

운영체제 Report 2: 프로세스 생성

2019312014 박병준

2023-2 학기 류은석 교수님 강의

-과제 시작 전 리눅스 환경설정 구축

3 년 전 리눅스 시스템 과목을 수강할 때 가상머신으로 VirtualBox 를 설치하고, 그 위에 리눅스 배포판 중 하나인 Ubuntu 를 설치하여 리눅스 환경을 구축한 적이 있었다. 이때 구축한 환경은 사용하기에도 쉽고 호환성이 좋아서 과제를 하는 데 큰 문제는 없었지만, 상당히 느리고 리소스 사용량이 높다는 단점이 있었다. 따라서 이번 기회에 VirtualBox 에서 벗어나 빠르고 가벼운 환경에서 리눅스를 사용해 앞으로의 과제를 수행해보고자 했다. 이를 위해 다양한 선택지를 고민해 본 결과 WSL 에 대해 알게 되었다. WSL 은 Windows Subsystem for Linux 의 줄임말로, 윈도우의 가상화 기능을 활용해서 가상머신을 구축하지 않아도 윈도우 위에서 리눅스를 사용할 수 있는 특징이 있다. 이를 이용하여, 평소에 쓰는 VSCode 에 WSL2 를 연동하여 리눅스 환경 구축을 시도했다. 리눅스 배포판은 3 년 전 사용했던 Ubuntu 20.04 LTS 버전을 그대로 사용했다. 버전은 그대로인 대신 컴파일러 gcc 와 g++을 최신 버전으로 업그레이드했고, 디버거인 gdb 를 새로 설치했다.

VSCode 에서 Extention 으로 Remote Development 을 설치하고, WSL 명령창에서 VSCode 를 실행할 디렉토리로 이동한 다음 code .을 입력하자, WSL 환경으로 VSCode 가 실행되었다. 이제 VSCode 좌측 하단의 >< 버튼을 통해 로컬과 WSL 환경을 자유롭게 이동할 수 있게 되었다.

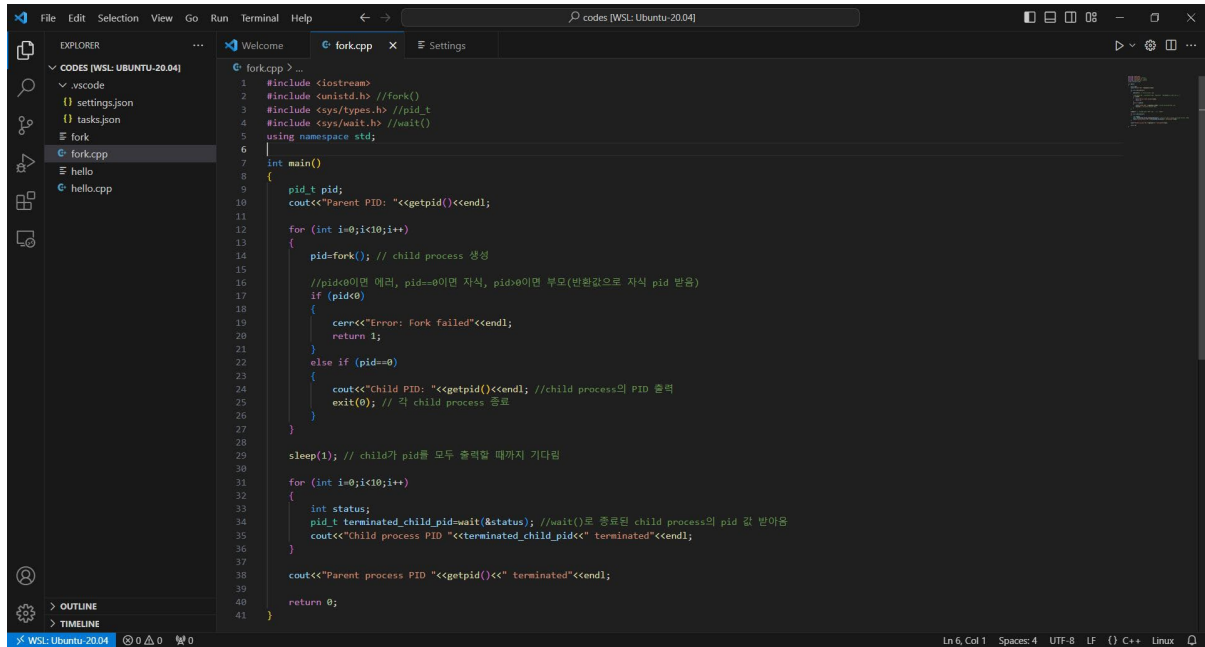
-과제: 리눅스에서 fork 명령어를 사용해 child process 를 10 개 생성하고 PID 를 출력한 후 각각 종료시킨 뒤 parent process 도 종료하는 프로그램 구현

