cs #study

0) 목차

- 운영체제 전반
- 프로세스와 스레드
- 동기화와 교착 상태

1) 운영체제 전반

운영체제란?

- 메모리에 실행 프로그램을 누가 결정하고 탑재할까? 프로그램이 끝나면 그걸 파악해서 메모리에서 제거하는 게 누구일까? 프로세스와 스레드의 CPU 점유 배분을 누가 하는 걸까? 이처럼 컴퓨터에 주어진 한정된 자원을 배분하고 관리하는 프로그램을 운영체제라 한다.
- 운영체제의 핵심은 커널(kernel)이며 사실상 커널이 하는 일을 설명한다고 봐도 무방하다.

운영체제 역할

- 자원 할당 및 관리
- 프로세스 및 스레드 관리

자원 할당 및 관리

- 자원(resource)이란 프로그램 실행에 필요한 요소들을 의미한다. 이는 하드웨어와 소프트웨어를 전부 포함하는 개념.
- CPU 관리 : CPU 스케쥴링
  - CPU의 한정된 자원을 여러 프로그램이 두루 쓸 수 있도록 배분
- 메모리 관리 : 가상 메모리
  - 신규 프로그램을 메모리에 적재하고 안 쓰는 프로그램을 메모리에서 정리한다
  - 이 때, 낭비되는 용량이 없도록 가상 메모리를 이용하여 메모리를 관리한다
- 파일/디렉터리 관리 : 파일 시스템
  - 보조기억장치는 용량이 거대하며 이를 효율적으로 관리하고 저장하기 위해서는 보관/탐색에 있어 적절한 자료 저장 방식이 필요하다. 이 때, 특정 기준으로 데이터를 묶고 규합하여 계층 구조를 만들어낸 것을 파일 시스템이라 한다.
- 프로세스 및 스레드 관리
  - 프로세스는 실행 중인 프로그램이며 스레드는 프로그램 내부에서 이루어지는 각각의 업무 단위이다. 너무 많은 프로세스와 스레드가 CPU 자원 탈환을 위해 달려들면 CPU가 과열될 것이다. 이러한 일을 막고 공정하게 자원을 분포 및 관리하기 위해 운영체제가 적절한 업무 분배를 수행한다

시스템 콜과 이중모드

- 운영체제는 프로그램이며 프로그램은 메모리에 들어가 있어야 실행할 수 있다. 운영체제는 없으면 안 되는 필수 프로그램이므로 "커널 영역(kernel space)"이라 불리는 별도의 메모리 영역에서 적재한다.
- 사용자가 쓰는 응용 프로그램이 점유하는 메모리 영역은 "사용자 영역(user space)"이라 부른다.
- 응용 프로그램이 운영체제의 기능(자원 쓰기, 프로세스 점유하기 등)을 사용하려면 운영체제의 기능을 불러와야 한다. 이 때, 운영체제 기능을 사용하기 위해 요청하는 방식을 "시스템 콜(system call)"이라 한다.
- 시스템 콜을 통해 프로세스, 스레드, 파일, 디렉터리, 파일 시스템에 접근할 수 있으며 CPU가 시스템 콜을 인지할 수 있도록 인터럽트(interrupt) 신호를 발생시킨다. 이처럼 별도의 명령어를 통해 인터럽트 신호를 발생시키는 것을 소프트웨어 인터럽트라고 한다.
- 전체적인 동작 방식은 다음과 같다
  - 응용프로그램에서 시스템 콜 요청
  - 운영체제에서 CPU로 소프트웨어 인터럽트 신호 발신
  - CPU에서 커널 모드로 전환하여 운영체제 코드 처리
  - 운영체제에서 시스템 콜 결과값을 응용프로그램으로 전송
  - CPU에서 사용자 모드로 전환하여 응용프로그램 동작 처리

Quiz

- 일반적인 사용자 프로세스가 어떻게 CPU를 할당 받아 실행되는가?

