



Project1 [getgpuid() 구현]

1. Design

먼저 `getgpuid()`에 대해 알아보도록 합니다. `getgpuid()`란, grand parent의 process id 즉 현재 process(자손)를 기준으로 `할머니/할아버지`의 process id를 불러오는 system call을 요청하는 함수입니다. 저번 실습 시간에 직접 `prac_syscall.c`를 xv6 커널에 등록해서 system call을 호출하는 과정을 배웠습니다.

따라서 실습시간에 배운 방식을 응용하여 xv6 커널에 `getgpuid()`를 호출하는 system call을 등록하여 xv6를 부팅하고 `getgpuid()`를 호출했을 때 원하는 내용을 출력하도록 구현을 할 것입니다.

Idea

작년 2학기 “시스템프로그래밍” 수업 때 process에 대해 배우며 `fork()`를 통해 프로세스를 생성하고 `getpid` 함수를 통해 pid 값을 가져오는 내용에 대해 배웠던 기억이 있어서 2번의 `fork`로 process를 생성하고 그로 인해 생성된 process들을 통해 pid 값을 불러오는 방향으로 접근을 해보려고 하였습니다.

우선 `user.h` 파일을 확인해보면 pid를 불러오는 함수인 `getpid(void)` 와 process를 생성하는 함수인 `fork()` system call 함수가 이미 구현되어 xv6에 등록이 되어있는 것을 확인할 수 있습니다. `process`란, “프로그램에 대한 모든 정보를 담고 있는 자료구조” 라고 배웠기 때문에 process의 생성을 담당하는 `fork()` 함수에 구조체와 포인터, 함수 등 프로세스를 생성하기 위해 필요한 여러 기능들이 포함되어 있는 것이 당연합니다. 따라서 `getgpuid()` 함수를 구현하는 것 보다 우선시 되어야 하는 것은 기존에 구현되어 있는 함수들이 어떠한 방식으로 프로세스를 생성하여 pid를 불러오는지에 대해 이해하고 이를 바탕으로 조부모 프로세스의 pid를 어떻게 불러와야할지 생각을 해보아야 합니다.

```

C syscall.h M  C user.h M X  ASM usys.S M  M Makefile M
C user.h > ...
1 struct stat;
2 struct rtdat;
3
4 // system calls
5 int fork(void);
6 int exit(void) __attribute__((noreturn));
7 int wait(void);
8 int pipe(int*);
9 int write(int, const void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
18 int link(const char*, const char*);
19 int mkdir(const char*);
20 int chdir(const char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 int myfunction(char*);

```

```

C proc.c > ...
180 int
181 fork(void)
182 {
183     int i, pid;
184     struct proc *np;
185     struct proc *curproc = myproc();
186
187     // Allocate process.
188     if((np = allocproc()) == 0){
189         return -1;
190     }
191
192     // Copy process state from curproc.
193     if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
194         kfree(np->kstack);
195         np->kstack = 0;
196         np->state = UNUSED;
197         return -1;
198     }
199     np->sz = curproc->sz;
200     np->parent = curproc;
201     *np->tf = *curproc->tf;
202
203     // Clear %eax so that fork returns 0 in the child.
204     np->tf->eax = 0;
205
206     for(i = 0; i < NOFILE; i++)
207         if(curproc->ofile[i])
208             np->ofile[i] = filedup(curproc->ofile[i]);
209     np->cwd = idup(curproc->cwd);
210
211     safestrcpy(np->name, curproc->name, sizeof(curproc->name));
212
213     pid = np->pid;
214
215     acquire(&table.lock);
216
217     np->state = RUNNABLE;
218
219     release(&table.lock);
220
221     return pid;
222 }

```

이번 과제가 process에 대한 system call을 구현하여 호출하는 과제이기 때문에 process의 생성/삭제 등과 관련된 코드들을 이해하는게 중요하다고 생각했습니다. 따라서 가장 먼저 process와 관련 있는 파일인 **proc.c**와 **proc.h** 파일을 살펴보았습니다. 확인한 결과 이미 **proc.h** 파일의 proc 구조체에 **pid**, **parent process** 등이 선언되어 있고 이 구조체를 **proc.c** 파일에서 **myproc()**으로 process에 대한 내용이 구현되어 있는 것을 확인할 수 있었습니다.

```

C Project01.c u  C proc.h X  C proc.c M  M Makefile M  C gpid_syscall.c u
C proc.h > ...
33 };
34
35 enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
36
37 // Per-process state
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; // switch() 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 };

