

# HW4. XV6 System Call 구현하기

학번: 20170404

이름: 한종수

제출일: 2021. 4. 5

## 1. 과제 개요

본 과제에서는 프로세스와 관련된 세 가지 시스템콜인 getnice, setnice, ps를 구현하고 test 코드를 통해 올바른 동작을 확인한다.

## 2. 소스코드

(수정한 코드를 캡처하고 간단히 설명함)

```
defs.h (~/Desktop/2021_OperatingSystem/Lab4_ImplementSysCall/xv6_ssu_syscall) - VIM
File Edit View Search Terminal Help
99 int pipealloc(struct file**, struct file**);
100 void pipeclose(struct pipe*, int);
101 int piperead(struct pipe*, char*, int);
102 int pipewrite(struct pipe*, char*, int);
103
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);
117 void setproc(struct proc*);
118 void sleep(void*, struct spinlock*);
119 void userinit(void);
120 int wait(void);
121 void wakeup(void*);
122 void yeld(void);
123 int setnice(int, int); //HW#4
124 int getnice(int); //HW#4
125 void ps(void); //HW#4
126
127 // swtch.S
128 void swtch(struct context**, struct context*);
129
130 // spinlock.c
131 void acquire(struct spinlock*);
132 void getcallerpcs(void*, uint*);
```

defs.h에서 sys call 등록

```
proc.h (~/Desktop/2021_OperatingSystem/Lab4_ImplementSysCall/xv6_ssu_syscall) - VIM
File Edit View Search Terminal Help
28 uint edi;
29 uint esi;
30 uint ebx;
31 uint ebp;
32 uint eip;
33 };
34
35 enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
36
37 // Per-process state
38 struct proc {
39 ulint 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; // Saved here to run processes
47 void *chan; // If non-zero, sleeping on chan
48 int killed; // If non-zero, have been killed
49 struct file *ofile[NFILE]; // Open files
50 struct inode *cwd; // Current directory
51 char name[16]; // Process name (debugging)
52 int priority; // Process priority
53 int runtime; // Process runtime Hm#
```

proc.h에서 proc struct에 runtime value 추가

```

user.h (-/Desktop/2021_Operati...tSysCall/xv6_ssu_syscall) - VIM
File Edit View Search Terminal Help
1 struct stat;
2 struct rtcdate;
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 //HW#4
27 int setnice(int pid, int nice);
28 int getnice(int pid);
29 void ps(void);
30
31 // ulib.c
32 int stat(const char*, struct stat*);
33 char* strcpy(char*, const char*);
34 void *memmove(void*, const void*, int);

```

26,1      Top

user.h에 system call 지정

```

usys.S (-/Desktop/2021_Operati...tSysCall/xv6_ssu_syscall) - VIM
File Edit View Search Terminal Help
1 #include "syscall.h"
2 #include "traps.h"
3
4 #define SYSCALL(name) \
5     .globl name; \
6     name: \
7         movl $SYS_ ## name, %eax; \
8         int $T_SYSCALL; \
9         ret
10
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(setnice)
33 SYSCALL(getnice)
34 SYSCALL(ps)

```

"usys.S" 34L, 507C      33,1      All

usys.S에서 SYSCALL 호출 시 syscall 번호 넣도록 수정

```

syscall.h (-/Desktop/2021_OperatingSystem/Lab4_ImplementSys
File Edit View Search Terminal Help
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_setnice 22
24 #define SYS_getnice 23
25 #define SYS_ps 24

```

syscall.h에서 syscall\_number 지정

```

syscall.c (~/Desktop/2021_OperatingSystem/L
File Edit View Search Terminal Help
85 extern int sys_chdir(void);
86 extern int sys_close(void);
87 extern int sys_dup(void);
88 extern int sys_exec(void);
89 extern int sys_exit(void);
90 extern int sys_fork(void);
91 extern int sys_fstat(void);
92 extern int sys_getpid(void);
93 extern int sys_kill(void);
94 extern int sys_link(void);
95 extern int sys_mkdir(void);
96 extern int sys_mknod(void);
97 extern int sys_open(void);
98 extern int sys_pipe(void);
99 extern int sys_read(void);
100 extern int sys_sbrrk(void);
101 extern int sys_sleep(void);
102 extern int sys_unlink(void);
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_setnice(void);
107 extern int sys_getnice(void);
108 extern int sys_ps(void);
109
110 static int (*syscalls[])(void) = {
111 [SYS_fork] sys_fork,
112 [SYS_exit] sys_exit,
113 [SYS_wait] sys_wait,
114 [SYS_pipe] sys_pipe,
115 [SYS_read] sys_read,
116 [SYS_kill] sys_kill,
117 [SYS_exec] sys_exec,
118 [SYS_fstat] sys_fstat,
119 [SYS_chdir] sys_chdir,
120 [SYS_dup] sys_dup,
121 [SYS_getpid] sys_getpid,
122 [SYS_sbrrk] sys_sbrrk,
123 [SYS_sleep] sys_sleep,
124 [SYS_uptime] sys_uptime,
125 [SYS_open] sys_open,
126 [SYS_write] sys_write,
127 [SYS_mknod] sys_mknod,
128 [SYS_unlink] sys_unlink,
129 [SYS_link] sys_link,
130 [SYS_mkdir] sys_mkdir,
131 [SYS_close] sys_close,
132 [SYS_setnice] sys_setnice,
133 [SYS_getnice] sys_getnice,
134 [SYS_ps] sys_ps
135 };
136

