Project01

컴퓨터소프트웨어학부 2020055350 강현중

1. Design

1) getpid 분석

```
Cscope 태그: getpid
   #   줄  파일 이름 / 콘텍스트 / 줄
   1    92  syscall.c <<<unknown>>>
            extern int sys_getpid(void);
   2   120  syscall.c <<<unknown>>>
            [SYS_getpid]  sys_getpid,
   3    12  syscall.h <<<unknown>>>
            #define SYS_getpid 11
   4    40  sysproc.c <<<unknown>>>
            sys_getpid(void)
   5    22  user.h <<<unknown>>>
            int getpid(void);
```

cscope 분석결과

```
39 int
40 sys_getpid(void)
41 {
42   return myproc()->pid;
43 }
44
```

sysproc.c

cscope로 getpid를 찾아본 결과 sysproc.c 안에 정의되어 있었다.
system call getpid는 myproc()의 pid를 return 하고 있다.

```
struct proc*
myproc(void) {
  struct cpu *c;
  struct proc *p;
  pushcli();
  c = mycpu();
  p = c->proc;
  popcli();
  return p;
}
```

proc.c

myproc()는 자신의 프로세스 struct의 포인터를 반환한다.

```
38 struct proc {
39   uint sz;                     // Size of process memory (bytes)
40   pde_t* pgdir;                // Page table
41   char *kstack;                // Bottom of kernel stack for this process
42   enum procstate state;        // Process state
43   int pid;                     // Process ID
44   struct proc *parent;         // Parent process
45   struct trapframe *tf;        // Trap frame for current syscall
46   struct context *context;     // swtch() here to run process
47   void *chan;                  // If non-zero, sleeping on chan
48   int killed;                  // If non-zero, have been killed
49   struct file *ofile[NOFILE];  // Open files
50   struct inode *cwd;           // Current directory
51   char name[16];               // Process name (debugging)
52 };
```

proc.h

proc.h에서 proc의 구조를 살펴본 결과 parent를 통해 부모 프로세스에 접근할 수 있다.
명세에서는 조부모 프로세스의 id를 출력하는 것이 목적이므로 getgpid() system call은 myproc()->parent->parent->pid를 반환하도록 구현할 것이다.

2. Implement

1)getgpid system call 구현

```
1 #include "types.h"
2 #include "x86.h"
3 #include "defs.h"
4 #include "param.h"
5 #include "mmu.h"
6 #include "proc.h"
7 int
8 sys_getgpid(void)
9 {
10         return myproc()->parent->parent->pid;
11 }
12
```

getgpid.c 파일을 만들어 getgpid system call을 구현했다. getgpid는 myproc()의 조부모의 접근하여 pid를 반환한다.

2)새로운 system call 등록

```
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_myfunction(void);
extern int sys_getgpid(void);
```

```
130 [SYS_close]   sys_close,
131 [SYS_myfunction]   sys_myfunction,
132 [SYS_getgpid]   sys_getgpid,
133 };
```

```
20 #define SYS_link   19
21 #define SYS_mkdir  20
22 #define SYS_close  21
23 #define SYS_myfunction 22
24 #define SYS_getgpid 23
```

syscall.c　　syscall.h

syscall.c에서는 *syscalls[] 배열 안에 sys_getgpid를 등록하였고, syscall.h에서는 SYS_getgpid의 index를 등록했다.

```
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 int myfunction(char*);
27 int getgpid(void);
```

user.h

user.h에 user program에서 사용할 getgpid()를 등록한다.

```
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(myfunction)
33 SYSCALL(getgpid)
```

usys.S

usys.S에 getgpid를 매크로로 등록하여 유저프로그램에서 사용할 수 있게 한다.

3)유저 프로그램 project01.c 작성

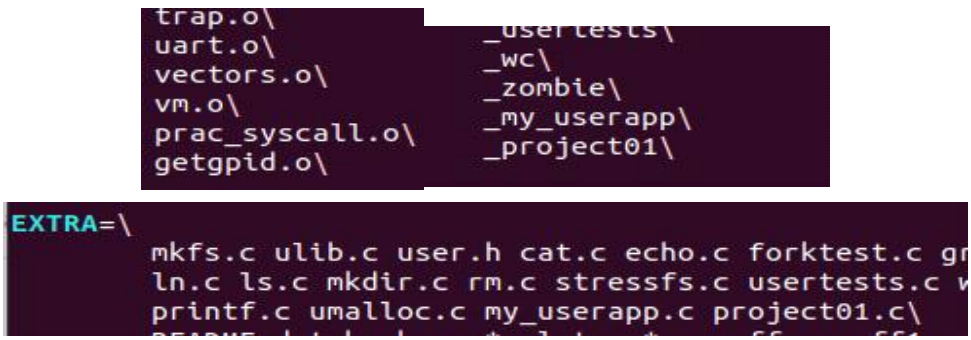
```c
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[])
{
printf(1,"My student id is 2020055350\n");
printf(1, "My pid is: %d\n" , getpid());
printf(1, "My gpid is: %d\n" , getgpid());
exit();
}
```

project01.c

과제의 명세에 맞게 학번을 출력하도록 작성하였으며, 기존 system call 인 getpid, 새로 만든 system call인 getgpid를 사용해 과제의 명세에 맞게 작동하는 프로그램을 작성했다.

