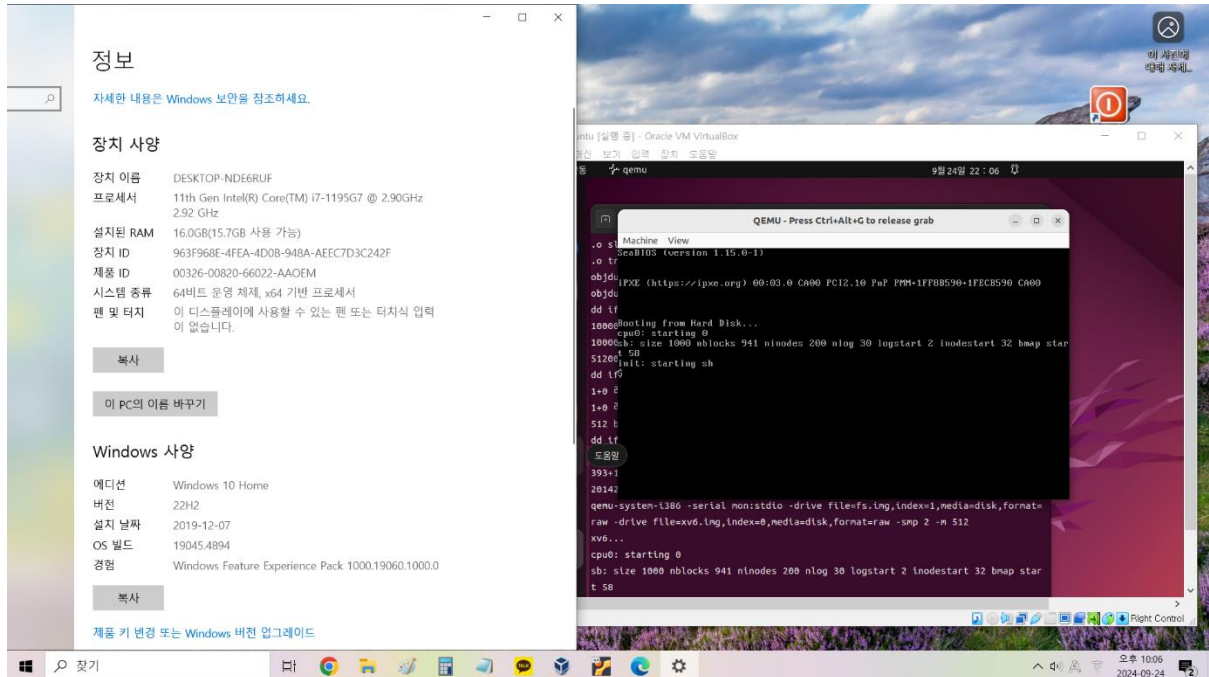


소속 : 소프트웨어학부

학번 : 20221993

이름 : 이재준

개발 환경



Lg gram 2022를 사용하고 있습니다. 사양은 위와 같습니다. 운영체제는 virtual machine을 이용한 우분투 리눅스를 사용했습니다.

XV6 운영체제의 스케줄러인 void scheduler(void) 함수 분석하기

```
void
scheduler(void)
{
    struct proc *p;
    struct cpu *c = mycpu();
    c->proc = 0;

    for(;;){
        // Enable interrupts on this processor.
        sti();

        // Loop over process table looking for process to run.
        acquire(&ptable.lock);
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
            if(p->state != RUNNABLE)
                continue;

            // Switch to chosen process. It is the process's job
            // to release ptable.lock and then reacquire it
            // before jumping back to us.
            c->proc = p;
            switchvm(p);
            p->state = RUNNING;

            swtch(&(c->scheduler), p->context);
            switchkvm();

            // Process is done running for now.
            // It should have changed its p->state before coming back.
            c->proc = 0;
        }
        release(&ptable.lock);
    }
}
```

우선 scheduler() 함수는 위와 같습니다. (출처 : xv6-source-rev11). scheduler()는 cpu 구조체에 mycpu()를 할당하고 현재 proc 에 0 을 할당합니다. 그리고 무한 루프를 실행하는데 이는 운영체제가 스케줄러를 이용해 계속 프로세스를 관리하기 때문입니다.

sti()는 인터럽트를 허용하는 함수입니다.

