Operating Systems-Project01

컴퓨터소프트웨어학부

2022056262 곽주리

-Design

Cscope를 이용해서 getpid, fork의 실행 방식을 확인했다.

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

Getpid 함수는 현재 프로세스의 pid를 myproc()->pid 방식으로 받아오는 방식이었다. 이 함수를 통해 pid는 늘 int형인 것, myproc()은 포인터 형태인 것을 확인할 수 있었다.

Fork, pid의 개념이 정립되지 않은 상태라서 forktest 함수를 통해 자식 프로세스를 생성하고, 종료하고, 에러가 생기는 부분에 대해 먼저 이해하였다.

```
void
forktest(void)
{
  int n, pid;

  printf(1, "fork test\n");

  for(n=0; n<N; n++){
    pid = fork();
    if(pid < 0)
      break;
    if(pid == 0)
      exit();
  }

  if(n == N){
    printf(1, "fork claimed to work N times!\n", N);
    exit();
  }

  for(; n > 0; n--){
    if(wait() < 0){
      printf(1, "wait stopped early\n");
      exit();
    }
  }

  if(wait() != -1){
    printf(1, "wait got too many\n");
    exit();
  }

  printf(1, "fork test OK\n");
}
```

그림 1 cscope를 통해 확인한 forktest 함수

```
// caller must set state of returned proc to RUNNABLE.
int
fork(void)
{
  int i, pid;
  struct proc *np;
  struct proc *curproc = myproc();

  // Allocate process.
  if((np = allocproc()) == 0){
    return -1;
  }

  // Copy process state from proc.
  if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
    kfree(np->kstack);
    np->kstack = 0;
    np->state = UNUSED;
    return -1;
  }
  np->sz = curproc->sz;
  np->parent = curproc;
  *np->tf = *curproc->tf;

  // Clear %eax so that fork returns 0 in the child.
  np->tf->eax = 0;

  for(i = 0; i < NOFILE; i++)
    if(curproc->ofile[i])
      np->ofile[i] = filedup(curproc->ofile[i]);
  np->cwd = idup(curproc->cwd);

  safestrcpy(np->name, curproc->name, sizeof(curproc->name));

  pid = np->pid;

  acquire(&ptable.lock);

  np->state = RUNNABLE;

  release(&ptable.lock);

  return pid;
}
```

Fork를 뜯어본 결과, getpid() 함수에서 사용했던 myproc()의 자료형이 Struct proc*형이라는 것과 np->parent과 같은 방식을 통해 부모 프로세스에 접근이 가능하다는 것을 알아냈다.

혹시 몰라서, cscope를 통해 myproc()의 자료형을 한 번 더 확인하는 과정을 거쳤다.

```
104 //PAGEBREAK: 16
105 // proc.c
106 int             cpuid(void);
107 void            exit(void);
108 int             fork(void);
109 int             growproc(int);
110 int             kill(int);
111 struct cpu*     mycpu(void);
112 struct proc*    myproc();
113 void            pinit(void);
114 void            procdump(void);
115 void            scheduler(void) __attribute__((noreturn));
116 void            sched(void);
```

구현 계획:

np->parent->parent->pid 형식으로 조부모 프로세스의 pid를 얻어내는 방식으로 getgpid syscall 구현. getgpid 함수에서는 조부모 프로세스의 pid만 반환, project01.c 파일에서 getpid(), getgpid() 를 이용해서 각각 현재 프로세스의 pid, 조부모 프로세스의 pid 출력.

-Implement

```
int getgpid(void){
    //현재 실행 중인 프로세스 받아야 함
    struct proc* currentproc = myproc();
    if (currentproc) {
        cprintf("current process pid is %d\n",currentproc->pid);
        //일단 부모 프로세스가 있는 지 확인
        if(currentproc->parent){
            //조부모 프로세스까지 확인
            if(currentproc->parent->parent){
                cprintf("grand parent process pid is %d\n",currentproc->parent->parent->pid);
                return currentproc->parent->parent->pid; // 조부모 프로세스의 PID 반환
            }
            else cprintf("It doesn't exist grand parent process\n");
        }
        else cprintf("It doesn't exist parent process\n");

    }
    return -1;
}
```

myproc()을 통해 현재 실행 중인 프로세스를 받아서 currentproc에 담는다. 만약 현재 실행 중인 프로세스가 없다면 -1를 return해서 문제가 있음을 알린다. -1를 return하는 경우 에러 메시지를 출력할 수 있도록 project01.c에 처리 코드를 작성해두었다. 정상적으로 받았다면 현재 실행 중인 프로세스의 pid를 출력한다. 혹시 몰라서 디버깅용으로 작성하였다.

조부모 프로세스가 있으려면 우선 부모 프로세스가 있어야 하므로 부모 프로세스가 있는 지 확인하였다. 없다면 에러 메시지 출력.

