Project #1 Report

Make System Calls

2019311779 최재경

1. System Call Implementation in *proc.c*

The following is the actual implementation of the **getnice**, **setnice**, and **ps** system call written in the *proc.c* file.

1.1. *int getnice(int pid)*

```
int
getnice(int pid)
{
  struct proc *p;

  int niceval;

  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->pid == pid){
      niceval = p->niceval;
      release(&ptable.lock);
      return niceval;
    }
  }
  release(&ptable.lock);
  return -1;
}
```

This function, *int getnice(int pid)* gets input, **pid**, and compares with pid of all processes on system process table, which represented by **struct ptable** and its member variable **struct proc**. If it successes to find same pid, it gets **nice value** of that process and return the value. If it fails to find same pid, it returns -1.

1.2. *int setnice(int pid, int niceval)*

```
int
setnice(int pid, int niceval)
{
  struct proc *p;

  if(niceval < 0 || niceval > 39){
    return -1;
  }

  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->pid == pid){
      p->niceval = niceval;
      release(&ptable.lock);
      return 0;
    }
  }
  release(&ptable.lock);
  return -1;
}
```

This function, *int setnice(int pid, int niceval)* gets input of **pid** and **niceval** and compares with pid of all processes on system process table, which represented by **struct ptable** and its member variable **struct proc**. The input range is first checked by if state, and if it successes to find same pid, it sets nice value of that proccess by input niceval. If it fails to find same pid or invalid niceval comes, it returns -1.

1.3. *void ps(int pid)*

```
void
ps(int pid)
{
  struct proc *p;

  int header_flag = 0;
  char *state[] = {"UNUSED", "EMBRYO", "SLEEPING", "RUNNABLE", "RUNNING ", "ZOMBIE  " };
  //string array structure to interperet value of p->state, represented by enum at proc.h
  //It can be also represented like


  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->pid == pid || pid == 0){
      if(header_flag == 0){
        cprintf("name\tpid\tstate\t\tpriority\n"); //It only prints once by header_flag
        header_flag = 1;
      }
      if(p->state == 0) //To avoid printing out UNUSED state processes
        break;
      cprintf("%s\t%d\t%s\t%d\n", p->name, p->pid, state[p->state], p->niceval);
    }
  }
  release(&ptable.lock);
}
```

This function, *void ps(int pd)* gets input of **pid** and prints it result - process **name**, **pid**, **state**, **nice value -** directly out to consol by *cprintf*. If ps system call argument is 0, it prints information of all process running on system, and if argument is particular pid, it only prints out information of the very process. If it fails to find same pid on ptable, it just prints nothing and terminates. To avoid UNUSED state processes be printed, if founded at ptable, it terminates loop.

`acquire(&ptable.lock);` and `release(&ptable.lock);` is for locking, to handle multiprocessors.

1.4. *struct ptable*

```
struct {
  struct spinlock lock;
  struct proc proc[NPROC];
} ptable;
```

```
Struct proc {
  …
  int niceval;           // EDIT : default nice value is 20.
};
```

The **ptable** consists of two structures representing the process table. First, **proc** is a structure that represents a real world process in a system, defined in *proc.h*, and so stores various information like size of process memory, pid, parent, and kill state. Therefore, modifying the scheduling priority of the process(**nice value)**, can be expressed as modifying the nice value of the proc structure. To this end, an int **niceval** member variable to store the nice value was added to the proc structure.

2. Follow the procedure of System Call

The following is name of the source code file that needs to be modified to create a system call.

usys.S, syscall.h, syscall.c, sysproc.c, proc.c, proc.h, user.h, defs.h, Makefile.

2.1. Assembly command in usys.S

```
#define SYSCALL(name) \
  .globl name; \
  name: \
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL; \
    ret
```

In *usys.S,* macro function is defined like above. This macro fucniton works as a system call warper function. It simplifies the process of calling system call wrapper function by providing format.

'\' : used to continue a marco definition on the next line

'##' : concatenating operator

'$' : to represent immediate value

'movl' : set the value of %eax register to SYSCALL_name

'int $T_SYSCALL' : instruction that invokes interrupt 64($T_SYSCALL) as system call interrupt.

So if input name is "fork", the instruction becomes,

```
movl $SYS_fork, %eax; \
    int $T_SYSCALL; \
    ret
```

We have to add this line:

```
SYSCALL(getnice)
SYSCALL(setnice)
SYSCALL(ps)
```

2.1.1. IDT and T_SYSCALL

int $T_SYSCALL is int 64, and performs 64th vector, defined in the idt. And syscall() in syscall.c dose the rest.

