OS PA_1 report

소프트웨어학과 2018314827 차승일

Abstract

바꾼 파일의 리스트는 아래와 같다.

```
● 10:22:46  |base|jet981217@jet981217-Z690-AORUS-ELITE-AX-DDR4 xv6-public ±|master ✗|→ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Makefile
        modified:   defs.h
        modified:   proc.c
        modified:   proc.h
        modified:   syscall.c
        modified:   syscall.h
        modified:   sysproc.c
        modified:   user.h
        modified:   usys.S
```

과제 설명 영상에서 가이드한 순서대로 파일을 modify 하였고, 이 순서대로 설명하겠다.

proc.h

```
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     int nice;
53   };
```

프로세스 구조체가 nice값을 가지고 있어야 하므로 int nice를 추가해줬다.

usys.S

```
31  SYSCALL(uptime)
32  SYSCALL(getnice)
33  SYSCALL(setnice)
34  SYSCALL(ps)
```

기존 uptime 시스템콜 밑에 getnice, setniice, ps 시스템콜을 추가하였다.

syscall.h

```
22  #define SYS_close  21
23  #define SYS_getnice 22
24  #define SYS_setnice 23
25  #define SYS_ps 24
```

기존 syscall number 21번 밑 22,23,24번으로 getnice, setnice, ps를 할당해 주었다.

syscall.c

```
106  extern int sys_getnice(void);
107  extern int sys_setnice(void);
108  extern int sys_ps(void);
```

```
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_sbrk]    sys_sbrk,
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_getnice] sys_getnice,
133  [SYS_setnice] sys_setnice,
134  [SYS_ps]      sys_ps,
135  };
```

과제 가이드라인대로 해당 시스템 콜에 대해 extern을 추가하였고, syscall element를 syscall.c에 추가하였다.

sysproc.c

```
93  int
94  sys_getnice(void)
95  {
96    int pid;
97    int return_value;
98
99    if(argint(0, &pid) < 0)
100     return_value = -1;
101   else{
102     return_value = getnice(pid);
103   }
104   return return_value;
105 }
106
107 int
108 sys_setnice(void)
109 {
110   int pid, value;
111   int return_value;
112
113   if(argint(1, &value) < 0 || argint(0, &pid) < 0)
114     return_value = -1;
115   else{
116     return_value = setnice(pid, value);
117   }
118
119   return return_value;
120 }
121
122 int
123 sys_ps(void)
124 {
125   int pid;
126   if(argint(0, &pid) < 0)
127     return -1;
128   else{
129     ps(pid);
130     return 0;
131   }
132 }
```

sys_getnice, sys_setnice, sys_ps라는 wrapper function을 sysproc.c에 implement 하였다.

sys_getnice 에서는 pid가 음수인지 체크하고, 음수라면 invalid 하므로 -1을 return 하고, 아니라면 getnice 함수를 불러 pid에 해당하는 nice값을 return 한다

sys_setnice 에서도 pid가 음수인지 체크하고, nice값이 음수는 될 수 없으므로 set value도 음수인지 체크한다. 그리고 둘 중 하나라도 음수라면 invalid 하므로 -1을 return. 아니라면 setnice에 nice값을 set해야하는 pid와 바꿀 nice값이 value를 파라미터로 전달하여 setnice 함수의 리턴값을 리턴한다.

sys_ps 에서도 마찬가지로 음수인 pid인지 check하고 잘못되었다면 -1을 리턴하고 아니라면 ps를 call하여 출력되도록 한다. 그리고 0을 리턴한다.

proc.c

프로세스를 만들때 디폴트 nice값을 정해준다.(p->nice = 20;)

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

  release(&ptable.lock);

  // Allocate kernel stack.
  if((p->kstack = kalloc()) == 0){
    p->state = UNUSED;
    return 0;
  }
  sp = p->kstack + KSTACKSIZE;

  // Leave room for trap frame.
  sp -= sizeof *p->tf;
  p->tf = (struct trapframe*)sp;

  // Set up new context to start executing at forkret,
  // which returns to trapret.
  sp -= 4;
  *(uint*)sp = (uint)trapret;

  sp -= sizeof *p->context;
  p->context = (struct context*)sp;
  memset(p->context, 0, sizeof *p->context);
  p->context->eip = (uint)forkret;

  return p;
}
```

위의 wrapper function에서 부르는 실제 function인 getnice, setnice, ps를 proc.c에 구현하였다.
먼저 getnice에 대해 설명하겠다.