조부모 프로세스가 있다면 조부모 프로세스의 pid를 출력하고 반환한다. 조부모 프로세스가 없다면 역시 에러 메시지를 출력한다.

=>추후에 구글링을 통해 확인한 결과, 조부모 process는 늘 존재한다는 것을 알게되었다.

```
1  #include "types.h"
2  #include "stat.h"
3  #include "user.h"
4  int main(int argc, char *argv[]){
5      printf(1,"My student id is 2022056262\n");
6      int pid,gpid;
7      pid=getpid();
8      gpid=getgpid();
9      if(gpid==-1) printf(1,"Error!");
0      printf(1,"My pid is %d\n",pid);
1      printf(1,"My gpid is %d\n",gpid);
2      exit();
```

Project01.c 코드이다.

학번 출력 후, getpid, getgpid를 통해 현재 실행 중인 프로세스와 조부모 프로세스의 pid를 받아온다. 만약 getgpid의 반환값이 -1이라면 현재 실행 중인 프로세스를 정상적으로 currentproc에 담지 못한 것이므로 에러 메시지를 출력했다.

정상적으로 getgpid()가 종료됐다면, pid와 gpid를 출력한 후 프로세스를 종료한다.

-Result

```
$ project01
My student id is 2022056262
current process pid is 3
grand parent process pid is 1
My pid is 3
My gpid is 1
$
```

Project01을 실행했을 때, 제일 먼저 학번이 출력되고 getgpid()함수가 호출되어 실행되는 것을 확인할 수 있다. 현재 프로세스를 잘 받아서, pid를 출력하였고, 부모 프로세스와 조부모 프로세스 역시 정상적으로 받아서 조부모 프로세스의 pid를 출력하였다. 함수가 종료되고 project01에서 getpid(), getgpid()의 반환값을 정상적으로 받아 출력한 것을 볼 수 있다.

-Trouble shooting

Forktest 함수는 자식 프로세스를 만들고 자식 프로세스라면, exit()하고 부모 프로세스는 모든 자식 프로세스가 종료될 때까지 기다렸다가 exit()하는 형태였다.

prac_getgpid.c + (~/xv6-publi

```c
#include "types.h"
#include "defs.h"
#include <unistd.h>
void printf(int fd,const char *s,...){
    write(fd,s,strlen(s));
}
int getgpid(char *buf)
{
    int ppid,pid;
    printf("%s\n",buf);
    ppid = fork();
    if (ppid < 0)
    {
        printf("failed to make parent process\n");
        exit();
    }
    else if (ppid == 0)
    {
        printf("Parent process\n");
        pid = fork();
        if (pid < 0)
        {
            printf("failed to make child process\n");
            exit();
        }
        else if (pid == 0)
        {
            printf("child process\n");
            printf("My pid is %d\n", getpid());
            exit();
        }
        else
        {
            wait();
            exit();
        }
    }
    else
    {
        wait();
        printf("My gpid is %d\n", getpid());
    }
    return 0;
}

```

Forktest와 유사하게 구현하면 될 것이라 생각했고

조부모 프로세스->부모 프로세스 fork->자식 프로세스 fork()->자식 프로세스 pid 출력->exit()-

>exit()->조부모 프로세스 pid 출력->exit() 순서로 구현하려 하였으나, 현재 실행 중인 프로세스의 조부모 pid가 아니라 현재 실행 중인 프로세스의 pid와, 자손 프로세스의 pid를 반환하는 함수라서 Design 파트에 작성한 방식으로 변경했다.

```c
#include "types.h"
#include "defs.h"
//조부모 프로세스의 pid를 반환하니까 int형 함수
struct proc;
int getgpid(void){
    //현재 실행 중인 프로세스 받아야 함
    struct proc* currentproc = myproc();
    if (currentproc) {
        cprintf("current process pid is %d\n",currentproc->pid);
        //일단 부모 프로세스가 있는 지 확인
        if(currentproc->parent){
            //조부모 프로세스까지 확인
            if(currentproc->parent->parent){
                cprintf("grand parent process pid is %d\n",currentproc->parent->parent->pid);
                return currentproc->parent->parent->pid; // 조부모 프로세스의 PID 반환
            }
            else cprintf("It doesn't exist grand parent process\n");
        }
        else cprintf("It doesn't exist parent process\n");

    }
    return -1;
}
int sys_getgpid(void){
    return getgpid();
}
```

```
juri@juri-VirtualBox:~/xv6-public$ make
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werro
r -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o prac_getgpid.o
 prac_getgpid.c
prac_getgpid.c: In function 'getgpid':
prac_getgpid.c:9:52: error: dereferencing pointer to incomplete type 'struct proc'
    9 |   cprintf("current process pid is %d\n",currentproc->pid);
      |                                                    ^~
make: *** [<내장>: prac_getgpid.o] 오류 1
```