2) 프로세스와 스레드

프로세스란?

- 메모리에 적재되어 실행 중인 프로그램 단위
- 포그라운드 프로세스(foreground process) : 사용자가 보는 공간에서 사용자와 상호작용하며 실행되는 프로그램
- 백그라운드 프로세스(background process) : 사용자가 보지 못하는 공간에서 실행되는 프로그램
- 데몬(demon) : 사용자와 별다른 상호작용 없이 주어진 작업만 수행하는 백그라운드 프로세스. 윈도우에서는 서비스(service)라 부른다.

메모리에서 프로세스가 점유하는 영역

- 커널 영역
  - PCB(process control block) : 프로세스 식별 정보를 기록한다. 정보는 크게 PID(process id), 프로세스가 사용한 레지스터 값, 프로세스 현재 상태, CPU 스케쥴링 프로세스 순위, 프로세스가 점유한 메모리 관련 정보, 프로세스가 사용한 파일 입출력장치 관련 정보를 저장한다.
    - PCB 들은 프로세스 테이블(process table) 형태로 관리한다. 프로세스 테이블은 PCB 모음집을 의미한다.
- 사용자 영역

- 코드 영역(code segment) : 실행 가능한 명령어가 저장되는 공간. CPU가 읽고 실행할 명령어가 담겨 있어서 수정이 되면 큰일 날 수 있으므로 읽기 전용(read-only) 공간이다.
- 데이터 영역(data segment) : 프로그램을 실행하는 동안 유지할 필요가 있는 데이터를 저장하는 공간. 정적 변수나 전역 변수가 대표적.
- BSS 영역 : 데이터 영역에서 추가 구분을 위해서 별도로 지정함. BSS 영역을 구분하는 경우에는 데이터 영역은 초기값이 있는 데이터, BSS는 초기값이 없는 데이터를 저장한다.
- 힙 영역(heap segment) : 사용자가 직접 할당 가능한 저장 공간. 사용자가 직접 할당하고 싶은 만큼 메모리 영역을 확보하기에 반환할 때도 직접 반환해야 한다. 반환하지 않으면 메모리 누수(memory leak) 현상이 발생한다. 물론, 자동으로 힙 영역을 정리하는 가비지 컬렉터(garbage collector) 기능이 있는 프로그래밍 언어도 존재하나 그것만 믿고 그냥 메모리 할당하면 나중에 피를 볼 것이다.
- 스택 영역(stack segment) : 일시적으로 사용할 데이터 값이 저장되는 공간. 함수 매개변수, 지역 변수, 함수 복귀 주소 등이 그렇다.
    - 스택 영역은 스택이라는 이름 답게 함수 호출을 하는 경우 최상위 함수부터 마지막 호출한 함수까지 차곡차곡 호출 내역을 기록한다. 이를 스택 트레이스(stack trace)라 하며 디버깅할 때 찾아보면 좋다.

CPU에서의 프로세스 간 문맥 교환

- 프로세스는 CPU를 점유할 수 있는 시간이 어느정도 제한됨. 이 제한을 타이머(timer interrupt)로 알린다.
- 타이머 인터럽트로 프로세스가 자신의 업무를 마치고 할당된 자원이 다른 프로세스로 옮겨갈 때, 다시 자신의 차례에 중단되었던 부분부터 시작할 수 있도록 중단 시점의 정보들을 백업해야 한다
- 백업한 정보를 문맥(context)이라 하며 PCB에 문맥 정보가 들어가고 이 문맥 정보를 통해 PCB에 있는 프로세스 업무를 CPU에서 바로 받아 수행할 수 있는 것이다.
- 위와 같이 문맥을 PCB에 백업하고 PCB에 있는 다른 프로세스의 문맥을 불러오는 걸 문맥 교환(context switching)이라 한다.