```

```

C Project01.c u  C proc.c M X  M Makefile M  C gpid_syscall.c u
C proc.c > ...
54
55 // Disable interrupts so that we are not rescheduled
56 // while reading proc from the cpu structure
57 struct proc*
58 myproc(void) {
59     struct cpu *c;
60     struct proc *p;
61     pushcli();
62     c = mycpu();
63     p = c->proc;
64     popcli();
65     return p;
66 }

```

이를 통해 **myproc()**안에 **proc** 구조체를 포인터로 선언하고 있다는 것을 확인하였고, **getgpid()**를 구현할 때 **fork()**와 **getpid()**를 이용해서 구현하는 것 보다는 **myproc()**를 사용해서 process에 대한 정보를 가져오는 방법이 더 효율적인 방법이라고 생각해서 **fork()** 대신 **myproc()**을 사용해서 구현을 하면 되겠다고 생각을 했습니다.

2. Implement

Programming

가장 먼저 **project01.c** 파일을 구현하였습니다. 이 파일의 경우 이번 프로젝트에서 흔히 **main.c**와 같은 역할을 하는 파일이고 xv6를 실행하였을 때 명세에 주어진 **학번**, **pid**, **gpid**를 출력하는 함수이며 code는 다음과 같습니다.

학번도 다른 변수처럼 선언을 해야하는지 고민을 했지만, 학번은 xv6와 아무런 연관점이 없기 때문에 그냥 넣으면 될 것 같아서 바로

printf()에 넣었습니다.

```

/* project01.c */
#include "types.h"

```

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

int
main(int argc, char* argv[])
{
    printf(1, "My student id is 2021088304\n");
    printf(1, "My pid is %d\n", getpid());
    printf(1, "My gpid is %d\n", getgpid());
    exit();
}
// 0: stdin, 1: stdout, 2: stderr
```

그 다음, **getpid()**는 이미 구현되어 있기 때문에 구현할 필요 없이 바로 **gpid_syscall.c** 파일을 만들어서 이 함수에 gpid 값을 가져오는 **getgpid()**를 구현하였습니다. 구현한 code는 다음과 같습니다.

```
/* getgpid.c */
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "x86.h"
#include "proc.h"
#include "spinlock.h"

int
getgpid(void)
{
    struct proc *curproc = myproc();
    if (curproc && curproc->parent) {
        struct proc *parent = curproc->parent;
        if (parent->parent) {
            return parent->parent->pid;
        } else {
            return -1;
        }
    }
    return -1;
}

int
sys_getgpid(void)
{
    int gpid = getgpid();
    if(gpid<0)
        exit();
    return gpid;
}
```

getgpid 설명

1. **proc.c** 파일에 있는 **myproc()**을 구조체 포인터로 받아와서 현재 process에 대한 정보를 불러옵니다.