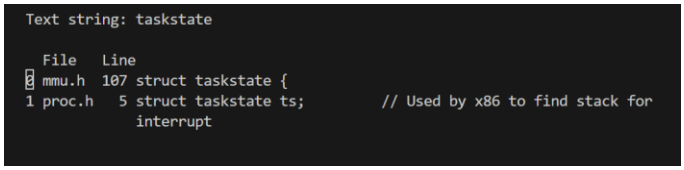
위와 같은 에러가 떠서 확인해보니, struct proc의 type이 명확하지 않아서 생기는 오류로 해당 구조체가 선언된 파일을 include해서 해결할 수 있었다.

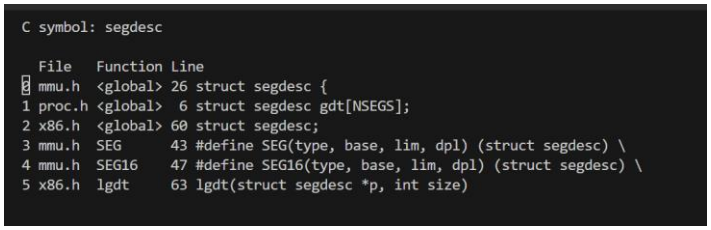
```c
3   #include "proc.h"
```

이 헤더파일을 추가하고 다시 make했더니

```
juri@juri-VirtualBox:~/xv6-public$ make
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -We
rror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o prac_get
gpid.o prac_getgpid.c
In file included from prac_getgpid.c:3:
proc.h:5:20: error: field 'ts' has incomplete type
    5 |   struct taskstate ts;         // Used by x86 to find stack for interrupt
      |                    ^~
proc.h:6:22: error: 'NSEGS' undeclared here (not in a function)
    6 |   struct segdesc gdt[NSEGS];   // x86 global descriptor table
      |                      ^~~~~
proc.h:13:24: error: 'NCPU' undeclared here (not in a function)
   13 | extern struct cpu cpus[NCPU];
      |                        ^~~~
proc.h:49:22: error: 'NOFILE' undeclared here (not in a function)
   49 |   struct file *ofile[NOFILE];  // Open files
      |                      ^~~~~~
make: *** [<내장>: prac_getgpid.o] 오류 1
```

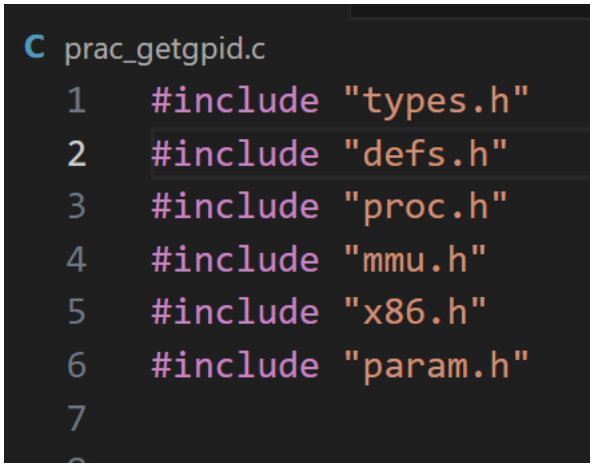
이러한 오류가 발생했다. cscope를 통해 taskstate, segdesc, cpu, file이 존재하는 헤더파일을 찾아서 getgpid 파일에 추가해주었다.



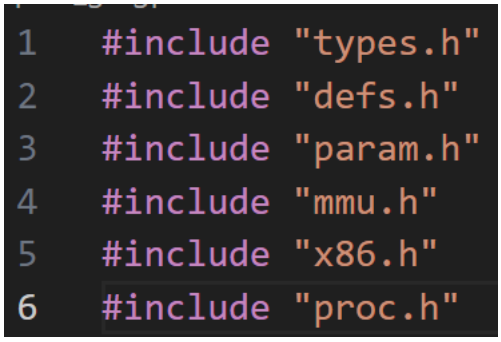


그런데도 해결되지 않아서 구글링을 통해 원인을 찾아냈다.

NOFILE, NCPU같은 상수는 대부분 param.h에 있기 때문에 param.h를 추가해주어야 한다는 것이다.



모든 헤더파일을 추가했는데도 오류가 발생해서 한참 고민에 빠졌다. 그러던 중, 실습 시간에 헤더 파일의 순서가 아주 중요하다는 조교님의 말씀이 생각이 났다. 당연히 4~6번 줄에 있는 헤더파일은 proc.h에 쓰이는 헤더파일이므로 proc.h보다 먼저 나와야 한다.



이 순서로 헤더파일의 순서를 바꿔줬더니 에러가 해결되었다.