3장 과제

1731017 김민정.

4. 프로세스는 프로그램과 다른가? 다르다면 어떤 면에서 다른가?

프로세스와 프로그램은 다르다.
기본적으로 프로그램은 명령어들의 집합이다. 프로그램은 디스크 저장 장치에 저장이 되어 다른 자원을 필요로 하지 않기 때문에 생명주기는 비교적 길다. 스스로 실행되는 것이 아니므로 비교적 수동적이라고 볼 수 있다. 반면, 프로세스는 프로그램이 실행되는 동안만 실행이 되어 생명주기가 제한 되어있다. 실행되는 동안에 CPU, 메모리 주소, 디스크, I/O 등의 자원을 필요로 하며 스스로 실행하거나 대기하고 있다.

2. 프로세스는 실행을 위해 메모리에 로딩된다. 그리고 프로세스의 코드가 실행되면서 접근하는 메모리 공간을 주소공간이라고 부른다. 주소 공간 내에는 어떤 요소들이 들어있는가? 간단히 설명하라.

주소공간은 사용자 주소 공간과 커널 주소 공간으로 나눌 수 있다.
먼저 사용자 주소 공간의 경우 코드, 데이터, 힙, 스택 영역이 있다.
코드 영역은 실행 중인 프로세스의 바이너리 코드들이 적재되는 영역으로 텍스트 영역이라고도 불린다. 데이터 영역은 실행 프로세스의 전역 변수와 정적 변수들을 위해 할당된 영역이다. 힙 영역은 프로세스가 실행 중에 동적 할당을 받는 영역으로 데이터 영역 아래에 위치하며 아래 방향으로 영역이 커진다. 스택 영역은 함수가 호출될 때, 지역변수, 매개변수, 함수의 리턴 값 등을 저장하기 위한 영역이며 위로 영역이 커진다.
각각의 프로세스별로 사용자 주소 공간을 가지는 것과 달리 영역을 공유하는 커널 주소공간은 커널 코드와 데이터, 스택 등이 있다.

CamScanner로 스캔하기

3. 32비트 컴퓨터에서 한 프로세스의 주소 공간(한 프로세스가 접근할 수 있는 주소 혹은 한 프로세스에게 허락된 주소의 범위)의 크기는 최대 얼마인가?
이 주소 공간은 가상주소 공간인가, 실제 주소 공간인가?

32bit 컴퓨터에서 한 프로세스의 주소 공간의 크기는 최대 4GB이다.
이 주소 공간은 가상주소공간이다. 0번지부터 시작하여 해당 프로세스가 연속적으로 메모리를 독점하고 있다고 가정하는 가상 주소 공간을 사용함으로서 다른 프로세스들 간 메모리의 독립성을 유지할 수 있다.

4. 어떤 프로세스 내 i++; 이라는 코드가 있다. 변수 i의 주소를 출력해보니, 3000 이었다. 3000은 실제 메모리의 주소인가? 가상주소인가? 왜 가상주소라고 생각하는가?

3000은 가상주소이다. 실제 메모리 공간은 세그먼트별 영역에 비연속적으로 할당되어있다. 근데 가상 주소에서 이를 연속적으로 할당함으로서 응용프로그램에서 메모리에 접근하는 것이 용이해진다. 그리고 만약 응용 프로그램 상에 실제 메모리 주소가 노출되면 사용자가 악용을 하거나, 잘못 접근하는 경우 컴퓨터 시스템에 큰 문제가 일어날 수도 있다.

5. 각 프로세스의 주소 공간은 모두 0번지부터 시작하므로, 충돌이 발생할 것으로 보인다. 운영체제는 이 문제를 어떻게 해결하는가?

논리/가상 주소 공간에서는 사용자의 편의성을 위해 0번지부터 코드 영역이 시작되므로 각 프로세스가 접근할 때 충돌이 발생될 것처럼 보이지만, 매핑 테이블이 실제 메모리 공간에 접근하여 각 프로세스를 다른 영역에 적재시키기 때문에 충돌이 발생하지 않는다.

CamScanner로 스캔하기

6. 프로세스의 크기와 프로세스의 주소 공간은 같은 의미인가? 서로 다른 의미인가? 설명하라.