```
C Project01.c u C proch x C proc.c M Makefile M C gpid_syscall.c u
C proch > ...
33 };
34
35 enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
36
37 // Per-process state
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; // switch() 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 };
```

(위에서도 설명했지만, `proc` 구조체에는 `process`에 대한 정보가 들어 있습니다.)

- 프로세스가 현재 `process` 이거나 부모 `process`가 존재하는 경우 `if`문 안으로 들어가고 그렇지 않는 경우 `-1`을 `return` 합니다.
- `if`문 안으로 들어간 경우 부모 `process`가 존재한다는 의미이기 때문에 부모 `process`에 대한 정보를 아까 현재 `process`에 대해서 했던 것 처럼 포인터로 불러옵니다.
- 부모 `process`의 부모 `process` (조부모 `process`)가 존재하는지 확인합니다. 이 때도 만약 조부모 `process`가 존재하지 않는다면 `getgpid` 함수의 목적과 부합하지 않기 때문에 `if`문을 빠져나와 `-1`을 `return` 합니다.
- 조부모 `process`가 존재한다면 **parent→parent→pid** 로 조부모 `process`의 `pid` 값을 `return` 합니다.
- 만약 **getgpid()**를 통해 `-1`이 `return`된 경우 부모 또는 부모의 부모 프로세스가 존재하지 않는 경우이기 때문에 `wrapper function`을 통해 **exit()** `system call`을 호출해서 종료하고 그렇지 않은 경우 그대로 `gpid` 값을 `return` 합니다.

이렇게 되면 이 함수를 통해 얻고자 하는 조부모 `process`의 `pid` 값을 얻을 수 있게 됩니다.

Add System Call

code 구현까지 끝났다면, `xv6`에 이 파일들을 등록하는 과정이 필요합니다.가장 먼저 `.o` 파일을 `Makefile`의 `OBJS` 부분에 입력해서 `Make` 명령어로 `build` 작업을 수행할 때 `gpid_syscall.c` 파일도 `build` 할 수 있도록 해야 합니다.

```
M Makefile
1  OBJS = \
2      bio.o\
3      console.o\
4      exec.o\
5      file.o\
6      fs.o\
7      ide.o\
8      ioapic.o\
9      kalloc.o\
10     kbd.o\
11     lapic.o\
12     log.o\
13     main.o\
14     mp.o\
15     picirq.o\
16     pipe.o\
17     proc.o\
18     sleeplock.o\
19     spinlock.o\
20     string.o\
21     swtch.o\
22     syscall.o\
23     sysfile.o\
24     sysproc.o\
25     trapasm.o\
26     trap.o\
27     uart.o\
28     vectors.o\
29     vm.o\
30     prac_syscall.o\
31     gpid_syscall.o\
```

잘 등록되었는지 확인하고 싶다면 “make clean”, “make | grep gpid_syscall” 명령어로 build를 해서 terminal에 **gpid_syscall.o** 가 출력되는지 확인해보면 됩니다.

다음으로 **def.h** 파일에 **gpid_syscall.c** 파일에서 정의한 **getgpid()**를 적어 다른 c파일에서도 사용할 수 있도록 정의해야 합니다.

```
C defs.h > ...
175 void      kvmalloc(void);
176 pde_t*     setupkvm(void);
177 char*      uva2ka(pde_t*, char*);
178 int        allocvm(pde_t*, uint, uint);
179 int        deallocvm(pde_t*, uint, uint);
180 void       freevm(pde_t*);
181 void       initvm(pde_t*, char*, uint);
182 int        loadvm(pde_t*, char*, struct inode*, uint, uint);
183 pde_t*     copyvm(pde_t*, uint);
184 void       switchvm(struct proc*);
185 void       switchkvm(void);
186 int        copyout(pde_t*, uint, void*, uint);
187 void       clearpteu(pde_t *pgdir, char *uva);
188 // prac_syscall.c
189 int        myfunction(char*);
190 // gpid_syscall.c
191 int        getgpid(void);
192
```

defs.h 파일에 입력하는 과정까지 끝났다면 다시 **gpid_syscall.c** 파일로 돌아가서 wrapper function을 입력해야 합니다. Wrapper function을 통해 trap frame에서 매개변수를 가져올 수 있습니다.

이제 system call을 새로 등록하는 과정이 필요합니다.

syscall.h 파일을 열어서 **SYS_getgpid 23** 을 입력합니다. 이 말인 즉슨 **getgpid()**를 system call 중에서 23번 번호를 부여하겠다는 의미입니다.그 다음 **syscall.c** 파일에도 **extern int sys_getgpid(void);** 와 **[SYS_getgpid]** **sys_getgpid,** 를 입력합니다.