```
537  int
538  getnice(int pid)
539  {
540    struct proc *p;
541    acquire(&ptable.lock);
542
543    int return_value = -1;
544    for(
545      p = ptable.proc;
546      p < &ptable.proc[NPROC];
547      p++
548    ){
549      if(
550        p->pid == pid
551      ){
552        return_value = p->nice;
553        break;
554      }
555    }
556    release(&ptable.lock);
557    return return_value;
558  }
```

nice값이 찾아야 하는 pid가 파라미터로 전달된다. nice값을 조회하는 동안 타 프로세스가 자원에 접근하는 것을 원하지 않으므로, 락킹을 걸어놓고 process table을 조회하여 해당 pid가 있는지 서치한다. 존재한다면, 해당 pid에 해당하는 process의 nice값을 reuturn 하고 없다면 -1을 리턴한다. 리턴하기 전에 락킹을 해제한다.

다음은 setnice에 대해 설명하겠다.

```
560  int
561  setnice(int pid, int value)
562  {
563    int return_value = -1;
564    if(value > 39 || value < 0){
565      return return_value;
566    }
567
568    struct proc *p;
569    acquire(&ptable.lock);
570    for(
571      p = ptable.proc;
572      p < &ptable.proc[NPROC];
573      p++
574    ){
575      if(p->pid == pid){
576        p->nice = value;
577        return_value = 0;
578
579        break;
580      }
581    }
582    release(&ptable.lock);
583    return return_value;
584  }
```

nice값을 수정할 pid와 수정할 나이스값을 파라미터로 전달한다.nice값은 39를 넘어갈 수 없고, 0보다 작을 수 없다. 따라서 이 조건에 맞지 않으면 invalid 하므로 -1을 return 한다. 다음 이 조건에 맞는다면 ptable에 위와 같은 이유로 락킹을 걸고 ptable을 iterate해 해당 pid를 가지고 있는 process가 있는지 조사한다. 조사해서 있다면 해당 pid의 process의 nice값을 업데이트한다. 성공하면 0을 리턴, 실패하면 -1을 리턴하고 리턴하기 전에 락킹을 해제한다.

다음은 ps에 대해 설명하겠다.

```
586 void ps(int pid){
587   char *states_by_idx[] = {"UNUSED", "EMBRYO  ", "SLEEPING", "RUNNABLE", "RUNNING ", "ZOMBIE  "}
588   struct proc *p;
589
590   acquire(&ptable.lock);
591   cprintf("name\t\tpid\t\tstate\t\tpriority\n");
592   if(pid){
593     for(
594       p = ptable.proc;
595       p <= &ptable.proc[NPROC];
596       p++
597     ){
598       if(p->pid == pid){
599         if(p->state != 0 && p->state <= 5){
600           cprintf(
601           "%s\t\t%d\t\t%s\t%d\n",
602             p->name, p->pid, states_by_idx[p->state], p->nice
603           );
604         }
605         break;
606       }
607     }
608   }
609   else{
610     for(
611       p = ptable.proc;
612       p <= &ptable.proc[NPROC];
613       p++
614     ){
615       if(p->state != 0 && p->state <= 5 && p->pid >= 0){
616         cprintf(
617           "%s\t\t%d\t\t%s\t%d\n",
618           p->name, p->pid, states_by_idx[p->state], p->nice
619         );
620       }
621     }
622   }
623
624   release(&ptable.lock);
625 }
```

pid는 파라미터로 전달된다. state는 int로 저장되어 있는데 이를 string으로 convert할 배열을 선언해주었다.(테이블 모양으로 맞추기 위해 빈칸이 이름 뒤에 붙어있는 배열 요소가 있다.) 먼저 락킹은 당연히 걸고 시작한다.

state가 0인 경우, 즉 unused인 경우는 출력하지 않고, state가 5를 초과할 경우에도 출력을 하면 안된다. 따라서 출력을 할때는 0<state<=5 이다.

pid가 0일 경우에는 모든 프로세스의 정보를 출력하라 했으므로, ptable을 돌며 각 프로세스의 name, pid, state(string으로 convert해서), nice값을 출력한다(당연히 0<state<=5이고 pid가 0 이상인 경우들만). 그 후 락킹 풀고 함수가 끝난다.

pid가 0이 아닌 경우에는 ptable을 돌며 pid에 해당되는 프로세스가 있는지 체크, 있다면 0<state<=5인지 체크 하고 똑같이 name, pid, state(string으로 convert해서), nice값을 출력하고 break한다. pid가 존재하지 않는다면 아무것도 출력하지 않는다.

defs.h and user.h

```
123  int             getnice(int);
124  int             setnice(int, int);
125  void            ps(int);
```

defs.h에는 아래와 같이 implement,

```
26  int getnice(int);
27  int setnice(int, int);
28  void ps(int);
```

user.h에도 아래와 같이 implement하였다.

Makefile과 **test code**는 채점에 반영되지 않는다 하였으므로 따로 설명을 기술하진 않겠다**.**