프로세스의 크기와 프로세스의 주소 공간은 다른 의미이다.
프로세스의 크기는 코드영역, 데이터 영역만으로는 정할 수 없다. 프로세스가 실행 중에 동적으로 메모리를 할당 받는경우 힙이 아래로, 스택이 위로 영역이 늘어나게 된다. 즉 프로그램 크기는 실행되기 전까지, 그리고 실행 시점에 따라 유동적으로 변하므로 정확한 크기를 알 수 없다. 따라서 현재 프로세스가 유동적으로 차지하고 있는 공간이 아닌 프로세스가 최대로 커질 수 있는 메모리 공간을 이용한다. 이를 프로세스의 주소 공간이라고 한다. 이는 CPU가 엑세스할 수 있는 최대 주소 공간이다.

7. 모든 프로세스의 주소 공간에서 사용자 공간은 분리되지만, 커널 공간은 공유된다. 그 이유는 무엇인가?

프로세스가 실행되면서 코드, 데이터, 힙 영역 등 사용자 영역에 메모리가 저장이 되는경우, 저장되는 형태와 크기 등 경우의 수가 모두 다른데 반해 시스템 호출이나 인터럽트 등이 발생하여 커널 영역의 커널 코드가 실행되는 경우 처리해야 하는 내용은 동일하므로 커널 공간을 공유한다.

8. 프로세스는 생성에서 소멸까지의 상태가 변하면서 실행되어 간다. Ready, Run, Terminated, Zombie, Blocked 5 상태에 대해 간단히 설명하라.

Ready는 프로세스가 스케줄링을 기다리는 준비상태이다. ready 상태의 프로세스들은 커널에 있는 준비 큐에 들어간다. 현재 실행 중인 프로세스가 중단하게 되는 경우, 커널은 준비큐에서 한 개의 프로세스를 선택하는 프로세스 스케줄링을 한다. 프로세스가 ready 상태로 되는 경우는 new 상태에서 준비 큐에 삽입이 될 때, running 상태에서 프로세스에게 할당 된 시간이 경과하거나 프로세스가 자발적으로 다른 프로세스에게 CPU 사용을 양보할때, 혹은 입출력 장치나 저장장치로 부터 요청한 작업이 완료되었을 때이다.

CamScanner로 스캔하기

Running은 프로세스가 현재 CPU 에 의해 실행되는 상태이다. 커널은 CPU 스케줄링을 통해 선택된 프로세스의 PCB에 상태를 running으로 기록한다. 실행 중인 프로세스의 시간 할당량을 경과할 때, 커널은 ready 상태로 만들고 프로그램의 실행이 종료되면 terminated 상태로 만든다. 커널은 프로세스를 running 상태에서 다른 상태로 바꿀 때 컨텍스트 스위칭을 진행한다. 실행 중인 프로세스의 컨텍스트를 PCB에 저장하고 CPU 스케줄링에 의해 선택된 프로세스의 PCB에서 컨텍스트를 CPU에 복귀시킨다.

Terminated 는 Terminated/Zombie, Terminated/Out 두 가지 상태로 나뉜다. out 상태의 경우 zombie 상태인 프로세스의 PCB로부터 종료코드를 읽어 PCB를 삭제하고 프로세스 테이블에서 제거한 상태를 말한다.

Zombie 프로세스는 차지하던 메모리와 프로세스가 할당받은 자원들을 모두 반환시키고 닫는것을 말한다. 이때 프로세스가 완전히 사라지지 않은 상태를 말한다.
프로세스가 종료할 때 종료 코드를 남기는데 이 종료코드는 프로세스의 PCB에 저장된 채 남아있게된다. 커널은 부모 프로세스가 이 종료코드를 읽어갈 때까지 zombie 상태에서 해제하지 않고 PCB도 제거하지 않으며 프로세스 테이블에 그대로 남겨둔다.

Blocked는 프로세스가 요청한 자원이 준비되기를 기다리거나 입출력 요청이 끝나기를 기다리는 상태이다. 프로세스의 실행 중 파일 읽기나 네트워크 수신 등의 입출력, 타이머 기다리기, 자원 요청 등의 시스템 호출이 일어나면 현재 프로세스는 기다리는 상황이 완료될 때까지 더 이상 실행을 계속 할 수 없기 때문에 커널은 현재 프로세스를 blocked 또는 wait 상태로 만들도록 CPU 스케줄링을 통해 ready 상태의 프로세스를 선택하고 두 프로세스 사이에 컨텍스트 스위칭을 한다.

CamScanner로 스캔하기

9. 운영체제 커널이 만드는 것으로 프로세스의 정보를 저장하는 구조를 PCB라고 부른다. 어떤 정보들이 저장되는지 간단히 설명하라.

