컴퓨터소프트웨어학부
2022074057
홍바다

Project01 - Wiki

I. Design

A. getgpid() (get grand parent process id) 시스템 콜 구현

조부모 프로세스의 id를 반환하는 시스템 콜인 getgpid()를 구현하기 위해 비슷한 시스템 콜인 getpid()를 먼저 분석하였다. 이때 grep 명령어를 사용하여 "getpid"라는 키워드가 사용된 파일들을 찾았다.

→ grep -rn "getpid"

위 명령어는 현재 디렉토리와 그 하위 디렉토리에 있는 모든 파일에서 "getpid"에 대한 재귀 검색("-r")을 수행한다. 그리고 "-n"이 "getpid"와 일치하는 파일 이름과 줄 번호를 표시해준다. 그 결과, sysproc.c 파일에서 getpid()가 정의된 걸 볼 수 있었다.

→ int sys_getpid(void) {
    return myproc()->pid;
}

이 코드를 기반으로 getgpid()를 어떻게 구현해야할지 계획했다.

먼저, myproc()이 정의되어있는 proc.c에서 myproc()이 어떻게 구현되었고, proc이라는 구조체가 어떻게 사용되는지 분석해보았다. proc.c에서 다른 함수들이 proc 구조체를 사용하는 것을 분석한 결과, proc 구조체에는 부모 프로세스인 parent가 저장되는 것을 확인하였다. 따라서 myproc()->parent는 부모 프로세스에 접근하는 것이고, myproc()->parent->parent는 조부모 프로세스에 접근하는 것이기에 getgpid()에서 조부모 프로세스에 접근하는데 쓰일 수 있을 것이라 판단하였다.

B. User program 구현

getgpid() 시스템 콜이 유저 프로그램에서도 사용될 수 있어야하기 때문에 일단 먼저 실습에서 진행한 바와 같이 user.h과 usys.S에 해당 시스템 콜을 등록해야겠다고 생각했다. 그 뒤에 새로운 유저 프로그램을 생성하기 위해 과제 명세에 명시되어있는 바와 같이 project01이라는 파일에 메인함수를 정의해야겠다고 생각했다. 이때, 부모 프로세스와 조부모 프로세스의 id를 모두 출력해야하기 때문에 getpid와 getgpid를 사용해야한다.

II. Implement

A. getgpid() 구현

먼저, proj01_syscall.c라는 파일을 xv6 디렉토리에 생성한 후 해당 파일에서 getgpid() 함수를 구현했다. 이 동작은 커널에 새로운 시스템 콜을 구현하는 것과 같다. 위에서 설명한 바와 같이 myproc()->parent->parent->pid를 사용하여 조부모 프로세스의 id를 찾을 수 있도록 했다. 이 함수는 id를 반환해야하기 때문에 int로 반환타입을 정해주었다. 매개변수는 필요없다고 판단하였다.

```c
#include "types.h"
#include "defs.h"
#include "param.h"
#include "mmu.h"
#include "proc.h"

//system call for getting grand parent process id
int
getgpid(void)
{
        return myproc()->parent->parent->pid;
};
```

또한, getgpid()에 대한 wrapper 함수인 sys_getgpid()를 만들어 한단계 걸러 getgpid()에 접근할 수 있도록 하였다. 그래서 sys_getgpid()는 getgpid()의 결과값을 반환하는 함수이다. 이는 후에 getgpid 함수를 확장하거나 최적화시킬때 유용하게 사용될 것이다.

```
//Wrapper for getgpid()
int
sys_getgpid(void)
{
        return getgpid();
};
~
```

그 다음 proj01_syscall.o를 Makefile에 넣어주었다. 이 동작은 나중에 "make" 명령어를 쳤을때 proj01_syscall에 있는 getgpid() 함수가 빌드될 수 있게 해준다. 더불어 defs.h와 user.h 헤더파일에 getgpid() 함수를 선언해주어 각각 다른 c 파일들과 유저 프로그램에서도 위 함수를 사용할 수 있게 한다. 또한, syscall.h와 syscall.c 파일들에 wrapper 함수인 sys_getgpid()를 저장하여 시스템 콜로써 사용될 수 있도록 해준다. 마지막으로 usys.S에서도 getgpid를 등록하여 유저 프로그램에서 이를 시스템 콜로 사용할 수 있게 해준다. 이 동작들은 후에 유저 프로그램을 구현할 때 getgpid()를 사용하여 조부모 프로세스의 id를 얻을 수 있게 한다.

B. User program 구현

다음으로 유저 프로그램을 구현하였다. 먼저 과제 명세대로 유저 프로그램 project01을 생성하기 위해 c파일을 xv6 디렉토리에 만들었다. 여기에는 메인함수를 구현하였는데, 나의 학번과 부모 프로세스, 그리고 조부모 프로세스의 id를 출력하도록 해주었다. printf() 형식에 맞게 정수타입인 1을 먼저 넣어주고 해당 출력문을 넣어주었다. 학번에 관한 출력문은 고정된 출력문이기에 다른 변수를 사용하지 않았고, 부모 프로세스의 id인 pid와 조부모 프로세스의 id인 gpid를 위한 변수를 선언해주었다. 각각에 맞는 getpid()와 getgpid()를 할당하여 반환값이 저장되도록 해주었다. pid와 gpid까지 과제 예시대로 출력하게 해준 후 exit()를 통해 프로그램이 끝나도록 했다.

```
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[])
{
        printf(1, "My student id is 2022074057\n");
        int pid = getpid();
        int gpid = getgpid();
        printf(1, "My pid is %d\nMy gpid is %d\n", pid, gpid);
        exit();
};
```

유저 프로그램 구현을 마친 후 Makefile을 수정해주었다. project01을 새롭게 생성한 뒤 이를 Makefile의 유저 프로그램 리스트에 추가해주고 해당 c파일도 넣어주었다. 이를 통해 위의 소스코드가 컴파일 되어 다른 유저 프로그램들과 연결시킬 수 있게 된다.

