

2022083436 주예지

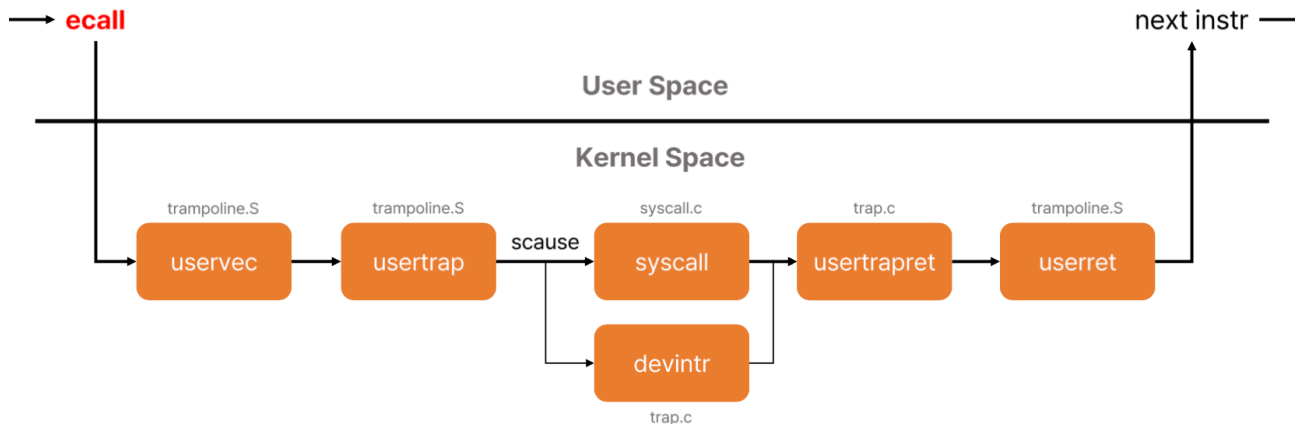
운영 체제 Assignment 1 Wiki

Outline:

- **Design:** introduce the purpose and the goal of this assignment
- **Implementation:** explain the methodologies and procedures
- **Results:** the expected output and the actual compilation result
- **Trouble-Shooting:** problems I met throughout the assignment

I. Design

In this assingment, I will implement the getppid() system call which returns the parent's process ID in vx6-riscv. Because there's already a system call getpid() (which returns the process ID, PID) existing in the system, what I have to do is to add a similar function in the kernel side. The purpose of the assignment was to successfully print the student's ID, the process ID PID, and the parent's process ID PPID at the end.



According to Lab 03's explanation, I need to enter the kernel side and implement the getppid() system call function. Also, I have to find a method that exposes the system call to user space programs.

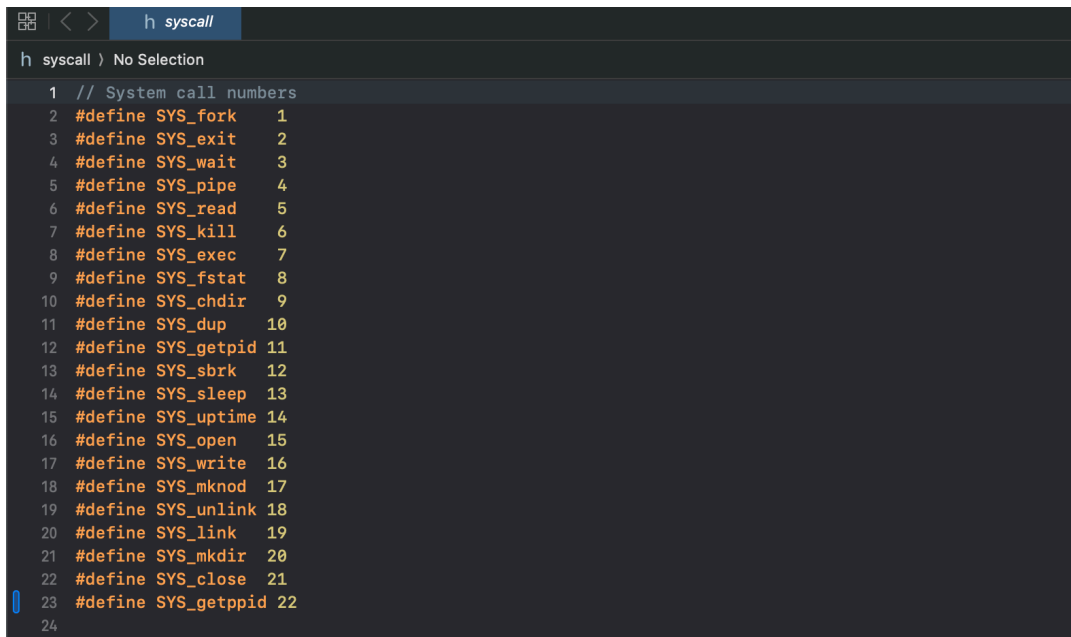
II. Implementation

Through lab03, I learned the 5 steps of kernel implementation and 2 steps of user program. In the kernel, I have to

- Implement a function
- Add the source file in Makefile

- Declare the function in defs.h
- Implement a wrapper function to call the function
- Register the new system call in syscall.h and syscall.c

First, I defined the **getppid()** system call in **syscall.h**. Underneath the already-built system calls (including **getpid()**), I assigned **getppid()** and assigned an unused syscall number which is 22.