```

syscall.c system call table에 핸들러 등록

```

sysproc.c + (~/Desktop/2021_OperatingSys.
File Edit View Search Terminal Help
92
93 //HW#4
94 int
95 sys_setnice(void)
96 {
97     int pid;
98     int nice;
99
100    if(argint(0, &pid) < 0)
101        return -1;
102
103    if(argint(1, &nice) < 0)
104        return -1;
105
106    return setnice(pid, nice);
107 }
108
109 int
110 sys_getnice(void)
111 {
112     int pid;
113
114     if(argint(0, &pid) < 0)
115         return -1;
116
117     return getnice(pid);
118 }
119
120 int
121 sys_ps(void)
122 {
123     ps();
124     return 0;
125 }

```

sysproc.c의 setnice, getnice, ps의 핸들러 인자가 제대로 들어왔는지 검사해서 인자들을 시스템 콜 핸들러를 실행.

```

proc.c (~/Desktop/2021_Operating...entSysCall/xv6_ssu_syscall) - VIM
File Edit View Search Terminal Help
538
539 int setnice(int pid, int nice)
540 {
541     if( nice < 0 || nice > 10)
542         return -1;
543
544     struct proc *p;
545     acquire(&ptable.lock);
546
547     /*****
548     /*      do program      */
549     *****/
550     for( p = ptable.proc; p < &ptable.proc[NPROC]; p++)
551     {
552         if(p->pid == pid)
553         {
554             p->priority = nice;
555             release(&ptable.lock);
556             return 0;
557         }
558     }
559
560     release(&ptable.lock);
561     return -1; //non-existing pid
562 }
563
564 int getnice(int pid)
565 {
566     struct proc *p;
567     acquire(&ptable.lock);
568
569     /*****
570     /*      do program      */
571     *****/
572     for( p = ptable.proc; p < &ptable.proc[NPROC]; p++)
573     {
574         if(p->pid == pid)
575         {
576             release(&ptable.lock);
577             return p->priority;
578         }
579     }
580
581     release(&ptable.lock);
582
583     return -1; //non-existing pid
584 }
585

proc.c (~/Desktop/2021_OperatingSystem/Lab4_ImplementSysCall/xv6_ssu_syscall) - VIM
File Edit View Search Terminal Help
587 void ps(void)
588 {
589
590     struct proc *p;
591     acquire(&ptable.lock);
592     cprintf("name\tpid \t state \t priority \t runtime \t tick %d\n", ticks);
593     for( p = ptable.proc; p < &ptable.proc[NPROC]; p++)
594     {
595         if(p->pid==0)
596             continue;
597         if(p->state==UNUSED)
598             cprintf("%s\t%d \t %s \t %d \t %d \t %d\n", p->name, p->pid, "UNUSED", p->priority, p->runtime);
599         else if(p->state==EMBRYO)
600             cprintf("%s\t%d \t %s \t %d \t %d \t %d\n", p->name, p->pid, "EMBRYO", p->priority, p->runtime);
601         else if(p->state==SLEEPING)
602             cprintf("%s\t%d \t %s \t %d \t %d \t %d\n", p->name, p->pid, "SLEEPING", p->priority, p->runtime);
603         else if(p->state==RUNNABLE)
604             cprintf("%s\t%d \t %s \t %d \t %d \t %d\n", p->name, p->pid, "RUNNABLE", p->priority, p->runtime);
605         else if(p->state==RUNNING)
606             cprintf("%s\t%d \t %s \t %d \t %d \t %d\n", p->name, p->pid, "RUNNING", p->priority, p->runtime);
607         else if(p->state==ZOMBIE)
608             cprintf("%s\t%d \t %s \t %d \t %d \t %d\n", p->name, p->pid, "ZOMBIE", p->priority, p->runtime);
609     }
610     release(&ptable.lock);
611     return;
612 }
613
614