```

18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 #define SYS_myfunction 22
24 #define SYS_getpid 23

```

```

129 [SYS_mkdir] sys_mkdir,
130 [SYS_close] sys_close,
131 [SYS_myfunction] sys_myfunction,
132 [SYS_getpid] sys_getpid,
133 ];
134
135 void
136 syscall(void)

```

```

103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_myfunction(void);
107 extern int sys_getpid(void);
108
109 static int (*syscalls[])(void) = {
110 [SYS_fork] sys_fork,
111 [SYS_exit] sys_exit,
112 [SYS_wait] sys_wait,

```

마지막으로 **user.h**와 **usys.S** 파일에 입력해서 user program에서 새로 등록한 함수들을 사용할 수 있도록 해주면 system call을 xv6에 등록하는 과정이 끝납니다.

user.h에는 **int getpid(void);**를 입력하고 **usys.S**에는 **SYSCALL(getpid)**를 입력하면 됩니다.

```

C user.h > getpid(void)
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
18 int link(const char*, const char*);
19 int mkdir(const char*);
20 int chdir(const char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 int myfunction(char*);
27 int getpid(void);
28

```

```

25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(myfunction)
33 SYSCALL(getpid)

```

Makefile에 main 함수가 있는 **Project01.c**를 추가하고 **EXTRA=** 부분에 **Projectt01.c** 를 입력하면 빌드 및 xv6 부팅에 대한 모든 준비는 끝납니다.

```

170
171 UPROGS=\
172 _cat\
173 _echo\
174 _forktest\
175 _grep\
176 _init\
177 _kill\
178 _ln\
179 _ls\
180 _mkdir\
181 _rm\
182 _sh\
183 _stressfs\
184 _usertests\
185 _wc\
186 _zombie\
187 _my_userapp\
188 _project01\

```

```

249 # CUT HERE
250 # prepare dist for students
251 # after running make dist, probably want to
252 # rename it to rev0 or rev1 or so on and then
253 # check in that version.
254
255 EXTRA=\
256 mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
257 ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
258 printf.c umalloc.c my_userapp.c project01.c\
259 README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
260 .gdbinit.tmpl gdbutil\
261
262 dist:
263 rm -rf dist
264 mkdir dist
265 for i in $(FILES); \
266 do \
267 | grep -v PAGEBREAK $$i >dist/$$i; \

```

3. Result

xv6 Booting

a) make clean


```

hyunjoon@hyunjoon-virtual-machine:~/바탕화면/xv6-public$ make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*.o *.d *.asm *.sym vectors.S bootblock entryother \
initcode initcode.out kernel xv6.img fs.img kernelmemfs \
xv6memfs.img mkfs .gdbinit \
_cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _wc _zombie _my_userapp _Project01

```

b) make

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

objdump -S initcode.o > initcode.asm
ld -m elf_i386 -T kernel.ld -o kernel entry.o bio.o console.o exec.o file.o fs.o ide.o ioapic.o kalloc.o kbd.o lapic.o log.o main.o mp.o picirq.o pipe.o proc.o sleeplock.o spinlock.o string.o switch.o syscall.o sysfile.o sysproc.o
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
10000+0 레코드 들어옴
10000+0 레코드 나감
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.118395 s, 43.2 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 레코드 들어옴
1+0 레코드 나감
512 bytes copied, 0.0017338 s, 295 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
423+1 레코드 들어옴
423+1 레코드 나감
216776 bytes (217 kB, 212 KiB) copied, 0.00507957 s, 42.7 MB/s
hyunjoon@hyunjoon-virtual-machine:~/바탕화면/xv6-public$

```

c) make fs.img

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o zombie.o zombie.c
ld -m elf_i386 -N -e main -Ttext 0 -o _zombie zombie.o ulib.o usys.o printf.o umalloc.o
objdump -S _zombie > zombie.asm
objdump -t _zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > zombie.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o my_userapp.o my_userapp.c
ld -m elf_i386 -N -e main -Ttext 0 -o _my_userapp my_userapp.o ulib.o usys.o printf.o umalloc.o
objdump -S _my_userapp > my_userapp.asm
objdump -t _my_userapp | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > my_userapp.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o project01.o project01.c
ld -m elf_i386 -N -e main -Ttext 0 -o _project01 project01.o ulib.o usys.o printf.o umalloc.o
objdump -S _project01 > project01.asm
objdump -t _project01 | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > project01.sym
./mkfs fs.img README _cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _wc _zombie _my_userapp _project01
rmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
ballocc: first 729 blocks have been allocated
ballocc: write bitmap block at sector 58
hyunjoon@hyunjoon-virtual-machine:~/바탕화면/xv6-public$

```

d) qemu-system-i386 -nographic -serial mon:stdio -hdb fs.img xv6.img -smp 1 -m 512

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

SeaBIOS (version 1.13.0-lubuntu1.1)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00

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
$