-공부한 내용

프로세스는 프로그램이 메모리에 올라가 실행 중인 프로그램을 의미한다. 프로세스 생성 과정은 부모(parent) 프로세스 자식(child) 프로세스를 만든다는 단순한 명제로부터 출발한다. PID(process identifier)로 프로세스를 식별하고 관리할 수 있다. Fork() 명령어는 새로운 프로세스를 만드는 시스템 콜이다. 실행 중인 부모 프로세스를 복사하여 똑같은 내용의 자식 프로세스가 새로 생성된다. 후술할 과제 코드를 보면 알겠지만, fork()함수가 실행되기 이전에는 기존의 프로세스 1 개만이 실행되다가, fork()함수가 실행된 후에는 기존의 프로세스(부모) 1 개와 새로운 프로세스(자식) 1 개, 총 2 개의 프로세스가 동작한다. 프로세스 종료(termination)와 관련하여, 자식 프로세스는 작업이 끝나면 exit() 시스템 콜로 운영체제에 삭제를 요청한다. 리소스는 운영체제에 의해 할당 해제되며, 종료된 자식 프로세스는 부모 프로세스에 status 값을 리턴하고, 부모는 wait()를 통해 자식의 종료를 기다리다 값을 전달받는다. Wait()는 종료된 프로세스의 PID 를 반환한다(pid=wait(&status) 코드 형태).

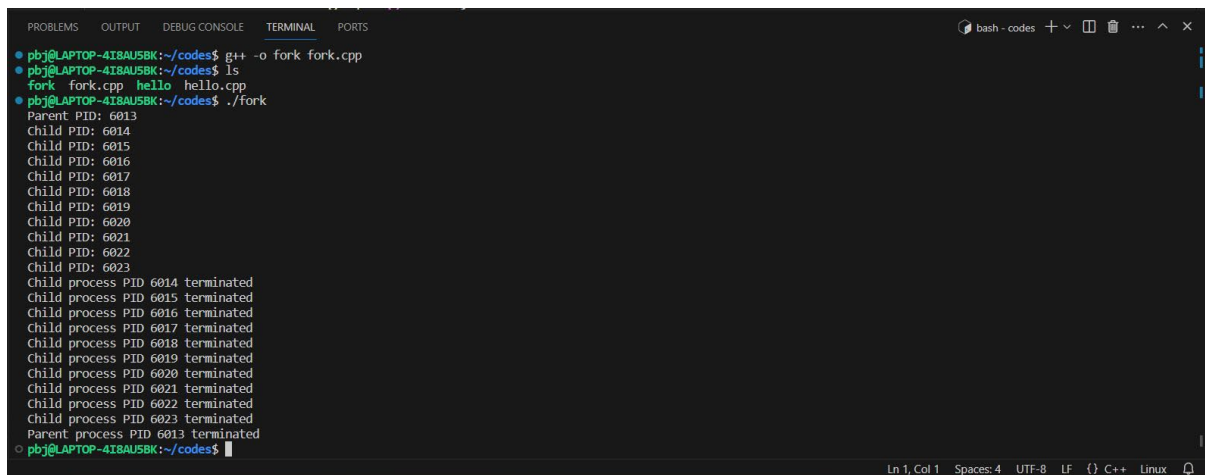
이번 과제에선 사용되지 않았지만 알게 된 내용은 다음과 같다. Exec() 명령어는 fork() 이후 프로세스의 메모리 공간을 새 프로세스로 대체하는 시스템 콜이다. 자식 프로세스는 부모 프로세스의 자원을 공유하여 사용할 수 있는데, 부모 프로세스는 자식 프로세스가 사용하는 자원을 제한할 수 있다. 실행 옵션과 관련하여, 부모 프로세스는 자식 프로세스와 동시에 실행되거나, 자식 프로세스가 모두 종료할 때까지 기다린다. 프로세스 종료와 관련하여, 부모 프로세스는 abort() 시스템 콜로 자식 프로세스의 수행을 강제로 종료할 수 있다. Abort()는 자식 프로세스가 할당된 리소스를 초과하여 작업을 수행하거나, 자식에게 할당된 작업이 더 이상 필요하지 않거나, 부모 종료 시 자식이 먼저 종료되도록 하기 위해 사용된다. 좀비 프로세스란 프로세스가 종료되었지만 부모 프로세스가 아직 wait() 호출을 하지 않은 프로세스를 뜻한다. 커널은 자식 프로세스가 종료되더라도 최소한의 정보(PID, 프로세스 종료 상태인 status 등)를 가지고 있기 때문에, 부모 프로세스가 wait() 호출을 해야 좀비 프로세스 문제가 해소된다. 반대로 부모 프로세스가 자식 프로세스보다 먼저 종료되면 그 자식 프로세스는 고아 프로세스가 된다.