PCB에 저장되는 정보들은 다음과 같다.

첫째, 프로세스 번호(PID)이다. 프로세스의 고유 번호로 식별을 위해 사용한다.

둘째, 부모 프로세스 번호(PPID)이다. 부모 자식의 관계 형성을 위해 사용한다.

셋째, 프로세스 상태(process state)이다. 프로세스가 생성된 후 종료될 때까지 실행동안 커널에 의해 여러 상태로 바뀐다.

넷째, CPU의 컨텍스트 정보(PC, IR 등 CPU 레지스터이다.).
커널은 현재 프로세스를 중단시키고 다른 프로세스를 실행시킬 때, 현재 프로세스를 실행하고 있던 상황(컨텍스트)를 PCB에 저장한다. 여기서 CPU 레지스터 값들이 PCB에 저장이 되는데, 레지스터에는 다음에 실행할 코드의 주소 등이 저장되어 있다.

다섯째, 스케줄링 정보이다. 프로세스의 priority, nice 값 등을 이용하여 프로세스 스케줄링에 이용한다.

여섯째, 종료코드. 프로세스가 종료할 때 부모 프로세스에게 전달하는 정수 값으로, 종료한 프로세스의 PCB에 저장된다. 이는 프로세스가 어떤 이유로 종료하였는지 부모에게 전달하기 위해 넘기는 정수 값이다.

일곱째. 프로세스의 오픈 파일 테이블 혹은 리스트. 프로세스가 실행 중에 연어놓은 파일에 대한 정보들은 프로세스별 오픈 파일 테이블에 유지되는데, 이 테이블은 일반적으로 PCB 내에 저장된다.

여덟째, 메모리 관리를 위한 정보들. 메모리 관련 레지스터들로 프로세스의 주소공간에서의 가상주소를 물리주소로 변환하는 과정에서 사용된다.

아홉째, 회계정보이다. 프로세스의 CPU 총 사용시간. 프로세스가 실행을 시작하여 경과한 총 시간, 프로세스의 제한 시간 등이 저장된다.

마지막으로, 프로세스의 소유자 이름, 즉 사용자의 로그인 이름 정보이다.

CamScanner로 스캔하기

10. 종료코드란 무엇인가? 왜 필요하며 누가 누구에게 전달하는가?
전달하고 받는 방법은 무엇인가?

Return Code 혹은 exit code는 자식 프로세스의 종료이유를 알리기위해 자식 프로세스가 부모 프로세스에게 전달하는 값이다. 자식 프로세스는 종료 시에 코드, 데이터, 스택, 힙 등의 모든 메모리 자원을 반환시키고, 열어놓은 파일이나 소켓 등을 닫는다. PCB와 프로세스 테이블의 항목은 그대로 두고 PCB내 프로세스 상태를 terminated로 바꾸고 종료코드를 저장한다. 부모 프로세스가 wait() 시스템 호출을 통해 PCB를 방문하여 자식이 남겨 놓은 종료코드를 읽으면, 자식 프로세스의 PCB를 반환하고 프로세스 테이블 항목을 제거하여 완전히 종료한다.

11. 프로세스는 운영체제의 스케줄링 단위인가?

멀티 태스킹은 태스크를 실행단위로 여러 태스크를 동시에 실행시키는 기법이다. 과거에는 프로세스를 하나의 실행단위로 보고 여러 프로세스를 동시에 실행시키는 멀티 태스킹을 하고자 하였다. 즉 프로세스는 운영체제의 스케줄링 단위였다. 하지만 각 일을 처리하기 위해서는 자식 프로세스를 추가로 생성하여 처리를 하기 때문에 생성비용이 만만치 않았다. 자식 프로세스마다 독립적인 메모리 공간을 할당하고 부모 프로세스를 복사하는 과정에 드는 비용이 있다. 그리고 PCB, 페이지 테이블 등 프로세스 관리를 위한 구조체를 생성하는 데에도 많은 시간이 걸린다.
다음으로는 프로세스 간 통신의 어려움이 있었다. 각 프로세스들은 상호독립된 메모리 공간을 가지고 있어서 프로세스가 다른 프로세스의 메모리를 접근하려면 공유메모리, 신호, 파이프 등을 이용해서 다른 데이터를 주고 받았어야 했다.
그 외에도 현재 실행 중인 프로세스를 중단시키고 다른 프로세스를 실행시키는 컨텍스트 스위칭 과정에서 시간적, 공간적 오버헤드가 컸다.