III. Result

A. 컴파일 및 실행

먼저, 프로그램을 실행하기 위해 "make clean" → "make" → "make fs.img" → "./bootxv6.sh" 순으로 명령어를 작성했다. 아래 사진에 보이듯이 qemu에 연결이 된 후 유저 프로그램인 project01을 실행시켰다. 그 결과 위에서 메인함수에 작성한 코드대로 나의 학번 정보와 해당 프로세스의 부모 프로세스와 조부모 프로세스의 id인 pid와 gpid가 각각 출력된 것을 볼 수 있었다.

Make clean을 통해 프로젝트를 다시 컴파일하기 전에 깨끗한 상태를 보장하고, make를 통해 Makefile에 있는 필요한 모든 소스 파일들을 호출하여 프로젝트를 컴파일한다. 그리고 make fs.img를 통해 xv6를 부팅하는데 필수적인 fs.img 파일을 생성하고 bootxv6.sh라는 쉘 스크립트를 실행한다. 이렇게 xv6 운영체제를 컴파일하고 실행하는 단계를 거쳤다. 그 후 해당 쉘에서 project01 유저 프로그램을 실행시켰다. 그 결과 Makefile에 등록해둔 project01.c 소스 파일의 메인 함수가 실행되었고, 코드에 따라 나의 학번 정보가 먼저 출력되었다. 그 다음 유저 프로그램, 즉 유저 모드로써 getpid() 시스템 콜을 부르게 된다. 그러면 커널 모드로 전환이 되고, 해당 시스템 콜을 실행시킨다. getpid()의 일이 끝나면 다시 유저 모드로 돌아오게 되고, pid 변수에 해당 프로세스(project01)의 프로세스 id(pid = 1)을 반환값으로 주게 된다. gpid 변수에도 조부모 프로세스의 pid를 저장하기 위해 같은 과정을 거치게 된다. 이전에 user.h와 usys.S에 getgpid() 시스템 콜을 유저 모드에서 부를 수 있게 설정해 두었다. 그리고 커널 모드에서 getgpid() 시스템 콜을 실행시키는 것은 사전에 syscall.c와 syscall.h에 등록해주었기에 가능한 일이다. 이때 getgpid() 함수에 정의한 것을 토대로 project01 프로세스(pid = 3)의 부모 프로세스(pid = 2)의 부모 프로세스(pid = 1), 즉 조부모 프로세스의 pid를 반환한다. 이렇게 pid와 gpid가 모두 저장된 후 다음 줄이 실행되면 아래와 같이 pid와 gpid 정보가 각각 출력된다.

```
Booting from Hard Disk...xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ project01
My student id is 2022074057
My pid is 3
My gpid is 1
$ QEMU: Terminated
```

IV. Trouble Shooting

과제 수행 중 2가지 문제점이 있었다. 첫번째로는 printf() 함수를 사용할 때 어려움이 있었다. 지금까지는 printf("any sentences"); 이런식으로 함수의 argument에 바로 const char* 타입의 문자열을 넣어주었다. 하지만 printf("My student id is 2022074057");로 실행하자 printf() 함수의 첫 인수는 정수타입의 값이 와야한다는 에러 메세지가 떴다. 그래서 실습 자료를 다시

공부하며 차이점을 찾았고, printf(1, "My student id is 2022074057"); 형식으로 써야한다는 것을 깨달았다. 1을 인수로 넣어 다시 실행시키니 문제가 해결되었다.

두번째 문제점은 해결하는데 시간이 꽤 오래걸렸다. 무엇이 문제인지 파악하는데 어려움을 겪었다. 아래의 사진에 나와있듯, proj01_syscall.c 파일에서 발생한 에러를 보면, 기존의 내가 작성하지 않은 파일인 proc.h에서 에러가 나와 당황했다. struct taskstate라는 구조체가 완전한 타입이 아니었고, NSEGS는 해당 파일에서 선언된 변수가 아닌걸로 보였다.

```
In file included from proj01_syscall.c:4:
proc.h:5:20: error: field ‘ts’ has incomplete type
    5 |   struct taskstate ts;         // Used by x86 to find stack for interrup
t
      |                    ^~
proc.h:6:22: error: ‘NSEGS’ undeclared here (not in a function)
    6 |   struct segdesc gdt[NSEGS];   // x86 global descriptor table
      |                      ^~~~~
```

위 문제점들을 원초적으로 해결하기 위해 taskstate 구조체와 NSEGS가 사용되는 파일들을 분석해보기로 했다.

→ grep -r -e "taskstate" -e "NSEGS" .

위 명령어를 통해 "taskstate"와 "NSEGS"가 사용된 모든 파일들을 현재 디렉토리와 하위 디렉토리에서 찾았다. 그 결과, 두 문자열들이 mmu.h와 param.h에서 동시에 포함되는 걸 알 수 있었다. 이 헤더파일들을 에러 표시가 난 proc.h 상단에 직접 포함해주었더니 다시 같은 에러가 났다. 그래서 다시 에러가 발생했던 proj01_syscall.c 파일의 상단에 포함시켜주었다. 다시 컴파일한 결과, 에러 메세지는 없어졌고, 문제는 해결되어 잘 작동되었다.

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "mmu.h"
#include "proc.h"
```