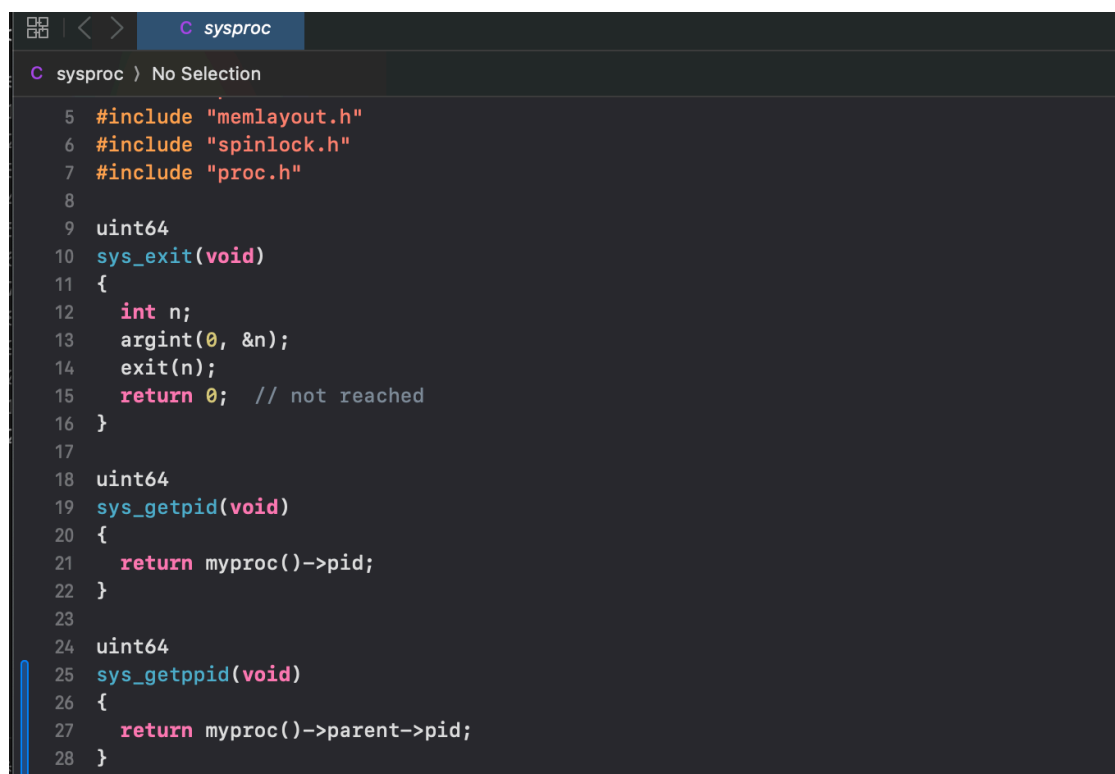


```

1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_wait 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_kill 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define SYS_sleep 13
15 #define SYS_uptime 14
16 #define SYS_open 15
17 #define SYS_write 16
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_getppid 22
24

```

Then, I edited the **sysproc.c** and implemented the wrapper function **sys_getpid(void)**, which will return the parent's process ID. I made the function to return the parent for **myproc()**. I made my function **getppid()** to get parameters from trapframe through wrapper function.



```

5 #include "memlayout.h"
6 #include "spinlock.h"
7 #include "proc.h"
8
9 uint64
10 sys_exit(void)
11 {
12     int n;
13     argint(0, &n);
14     exit(n);
15     return 0; // not reached
16 }
17
18 uint64
19 sys_getpid(void)
20 {
21     return myproc()->pid;
22 }
23
24 uint64
25 sys_getppid(void)
26 {
27     return myproc()->parent->pid;
28 }
29

```

Then, I access **syscall.c** to include `sys_getppid`. I registered this wrapper function as system call.