따라서 이와 같은 문제를 해결하고자 스레드가 등장하였다. 스레드는 프로세스보다 크기가 작은 실행 단위로서 프로세스의 생성 및 소멸에 따른 오버헤드를 감소시켜며, 프로세스보다 빠른 컨텍스트 스위칭이 가능하도록 한다. 그리고 프로세스의 통신 시간, 방법, 코딩 등의 어려움을 해소시켜주었다.

CamScanner로 스캔하기

현대의 멀티 스레드 운영체제에서 실행단위는 따라서 더 이상 프로세스가 아니라 스레드이다. 프로세스는 여러 개의 스레드들이 실행될 때 자원을 공유하는 컨테이너로 역할이 바뀌었다. 실행 단위가 스레드이기 때문에 오늘날 대부분의 운영체제는 스레드 스케줄링을 실행한다.

12. 어떤 프로세스를 고아 프로세스라고 부르는가?

대부분 프로세스는 exit() 시스템 호출을 부르면서 종료하지만 비정상 종료하는 경우도 있다. 둘 중 어느 경우이든, 부모가 먼저 종료된 자식 프로세스들을 고아 프로세스라고 한다. 프로세스를 종료할때, 자식 프로세스들이 있으면 커널은 부모 자식 관계의 일관성을 유지하기 위해 고아가 된 이들의 부모를 init 프로세스로 죽각 바꾼다. 그러나 init 이 버린 자식들의 부모가 되었다고 하더라도 자식들은 원래 부모를 잃어버렸기 때문에 이들을 여전히 고아 프로세스라고 한다.

13. 좀비 프로세스는 해로운 것인가? 좀비 프로세스를 없애려면 어떻게 해야하는가?

좀비 프로세스는 프로세스가 종료하였지만, 부모 프로세스가 wait()을 실행하기 전에 종료하였기 때문에 자신이 남긴 종료코드가 부모에 의해 읽혀지지 않아 완전히 종료되지 못한 상태를 말한다. 좀비 프로세스는 모든 자원을 반환하여 할당된 메모리가 없는 죽은 프로세스이기 때문에 프로세스 실행에 심각한 문제를 발생시키지는 않는다. 하지만 프로세스 테이블의 크기가 제한되어있으므로 프로세스 ID가 부족해 새로운 프로세스를 생성하지 못할 수 있기 때문에 해롭다.

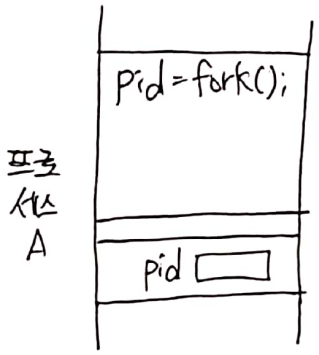
따라서 좀비 프로세스를 없애려면 쉘에서 KILL 명령으로 부모 프로세스에서 SIGCHLD 신호를 보내면 된다. 만약 부모 프로세스가 SIGCHLD 신호를 무시하도록 작성되어있다면 부모 프로세스를 죽이면 된다. 그러면 좀비 프로세스의 부모는 결과적으로 init 프로세스가 되고 init 프로세스는 주기적으로 wait() 시스템 호출을 실행하기 때문에 좀비 프로세스가 제거된다.

CamScanner로 스캔하기

14. fork() 와 exec() 의 동작과정을 그림으로 그리면서 설명하라.

1) fork()