2.2. In *syscall.h*

This file has definition of every system call numbers used in xv6. We have to add this line:

name and (immediate) value that will saved in %eax register when system call is called.

```
#define SYS_getnice 23
#define SYS_setnice 24
#define SYS_ps      25
```

2.3. In *syscall.c*

This file defines handler function that will called by kernel when system call is called. Also function *syscall()* is implemented.

We have to add this line:

```
…
extern int sys_getnice(void);
extern int sys_setnice(void);
extern int sys_ps(void);
…
```

```
static int (*syscalls[])(void) = {
…
[SYS_getnice]   sys_getnice,
[SYS_setnice]   sys_setnice,
[SYS_ps]       sys_ps,
};
 …
```

(Form *static int (*syscalls[])(void) = { [...] };* is designated initializer in C. It is used to initialize specific elements of an array or structure to specific values.)

2.3.1. void syscall(void)

By getting value of %eax to num, which is index of which system call to call is found by syscalls[num]() and calls handler function in sysproc.c

2.4. *sysproc.c*

sysproc.c calles actual running code of system calls in proc.c, by getting arguments by argint() function.

We need to add this line: (sys_setnice for example.)

```
int
sys_setnice(void)
{
  int pid;
  int niceval;

  if(argint(0, &pid) < 0)
    return -1;
  if(argint(1, &niceval) < 0)
    return -1;
  return setnice(pid, niceval);
}
```

Function *argint()* is used to Fetch the nth 32-bit system call argument. It gets first and second argument of consol by adding '+4+4*n-byte' to %esp register value(address).

2.5. In *proc.c*

2.5.1. *allocproc*

allocproc function

'found:' instruction set the initial value of proc. We have to modified like this to set :

```
found:
  p->state = EMBRYO;
  p->pid = nextpid++;
  p->niceval = 20;
```

2.5.2. Actual Function

Actual function implementations are described at **1.** part above.

2.6. *user.h defs.h*

In *user.h* header file, there is definition of system call wrapper function and memory control function.

We have to add this line:

```
int getnice(int);
int setnice(int, int);
void ps(int);
```

In defs.h header file, there is definition of various function definitions that are used in various xv6 files, also including proc.c,

We have to add this line:

```
int             getnice(int);
int             setnice(int, int);
void            ps(int);
```

2.7. .c code file

Following headers are needed and implement on main function, *int argc* and *char *argv[]* to get arguments from consol. Used *atoi* function to convert string value of argument(argv) to integer. (Find usage at kill.c:15)

We have to add this file:

```
#include "types.h"
#include "user.h"
#include "stat.h"
int main(int argc, char *argv[])
```

```
 kill(atoi(argv[i])); //kill.c, line 15
```

error on input of too many/less arguments are managed by if statement.

```
//getnice.c
if(argc != 2){
    printf(2, "error on param\n"); //stderr on 2
    exit();
}
printf(1, "%d\n", getnice(atoi(argv[1])));
exit();
```

```
//setnice.c
if(argc != 3) {
    printf(2, "error of param\n");
    exit();
}
printf(1, "%d\n", setnice(atoi(argv[1]), atoi(argv[2]))); //use atoi function
exit();
```

```
//ps.c
if(argc != 2){
    printf(2, "error on param\n");
    exit();
}
ps(atoi(argv[1]));
exit();
```

2.8. Makefile

After writing getnice.c setnice.c ps.c, write it to Makefile recipe UPROGS for complie.

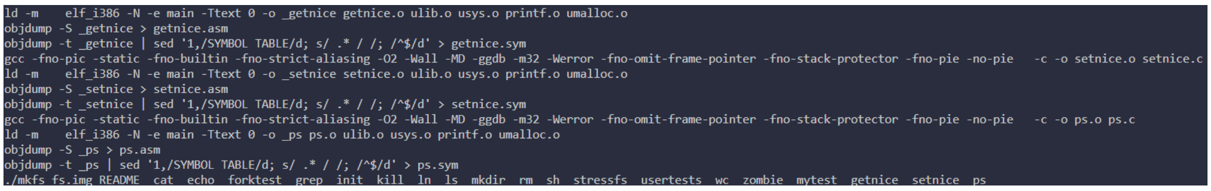
We have to add this line :

```
UPROGS=\
…
    _getnice\
    _setnice\
    _ps\
```

UPROGS contains a list of userspace programs. Builded using this rule :