-구현한 코드 내용



```
1 #include <iostream>
2 #include <unistd.h> //fork()
3 #include <sys/types.h> //pid_t
4 #include <sys/wait.h> //wait()
5 using namespace std;
6
7 int main()
8 {
9     pid_t pid;
10    cout<<"Parent PID: "<<getpid()<<endl;
11
12    for (int i=0;i<10;i++)
13    {
14        pid=fork(); // child process 생성
15
16        //pid<0이면 에러, pid==0이면 자식, pid>0이면 부모(반환값으로 자식 pid 받음)
17        if (pid<0)
18        {
19            cerr<<"Error: Fork failed"<<endl;
20            return 1;
21        }
22        else if (pid==0)
23        {
24            cout<<"Child PID: "<<getpid()<<endl; //child process의 PID 출력
25            exit(0); // 각 child process 종료
26        }
27    }
28
29    sleep(1); // child가 pid를 모두 출력할 때까지 기다림
30
31    for (int i=0;i<10;i++)
32    {
33        int status;
34        pid_t terminated_child_pid=wait(&status); //wait()로 종료된 child process의 pid 값 받아옴
35        cout<<"Child process PID "<<terminated_child_pid<<" terminated"<<endl;
36    }
37
38    cout<<"Parent process PID "<<getpid()<<" terminated"<<endl;
39
40    return 0;
41 }
```

-실행 화면



```
pbj@LAPTOP-418AU5BK:~/codes$ g++ -o fork fork.cpp
pbj@LAPTOP-418AU5BK:~/codes$ ls
fork fork.cpp hello hello.cpp
pbj@LAPTOP-418AU5BK:~/codes$ ./fork
Parent PID: 6013
Child PID: 6014
Child PID: 6015
Child PID: 6016
Child PID: 6017
Child PID: 6018
Child PID: 6019
Child PID: 6020
Child PID: 6021
Child PID: 6022
Child PID: 6023
Child process PID 6014 terminated
Child process PID 6015 terminated
Child process PID 6016 terminated
Child process PID 6017 terminated
Child process PID 6018 terminated
Child process PID 6019 terminated
Child process PID 6020 terminated
Child process PID 6021 terminated
Child process PID 6022 terminated
Child process PID 6023 terminated
Parent process PID 6013 terminated
pbj@LAPTOP-418AU5BK:~/codes$
```