프로세스 상태

- 생성(new) : 프로세스를 생성 중인 상태. 메모리에 적재되어 PCB 할당.
- 준비 상태(ready) : CPU를 할당 받아 일을 할 수 있지만, 자기 차례가 되지 않아 기다리는 상태. 준비 상태에서 실행 상태로 전환되는 걸 디스패치(dispatch)라고 한다
- 실행 상태(running) : CPU를 할당 받아 실행 중인 상태. 타이머 인터럽트 전까지 실행 중이고 타이머 인터럽트가 발생하면 준비 상태로 전환하고 연관된 장치들의 작업이 전부 끝나면 대기 상태로 전환
- 대기 상태(blocked) : 프로세스가 입출력 작업 요청 및 바로 확보할 수 없는 자원을 요청하는 등 바로 실행이 불가한 상황인 경우에 대기 상태가 된다.
- 종료 상태(terminated) : 프로세스가 종료된 상태. 운영체제는 PCB와 프로세스가 사용한 메모리 정리.

프로세스와 입출력 작업

- 블로킹 I/O : 실행 중인 프로세스가 시스템 콜로 입출력 장치를 요청하면 입출력 작업이 끝날 때까지 프로세스는 대기 상태이다.
- 논블로킹 I/O : 시스템 콜로 입출력 장치를 요청하면 입출력 장치가 먼저 사전 답변을 프로세스에게 보내고 프로세스는 자신의 일을 하다가 입출력 장치가 작업을 종료하고 알려주면 프로세스가 해당 업무를 실시한다.

멀티 프로세스와 멀티 스레드

- 멀티 프로세스 : 말 그대로 프로세스가 여러 개 실행되는 상황이다. 프로세스는 각 개별로 독립적인 자원을 할당 받기에 서로 영향을 거의 끼치지 않는 특징이 있다.
- 멀티 스레드 : 프로세스 내부에서 여러 업무를 수행하는 스레드가 다수 존재하는 상황이다. 프로세스 내부에서 업무를 분할한 것이기에 프로세스 내부 자원(동일한 주소 공간 코드, 데이터, 힙 영역)을 공유한다.

프로세스 간 통신(IPC)

- 공유 메모리 : 데이터를 주고받는 프로세스가 공통적으로 사용할 메모리 영역을 두는 방식. 즉, 메모리에서 특정 영역을 공유 영역으로 프로세스 내부 논리 구조에서 설정하고 그 영역을 사용한다. 메모리 영역에 접근하는 방식은 시스템 콜, 변수, 파일 등을 이용할 수 있다.
  - 커널의 개입이 거의 존재하지 않는다. 왜냐하면 프로세스가 메모리 영역을 확보하고 영역 내부에서 시스템 콜을 하지 않는 이상에야 커널이 개입할 여지가 없기 때문이다.
  - 단, 공유 메모리 영역 값을 수정하는 도중에 타이머 인터럽트가 들어오고 다음 작업을 하는 대상이 공유 메모리 영역 값을 마찬가지로 건드리는 프로세스라면? 데이터 일관성이 훼손되는 현상이 발생할 수 있다. 이를 레이스 컨디션(race condition)이라 한다.
- 메시지 전달 : 프로세스 간에 주고 받을 데이터가 커널을 거쳐 송수신되는 통신 방식
  - 메시지를 보내는 수단과 받는 수단인 시스템 콜 함수 구분이 명확하다. 수단으로는 파이프, 시그널, 원격 프로시저 호출, 소켓 등이 있다.
  - 파이프(pipe) : 단방향 프로세스 간의 통신 도구. 단방향이기에 한 쪽에서 쓰면 한 쪽은 읽을 수밖에 없다. 만약, 양방향 통신을 하고 싶으면 쓰기용 파이프, 읽기용 파이프 두 개를 구성한다.
  - 시그널(signal) : 프로세스에게 특정 이벤트가 발생한 걸 알리는 비동기 신호. 시그널 종류는 다양하며 시그널이 들어온 경우, 프로세스는 하던 일을 중단하고 신호에 맞는 시그널 핸들러 함수를 통해 일을 처리한다.
    - 사용자 정의 시그널을 제외한 시그널은 전부 수행할 기본 동작이 정해져 있다. 프로세스에 대한 비정상적인 개입 위주이며 특히 비정상적인 종료를 수행할 때(sigkill) 코어 덤프(core dump) 파일을 생성한다. 코어 덤프는 프로그램이 특정 시점에 작업하던 메모리 상태가 기록되어 있다.
  - 원격 프로시저 호출(RPC) : 원격 코드를 실행하는 기술. 프로세스 내에서 특정 코드를 실행하면 로컬 프로시저 호출, 프로세스가 원격으로 다른 프로세스 코드를 실행하면 원격 프로시저 호출이다. 대규모 트래픽 처리 환경에서 주로 쓰인다.