```

## proc.c

- 1) setnice : nice 값이 잘 들어왔는지 범위체크해서 범위 넘으면 -1 리턴. 아니면 ptable 둘면서 일치하는 pid 검색. 있으 면 해당 pid의 nice를 인자로 들어온 nice값으로 수정. 찾는 pid가 없으면 -1 리턴
- 2) getnice : nice 값이 잘 들어왔는지 범위체크해서 범위 넘으면 -1 리턴. 아니면 ptable 둘면서 일치하는 pid 검색. 있으 면 해당 pid의 nice를 리턴. 찾는 pid가 없으면 -1 리턴
- 3) ps : ptable을 둘며 현재 있는 process 들의 정보를 출력한다.

```

proc.c (~/Desktop/2021_Operating...entSysCall/xv6_ssu_syscall) - VIM      proc.c (~/Desktop/2021_Operating...entSysCall/xv6_ssu_syscall) - VIM
File Edit View Search Terminal Help                                     File Edit View Search Terminal Help
69 // Look in the process table for an UNUSED proc.          178
70 // If found, change state to EMBRYO and initialize        179 // Create a new process copying p as the parent.
71 // state required to run in the kernel.                   180 // Sets up stack to return as if from system call.
72 // Otherwise return 0.                                     181 // Caller must set state of returned proc to RUNNABLE.
73 static struct proc*          182 int
74 allocproc(void)           183 fork(void)
75 {                      184 {
76     struct proc *p;    185     int i, pid;
77     char *sp;          186     struct proc *np;
78     acquire(&ptable.lock); 187     struct proc *curproc = myproc();
79     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) 188
80         if(p->state == UNUSED) 189     // Allocate process.
81             goto found; 190     if((np = allocproc()) == 0){
82         p->state = EMBRYO; 191         return -1;
83     } 192     }
84     release(&ptable.lock); 193     // Copy process state from proc.
85     return 0; 194     if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
86 195         kfree(np->kstack);
87 196         np->kstack = 0;
88 found: 197         np->state = UNUSED;
89     p->runtime = 0; 198         return -1;
90     p->pid = nextpid++; 199     }
91     release(&ptable.lock); 200     }
92 201     np->sz = curproc->sz;
93 202     np->parent = curproc;
94 203     *np->tf = *curproc->tf;
95 // Allocate kernel stack. 204     np->priority = curproc->priority; //HW#4
96     if((p->kstack = kalloc()) == 0){ 205
97         p->state = UNUSED; 206     // Clear %eax so that fork returns 0 in the child.
98         return 0; 207     np->tf->eax = 0;
99     } 208     for(i = 0; i < NOFILE; i++)
100    sp = p->kstack + KSTACKSIZE; 209         if(curproc->ofile[i])
101 210             np->ofile[i] = filedup(curproc->ofile[i]);
102 // Leave room for trap frame. 211     np->cwd = idup(curproc->cwd);
103    sp -= sizeof *p->tf; 212     safestrcpy(np->name, curproc->name, sizeof(curproc->name));
104    p->tf = (struct trapframe*)sp; 213
105 214     pid = np->pid;
106 // Set up new context to start executing at forkret, 215     acquire(&ptable.lock);
107 // which returns to trapret. 216     np->state = RUNNABLE;
108    sp -= 4; 217     release(&ptable.lock);
109    *(uint*)sp = (uint)trapret; 218     return pid;
110 219 }
111    sp -= sizeof *p->context; 220 }
112    p->context = (struct context*)sp; 221
113    memset(p->context, 0, sizeof *p->context); 222
114    p->context->el1 = (uint)forkret; 223
115 224 }
116    return p; 225 }
117 } 226

```

72,1 12% 226,0~1 31%

  

```

proc.c (~/Desktop/2021_Operating...entSysCall/xv6_ssu_syscall) - VIM
File Edit View Search Terminal Help
120 // Set up first user process.
121 void
122 userinit(void)
123 {
124     struct proc *p;
125     extern char _binary_initcode_start[], _binary_initcode_size[];
126
127     p = allocproc();
128
129     initproc = p;
130     if((p->pgdir = setupkvm()) == 0)
131         panic("userinit: out of memory!");
132     inituvm(p->pgdir, _binary_initcode_start, (int)_binary_initcode_size);
133     p->sz = PGSIZE;
134     memset(p->tf, 0, sizeof(*p->tf));
135     p->tf->cs = (SEG_UCODE << 3) | DPL_USER;
136     p->tf->ds = (SEG_UDATA << 3) | DPL_USER;
137     p->tf->es = p->tf->ds;
138     p->tf->ss = p->tf->ds;
139     p->tf->eflags = FL_IF;
140     p->tf->esp = PGSIZE;
141     p->tf->ip = 0; // beginning of initcode.s
142
143     safestrcpy(p->name, "initcode", sizeof(p->name));
144     p->cwd = namei("/");
145
146     // this assignment to p->state lets other cores
147     // run this process. the acquire forces the above
148     // writes to be visible, and the lock is also needed
149     // because the assignment might not be atomic.
150     acquire(&ptable.lock);
151
152     p->state = RUNNABLE;
153     p->priority = 5; //HW#4
154
155     release(&ptable.lock);
156 }