4)Makefile 수정



Makefile

getgpid.c의 컴파일과 xv6 실행파일을 만들기 위한 정보를 Makefile 안에 입력했다.

3. Result

1) 컴파일 및 실행 과정

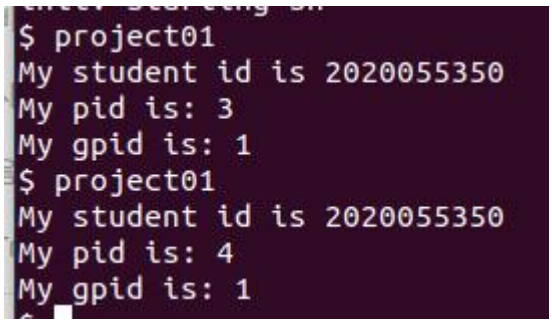
```
make clean
make
make fs.img
./bootxv6.sh

project01
```

컴파일 및 실행 과정

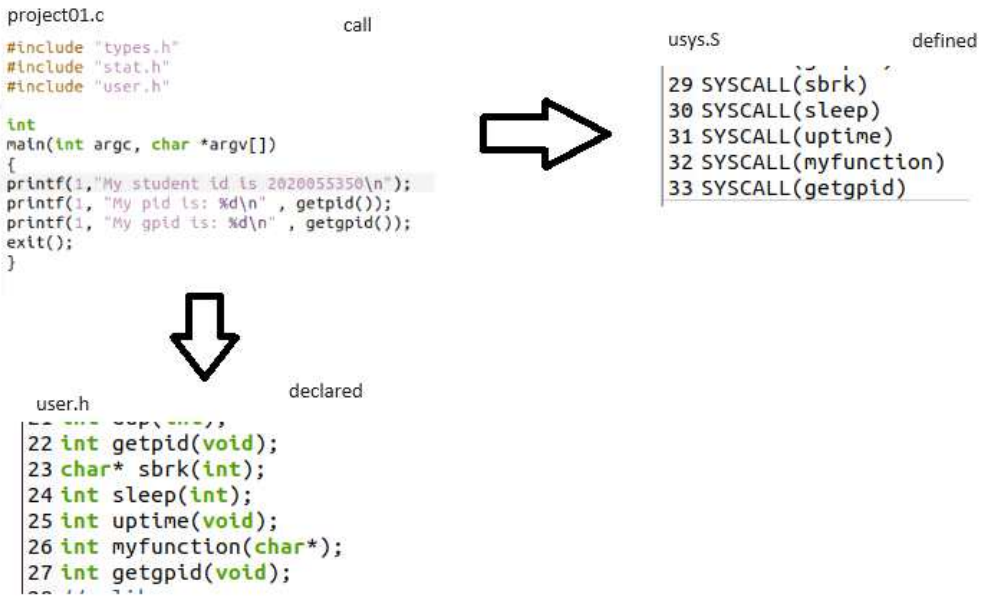
xv6-public 폴더에서 make 와 make fs.img 로 필요한 파일을 컴파일하고 xv6 실행에 필요한 파일을 만든다.

이후 ./bootxv6.sh를 통해 xv6를 실행하고 project01을 실행해 명세에서 요구한 대로 프로그램이 작동하는지 확인한다.



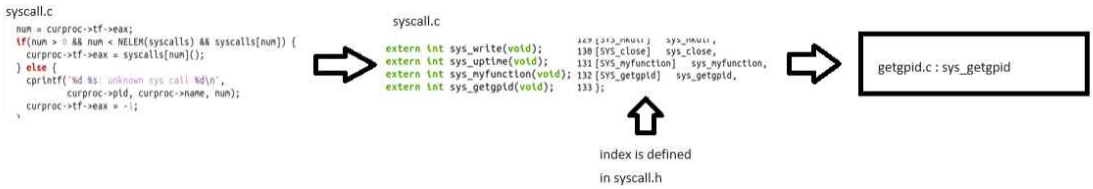
실행 결과

실행 결과 명세에서 요구한 대로 학번,pid, 조부모 프로세스의 pid가 출력 되는 것을 확인했다.



call getgpid system call

project01에서 getgpid()를 사용하면 user.h에 선언된 함수와 usys.S에 정의된 매크로에 의해 getgpid system call 을 호출하게 된다.



kernel에서의 동작과정

system call을 호출하면 커널에서는 syscall.h에 정의된 index에 따라 syscall.c에서 그에 맞는 시스템 call을 실행한다. 이번 프로젝트에서는 getgpid.c의 sys_getgpid를 실행하게 되고, sys_gpid는 조부모의 pid를 유저프로그램으로 반환한다.

4. Trouble shooting

기존 system call인 getpid를 cscope로 찾아봤을 때 paramiter가 없는 system call이어서 sys_getpid()가 warpper function 없이 그 자체로 수행하는 것으로 추측 됐고, defs.h에도 선언되어 있지 않았다. 다른 파일에서도 getpid()라는 함수가 따로 정의되어 있지 않아, 같은 역할인 sys_getgpid도 같은 방법으로 구현하였다.

컴파일 과정에서 Makefile을 만든 후 make를 했을 때, 실패해서 실습 시간에 진행했던 UPROGS 부분에 _my_userapp과 _project01을 순서를 바꿨더니 정상적으로 작동하였고, 다시 원래대로 순서를 바꿨는데도 정상적으로 작동했다. 처음에 make를 실패한 원인은 아직 파악하지 못했다.