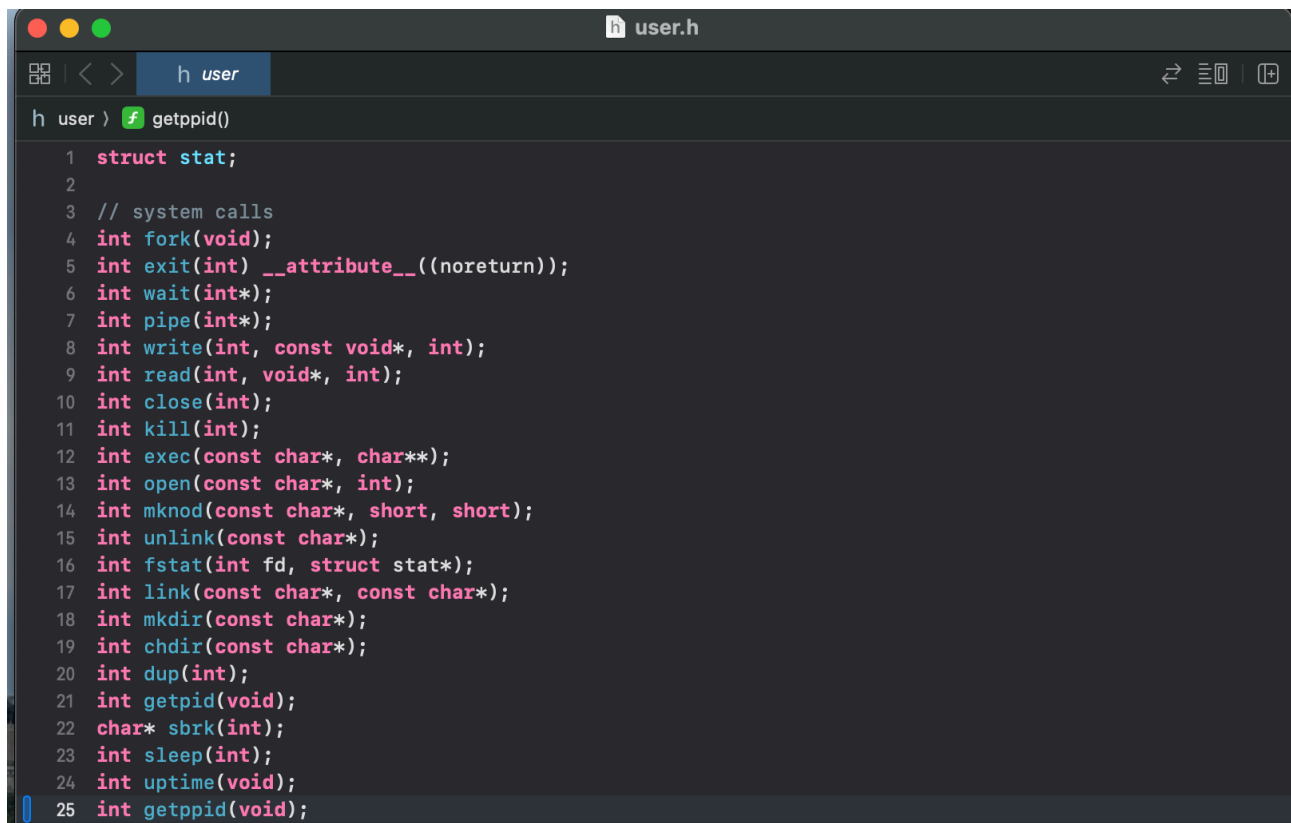
```

102 extern uint64 sys_mkdir(void);
103 extern uint64 sys_close(void);
104 extern uint64 sys_getppid(void);
127 [SYS_link] sys_link,
128 [SYS_mkdir] sys_mkdir,
129 [SYS_close] sys_close,
130 [SYS_getppid] sys_getppid,
131

```

Now the kernel work is done, I will expose this **getppid()** to user programs. What I have to do is :

- Declare the function in **user.h**
- Add a new macro in **usys.pl**



```

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

```

I declared the **getppid(void)** function in `user.h` header file. And then, I added an entry of `getppid` in **usys.pl** file, which is to generate **usys.S** that provides a wrapper for the system call. We can see that the **usys.S** automatically generates `getppid` wrapper function.

```

37 entry("sleep");
38 entry("uptime");
39 entry("getppid");
109 getppid:
110     li a7, SYS_getppid
111     ecall
112     ret