```

120,1 20%

## proc.c

allocproc()에서 프로세스 새로 할당시 runtime을 0

으로 초기화 시킨다

userinit()에서 init 프로세스 생성시 priority를 5로 설정한다.

fork()에서 child 생성시 parent의 priority를 복사한다.

```
getnice.c (~/Desktop/2021_Operat...sCall/xv6_ssu_syscall/user) - VIM
File Edit View Search Terminal Help
1 #include "types.h"
2 #include "stat.h"
3 #include "user.h"
4
5 int main(int argc, char **argv)
6 {
7     int result = 0;
8     int pid = 0;
9
10    if (argc < 2)
11    {
12        printf(2, "usage: getnice pid ...\\n");
13        exit();
14    }
15    pid = atoi(argv[1]);
16    result = getnice(atoi(argv[1]));
17
18    if(result < 0)
19    {
20        printf(2, "non-existing process\\n");
21    }
22    else
23    {
24        printf(2, "pid : %d nice : %d\\n", pid, result);
25    }
26    exit();
27 }
```

user/getnice.c 시스템 콜이 잘 되는지 시험하기 위한 파일. 인자가 잘 들어오는지 확인 후 시스템 콜을 출력한다. 호출 결과가 0 미만이면 에러 문장출력, 호출결과가 0 이상이면 찾으려 했던 pid와 nice 값을 출력

```
setnice.c (~/Desktop/2021_Operat...sCall/xv6_ssu_syscall/user) - VIM
File Edit View Search Terminal Help
1 #include "types.h"
2 #include "stat.h"
3 #include "user.h"
4
5 int main(int argc, char **argv)
6 {
7     int result = 0;
8     int pid = 0;
9     int nice = 0;
10    if (argc < 3)
11    {
12        printf(2, "usage: setnice pid nice ...\\n");
13        exit();
14    }
15    pid = atoi(argv[1]);
16    nice = atoi(argv[2]);
17    result = setnice(pid, nice);
18
19    if(result < 0)
20    {
21        printf(2, "non-existing process\\n");
22    }
23    else
24    {
25        printf(2, "set pid:%d nice to %d\\n", pid, nice);
26    }
27    exit();
28 }
```

user/setnice.c 시스템 콜이 잘 되는지 시험하기 위한 파일. 인자가 잘 들어오는지 확인 후 시스템 콜을 출력한다. 호출 결과가 0 미만이면 에러 문장출력, 호출결과가 0 이상이면 설정하려 했던 pid와 수정된 nice 값을 출력

### 3. 결과

(테스트 실행결과를 캡처 )

```
jongsoo@Linux: ~/Desktop/2021_OperatingSystem/Lab4_ImplementSysCall/xv6_ssu_syscall - □ ×
[jongsoo@Linux: ~/Desktop/2021_OperatingSystem/Lab4_ImplementSysCall/xv6_ssu_syscall 85x46]
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
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
$ ls
.          1 1 512
..         1 1 512
README     2 2 2170
cat        2 3 13224
echo       2 4 12448
forktest   2 5 13136
grep       2 6 14668
init       2 7 13020
kill       2 8 12496
ln         2 9 12396
ls         2 10 14440
mkdir      2 11 12592
rm         2 12 12568
sh         2 13 21624
stressfs   2 14 13184
usertests  2 15 54708
wc         2 16 13672
zombie    2 17 12228
ps         2 18 12264
test_nice  2 19 13940
test_sys_ps 2 20 13044
getnice    2 21 12388
setnice    2 22 12428
console    3 23 0
$ test_nice
== TEST START ==
case 1. get nice value of init process: OK
case 2. get nice value of non-existing process: OK
case 3. set nice value of current process: OK
case 4. set nice value of non-existing process: OK
case 5. set wrong nice value of current process: OK
case 6. get nice value of forked process: OK
== TEST END ==
$ test_sys_ps
== TEST START ==
1st pid: 6
== TEST END ==
$ ]
```