Quiz

- main 함수는 스레드일까?

- 지나치게 문맥 교환이 반복되면 어떤 문제가 발생하는가?
- 철학자가 식사하는 과정을 모사한 프로세스가 있다. 철학자는 스레드이며 철학자가 2명 있을 때 공유 자원으로 포크가 3개 있다. 철학자는 포크를 두 개 들어야만 식사를 할 수 있다. 이 때, 철학자 모두 식사 시간이 길어서 한 번 식사를 하면 다른 철학자는 시간 내 포크를 들지 못하면 죽는다. 그렇다면 해당 프로그램에서 철학자는 몇 명 죽는가?

3) 동기화와 교착 상태

동기화

- 프로세스나 스레드 간 공유하는 자원을 공유 자원(shared resource)이라 하며 당연하게도 무분별하게 접근하면 데이터 일관성이 깨질 수 있다
- 공유 자원에서 접근하는 코드 중, 동시 실행 시 문제가 발생할 수 있는 코드를 임계 구역(critical section)이라 하며 이 구역에 스레드 또는 프로세스가 접근할 시, 아무 조건 없이 진입하면 레이스 컨디션이 발생할 확률이 있다
- 레이스 컨디션을 방지하기 위해 프로세스와 스레드 둘 다 동기화(synchronization)을 거친다.

동기화 조건

- 실행 순서 제어 : 프로세스 및 스레드를 올바른 순서로 실행하기
- 상호 배제 : 동시에 접근해서는 안 되는 자원에 하나의 프로세스 및 스레드만 접근하기

동기화 기법

- 뮤텍스 락(mutex lock) : 상호 배제를 보장하는 동기화 도구. 임계 구역에 접근하기 위해서는 반드시 뮤텍스를 통해 자물쇠(lock)를 풀어야 하며 작업이 끝나면 뮤텍스를 통해 자물쇠를 다시 풀어 놓는다.
    - 즉, 스레드 A가 임계구역에 접근할 때, 뮤텍스 함수를 이용해 자물쇠를 채운다. 그러면 스레드 A가 일을 끝마칠 때까지 해당 임계구역은 접근할 수 없다.
- 세마포(semaphore) : 공유 자원을 한 명만 쓰지 않고 한 명 이상의 인원이 이용 가능할 때, 사용하는 동기화 도구. 공유 자원의 개수를 지정하여 가져가는 인원만큼 차감하여 자원 관리를 하고 작업을 종료한 프로세스 또는 스레드가 종료 함수를 통해 자원을 반환한다.
    - 세부적으로 이진 세마포와 카운팅 세마포로 구분하며 이진 세마포는 자원이 하나 밖에 없어서 사실상 뮤텍스와 유사하게 동작한다.
- 모니터(monitor) : 조건 변수(condition variable)를 통해 특정 조건 하에 프로세스 또는 스레드 실행/일시 중단하여 실행 순서를 제어한다. 즉, 실행 상태 <-> 대기 상태 전환으로 제어.
    - 프로세스 또는 스레드는 모니터로 진입하여 공유 자원을 얻기 위해 지정된 공유 자원 연산을 통과한다. 이미 모니터 내부에서 실행 중인 프로세스 또는 스레드가 있으면 별도로 마련된 큐에서 대기한다.