```

Furthermore, I created a **ppid.c** file in user side.



```

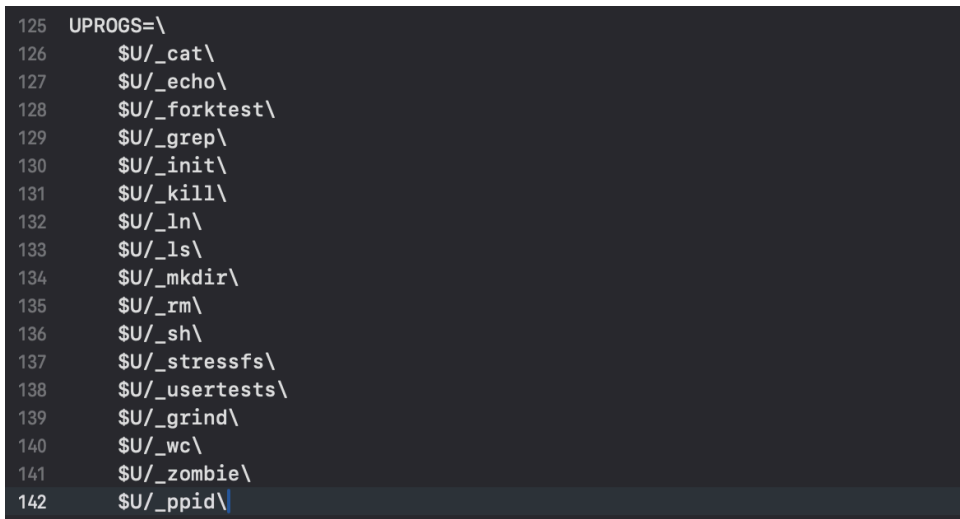
1  #include "kernel/types.h"
2  #include "kernel/stat.h"
3  #include "user/user.h"
4
5  int main() {
6      printf("My student ID is 2022083436\n");
7      printf("My pid is %d\n", getpid());
8      printf("My ppid is %d\n", getppid());
9      exit(0);
10 }
11

```

In **kernel/types.h**, it contains **uint64**, **int**, **char** that are required for data types. In **kernel/stat.h**, it provides the structures and constants related to file system operations. **User/user.h** is included for user-space system call definitions, including **getpid()**, **getppid()**, and **printf()**.

I returned my student ID which is 2022083436, and the pid and ppid.

Last but not least, I updated the makefile to include ppid.

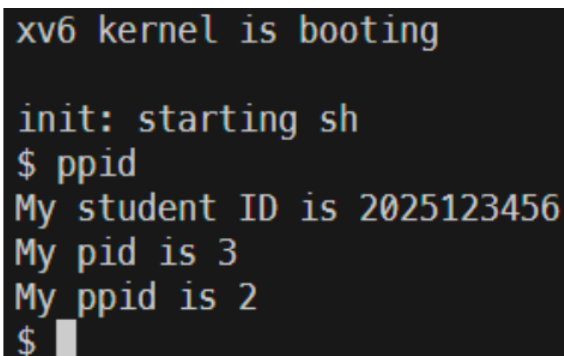


```

125 UPROGS=\
126     $U/_cat\
127     $U/_echo\
128     $U/_forktest\
129     $U/_grep\
130     $U/_init\
131     $U/_kill\
132     $U/_ln\
133     $U/_ls\
134     $U/_mkdir\
135     $U/_rm\
136     $U/_sh\
137     $U/_stressfs\
138     $U/_usertests\
139     $U/_grind\
140     $U/_wc\
141     $U/_zombie\
142     $U/_ppid\

```

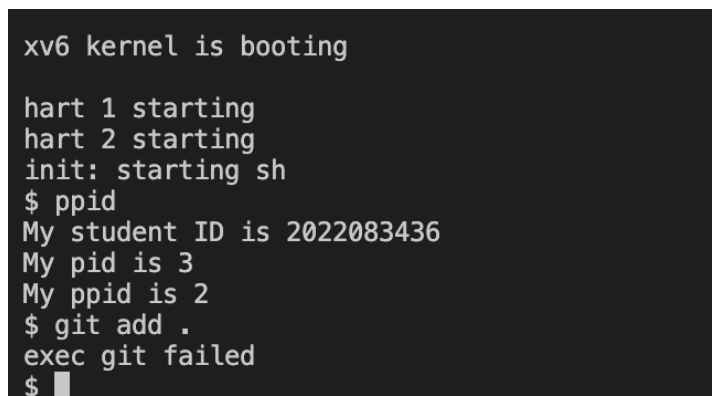
III. Result



```

xv6 kernel is booting
init: starting sh
$ ppid
My student ID is 2025123456
My pid is 3
My ppid is 2
$

```



```

xv6 kernel is booting
hart 1 starting
hart 2 starting
init: starting sh
$ ppid
My student ID is 2022083436
My pid is 3
My ppid is 2
$ git add .
exec git failed
$

```

Left side is the expected output in Assignment.pdf, and the right side screenshot is my terminal result of compilation. Just as the requirement, I successfully compiled my student ID, the process ID and the parent's process ID.

IV. Troubleshooting

One issue I met was the understanding of kernel side mechanism. I had to revise the previous lab assignments to fully understand the flow of system calls so that I could add ppid function in correct ways. This had been a very good practice of adding and making a simple system call in kernel and in user side.

Another issue I met was to update a function in user Makefile. I also followed lab 03 assignment guide.