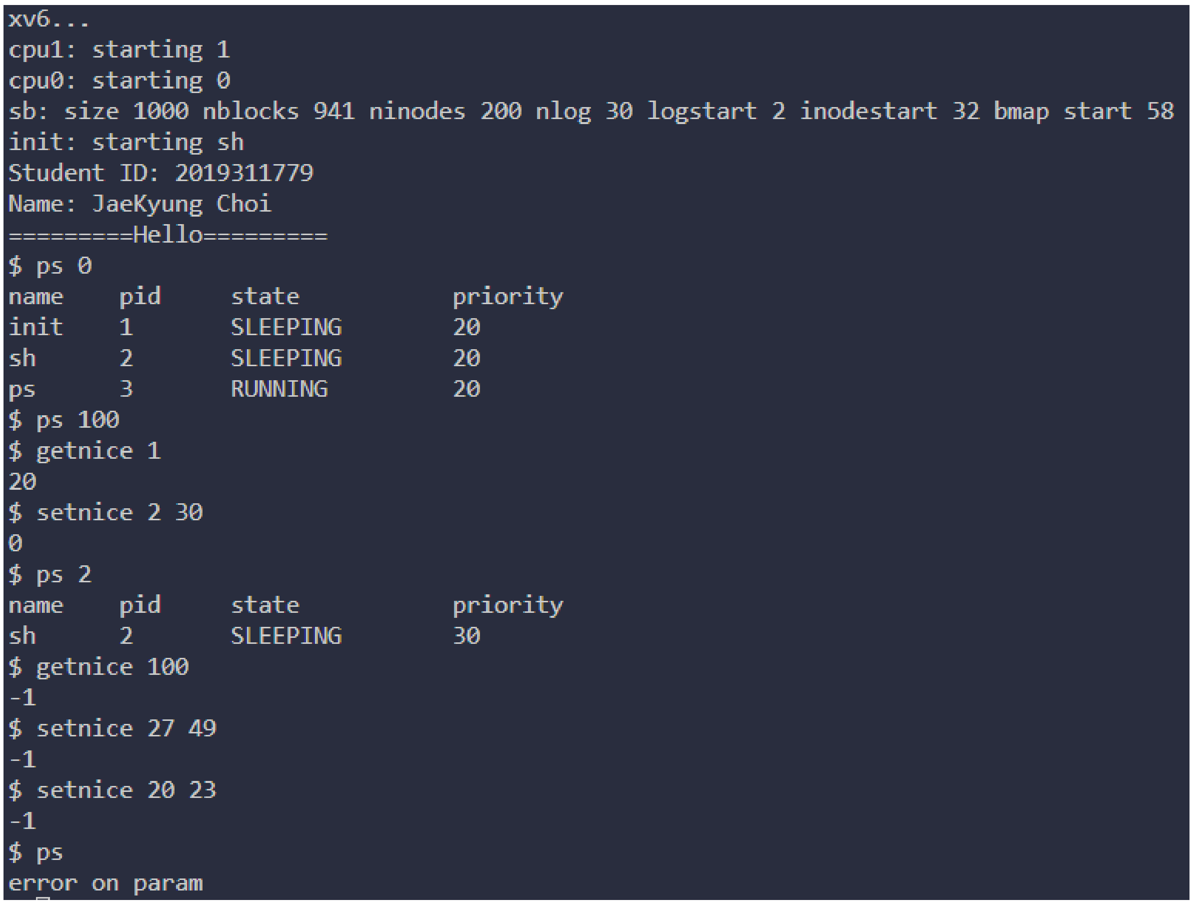
```
_%: %.o $(ULIB)
	$(LD) $(LDFLAGS) -N -e main -Ttext 0 -o $@ $^
```

(Makefile execution log)



```
ld -m    elf_i386 -N -e main -Ttext 0 -o _getnice getnice.o ulib.o usys.o printf.o umalloc.o
objdump -S _getnice > getnice.asm
objdump -t _getnice | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > getnice.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o setnice.o setnice.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _setnice setnice.o ulib.o usys.o printf.o umalloc.o
objdump -S _setnice > setnice.asm
objdump -t _setnice | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > setnice.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 ps.o ps.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _ps ps.o ulib.o usys.o printf.o umalloc.o
objdump -S _ps > ps.asm
objdump -t _ps | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > ps.sym
./mkfs fs.img README  cat  echo  forktest  grep  init  kill  ln  ls  mkdir  rm  sh  stressfs  usertests  wc  zombie  mytest  getnice  setnice  ps
```

3. Result



```
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
Student ID: 2019311779
Name: JaeKyung Choi
=========Hello=========
$ ps 0
name    pid     state           priority
init    1       SLEEPING        20
sh      2       SLEEPING        20
ps      3       RUNNING         20
$ ps 100
$ getnice 1
20
$ setnice 2 30
0
$ ps 2
name    pid     state           priority
sh      2       SLEEPING        30
$ getnice 100
-1
$ setnice 27 49
-1
$ setnice 20 23
-1
$ ps
error on param
```