```

이 화면이 뜨면 xv6가 부팅이 되었다는 의미입니다. 이제 \$ 뒤에 아까 만든 **project01**을 입력하면 project01에 있는 code (**학번**, **pid**, **gpid**에 대한 정보 출력)가 terminal에 출력되게 됩니다.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

SeaBIOS (version 1.13.0-1ubuntu1.1)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00

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 2021088304
My pid is 3
My gpid is 1
$
```

이렇게 출력이 되면 구현이 끝납니다.
참고로 booting 중인 xv6를 종료하고 싶다면 **ctrl+A, X** 를 입력하면 됩니다.

4. Troubleshooting

```
PROBLEMS 13 OUTPUT DEBUG CONSOLE TERMINAL PORTS

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00

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 2021088304
My pid is 3
$
```

```
/* project01.c */
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char* argv[])
{
    printf(1, "My student id is 2021088304\n");
    printf(1, "My pid is %d\n", getpid());
    // printf(1, "My gpid is %d\n", getgpid());
    exit();
}
// 0: stdin, 1: stdout, 2: stderr
```

getgpid() 를 구현하기 전에 gpid를 출력하는 부분을 주석 처리 한 상태로 xv6에 등록해서 부팅 후 출력을 해보았습니다. 잘 출력되는 것을 확인할 수 있습니다.

이젠 구현만 하면 될 것 같습니다.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

./vectors.pl > vectors.S
gcc -m32 -gdwarf-2 -Wa,-divide -c -o vectors.o vectors.S
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o vm.o vm.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o prac_syscall.oc
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o gpid_syscall.oc
In file included from gpid_syscall.c:4:
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
      |
make: *** [<내장>: gpid_syscall.o] 오류 1
hyunjoon@hyunjoon-virtual-machine:~/바탕화면/xv6-public$
* History restored
```

gpid_syscall.c 파일에 대략적인 코드를 작성 후 빌드를 해보았는데, 이전에는 잘 인식되던 변수들이 새파일을 추가하면서 인식이 안되는 문제가 생겼습니다. 먼저 이 문제를 해결해야 할 것 같습니다.

```
18 // Wrapper for my_syscall
19 int
20 sys_getpid(void)
21 {
22     return getpid();
23 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror
In file included from gpid_syscall.c:2:
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: 'NCPUs' undeclared here (not in a function)
   13 | extern struct cpu cpus[NCPUs];
      |
proc.h:49:22: error: 'NOFILE' undeclared here (not in a function)
   49 | struct file *ofile[NOFILE]; // Open files
      |
make: *** [<내장>: gpid_syscall.o] 오류 1
```