스레드 안전(thread safety)

- 멀티 스레드 환경에서 변수, 함수, 객체에 동시 접근이 이루어져도 실행에 전혀 지장이 없는 상태이다.
- 만약, 레이스 컨디션이 발생하면 이는 스레드 안전 상황이 아니다. 스레드 상황에서 각종 라이브러리 함수를 사용한다면 해당 함수가 스레드 안전을 보장하는 지 살펴 볼 필요가 있다.

교착 상태(deadlock)

- 부족한 자원을 할당받기 위해 기다리지만, 모두가 그 상태라 자원이 없어 무한 대기하는 상황이다.
- 발생 조건 : 아래 4개 조건을 전부 만족해야 교착 상태가 발생할 가능성이 있음
  - 상호 배제 : 한 프로세스가 사용하는 자원을 다른 프로세스가 사용할 수 없는 경우
  - 점유와 대기 : 한 프로세스가 자원을 점유한 상황에서 해당 자원을 받고자 대기하는 경우
  - 비선점 : 자원을 강제로 빼앗을 수 없어서 해당 프로세스의 작업이 끝나야만 자원을 이용할 수 있는 경우
  - 원형 대기 : 프로세스가 요청한 자원이 원의 형태를 이루는 경우. A는 B의 자원을 기다리고 B는 C의 자원을 기다리며 C가 A의 자원을 기다리는 상황.
- 해결 방법
  - 교착 상태 예방 : 4가지 필요 조건 중 하나라도 충족하지 않는다.
  - 교착 상태 회피 : 교착 상태가 발생하지 않도록 자원을 최대한 효율적으로 할당하는 것. 은행원 알고리즘(banker's algorithm) 같은 경우가 있음.
  - 교착 상태 검출 후 회복 : 교착 상태가 이루어졌을 때, 조치를 취하는 것.
    - 프로세스가 자원을 요구할 때마다 운영체제가 교착 상태 발생 여부를 확인
    - 교착 상태가 발생한 경우에는 자원 선점을 통해 프로세스를 돌아가게 하거나 교착 상태에 놓인 프로세스를 강제 종료함으로서 회복시킨다

Quiz

- 다음의 코드를 보고 레이스 컨디션이 발생할 지 생각해보시오.

```c
#include <stdio.h>
#include <pthread.h>

int shared_data = 0; // 공유 데이터

void* increment(void *arg) {
    int i;
    for (i = 0; i < 100000; i++) {
        shared_data++;
    }
    return NULL;
}

void decrement(void* arg) {
    int i;
    for (i = 0; i < 100000; i++) {
        shared_data--;
    }
    return NULL;
}

int main() {
```

```
    pthread_t thread1, thread2;

    pthread_create(&thread1, NULL, increment, NULL);
    pthread_create(&thread2, NULL, decrement, NULL);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    printf("Final value of shared_data: %d\n", shared_data);

    return 0;
}
```

- 철수는 5개의 도토리를 가지고 있다. 다람쥐를 꾀기 위해 도토리를 가지고 왔기에 최대한 많은 다람쥐가 도토리를 만지고 가기를 원한다. 단, 하나의 도토리는 하나의 다람쥐만 만질 수 있으며 다람쥐의 행위가 끝나야 다음 다람쥐가 만질 수 있다. 다람쥐를 스레드라고 가정할 때, 어떤 동기화 방법을 써야 효과적으로 수행할 수 있을까?
- 스레드가 안전하지 않은 메서드를 동기화하지 않으면 어떤 문제가 생길 수 있습니까?