(a) fork() 호출 전 프로세스 A 실행



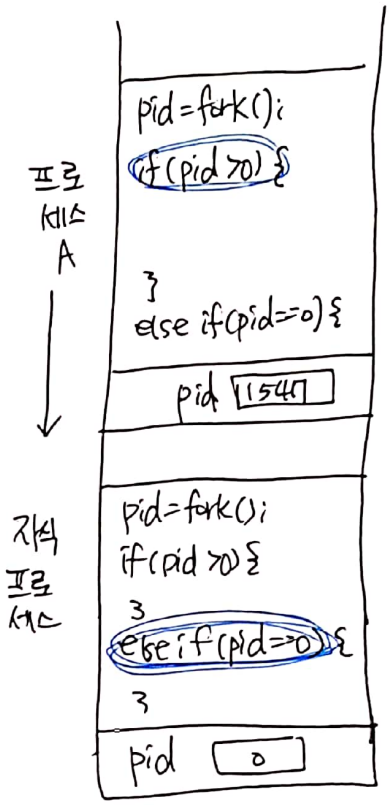
(b) fork() 실행

1) 자식 프로세스의 PID를 15144로 결정

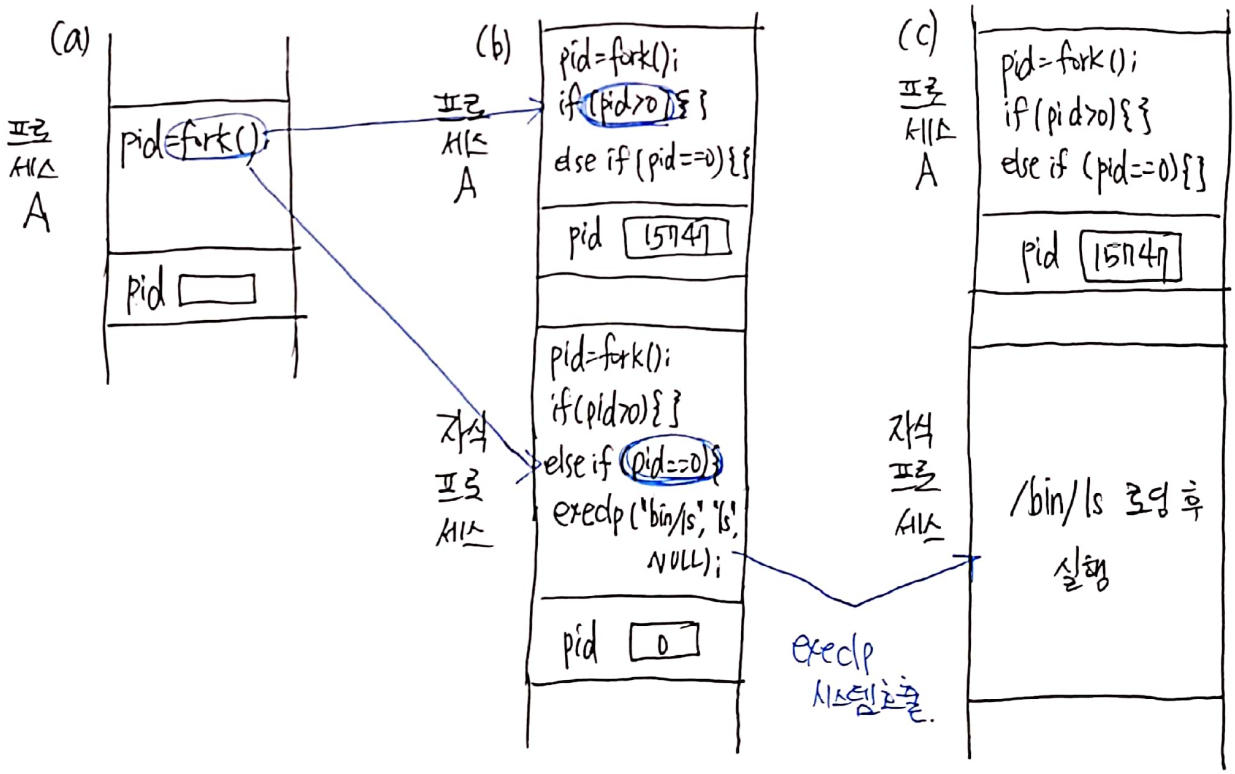
2) 부모 프로세스를 복사하여 자식 프로세스 구성

3) 자식 프로세스 만들고 리턴

(c) fork()로부터의 리턴, 자식 프로세스 생성



2) exec()



CamScanner로 스캔하기

15. fork() 시스템 호출을 사용하여 프로세스 자신은 6 에서 10 까지 합을 구하고 자식 프로세스에게는 1 에서 5 까지 계산시켜 1 에서 10 까지의 합을 출력하는 프로그램을 작성하라.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

int main(void) {
    pid_t pid;
    int status, i, sum;
    pid = fork();
    if (pid > 0) {
        for (i=6; i<=10; i++) {
            sum += i; }
        printf("parent : sum= %d \n", sum);
        wait (&status);
        sum += WEXITSTATUS (status);
        printf ("total: sum = %d \n", sum);
        return 0;
    }
    else if (pid == 0){
        sum=0;
        for (i=1; i<=5; i++) {
            sum+=i; }
        printf ("child: sum= %d \n", sum);
        return sum;
    }
    else {
        fprintf (stderr, "fork error");
        return 1;
    }
}
```

CamScanner로 스캔하기