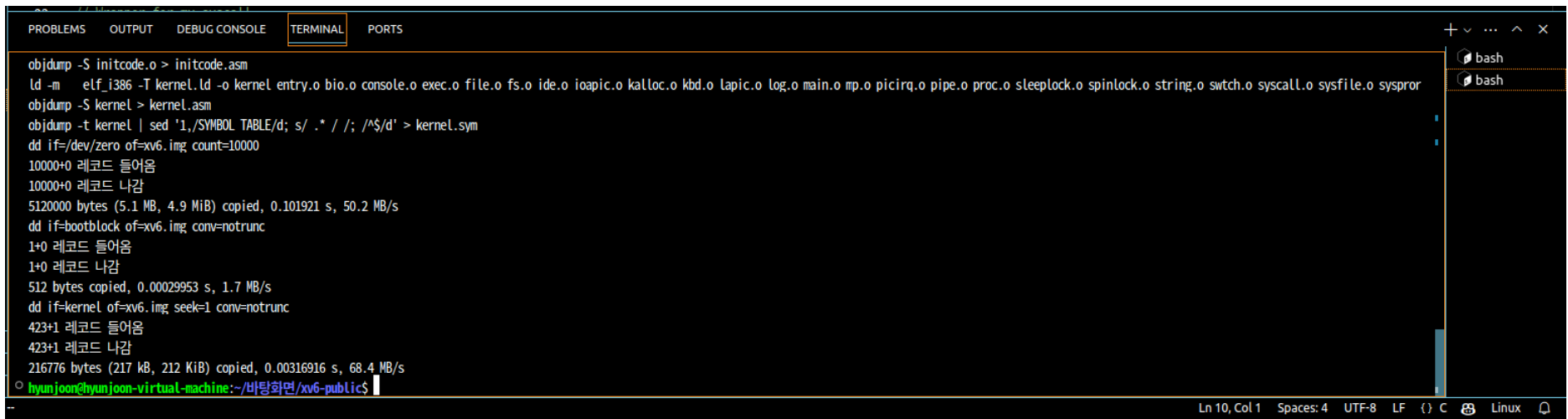
gpid_syscall.c 파일에 wrapper function을 구현하지 않아서 이게 원인일까 싶어 다시 빌드를 시도해보았지만 오히려 더 많은 변수들을 인식하지 못하는 현상이 생겼습니다. wrapper 함수에 error handling 처리를 하지 않아서 그런가 생각을 해보았지만 이 역시 문제가 아니었습니다. error message를 보면 구조체와 Macro 변수를 인식하지 못하는 것을 알 수 있습니다. 따라서 이 변수들이 정의된 파일을 include 하면 해결 될 것이라고 생각했습니다.

```
/* proc.c */
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "x86.h"
#include "proc.h"
#include "spinlock.h"
```

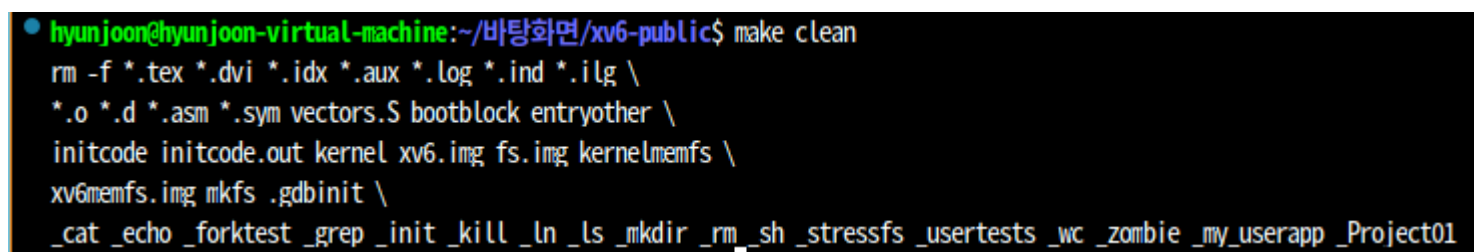
위의 문제의 경우 header 파일 참조로 인한 문제라고 생각했기 때문에 올바른 파일들을 참 해야 해결이 됩니다. proc.c에 있는 header 파일에 대한 참조를 복붙하니 위의 문제가 해결되었습니다. 그 이유를 파악해보니 여러 파일들끼리 변수들을 참조

하고 있어서 header 파일을 하나만 빼더라도 변수를 불러오지 못하는 문제가 생기는 것 같습니다.

process에 관련된 system call을 구현하는 함수이기 때문에 proc.c에 참조중인 파일들 전부를 참조해서 해결하려고 시도해 보았습니다.



이 방식으로 수정하니 빌드가 잘 되는 모습을 확인할 수 있었습니다.



```
int sys_getgid(void)
{
    return getgid();
}

-----
int sys_getgid(void)
{
    int gid = getgid();
    if (gid < 0)
        exit();
    return gid;
}
```

원래는 **getgid()**를 return 하는 코드만 작성했었는데, **getgid()**에서 조부모의 pid를 return할 수 없는 경우 -1을 return하도록 구현을 했기 때문에 -1이 return 되었을 경우 wrapper 함수로 프로그램을 종료하는 system call을 호출하도록 해주어야 할 것 같아서 **exit()** system call을 호출하도록 구현하였습니다.