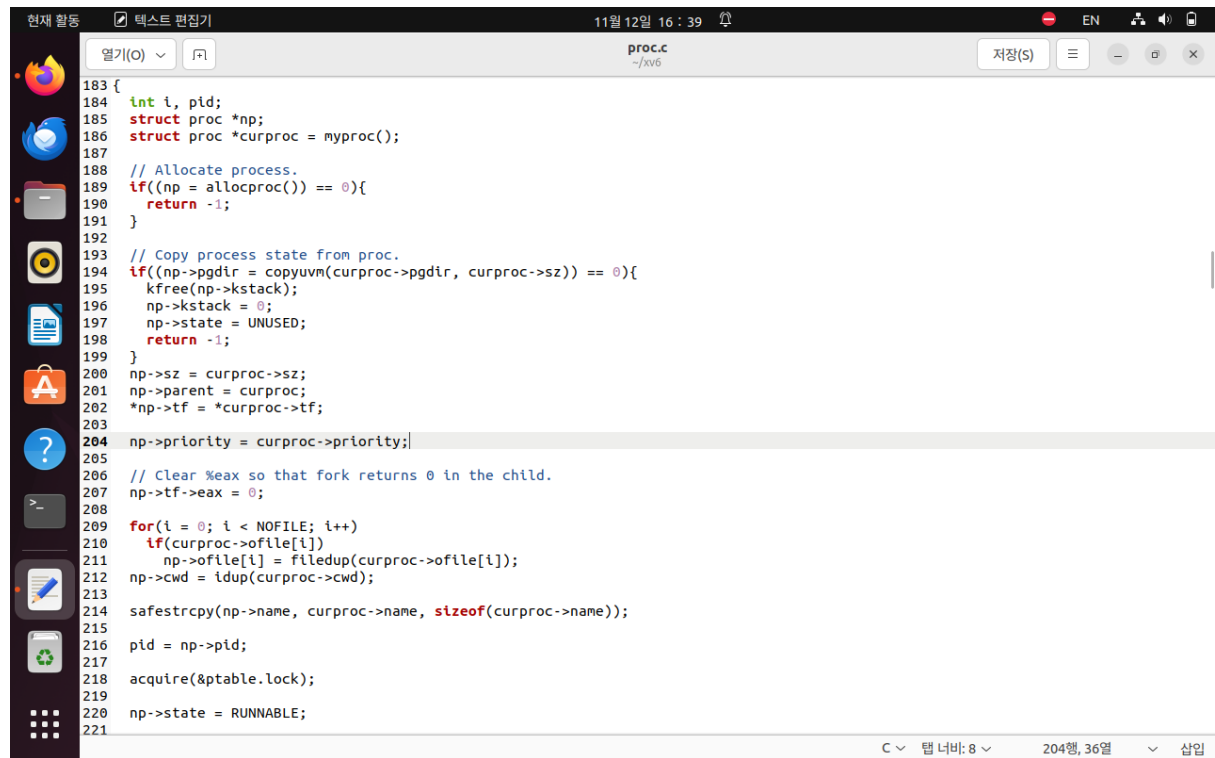
acquire(&ptable.lock)와 release(&ptable.lock)를 통해서 프로세스 테이블에 대한 접근을 관리합니다.

다음 for()문에서 프로세스 테이블의 처음부터 끝까지 순회합니다. 만약 RUNNABLE 아닌 경우 다음 프로세스를 확인합니다. for()문 안에는 swtch()와 switchkvm()이 있는데 각각 문맥 전환과 커널의 페이지 테이블로의 전환을 담당합니다.

swtch()의 경우 인자로 struct context ** old 와 struct context * new 를 받습니다. old 에 현재 레지스터를 저장하고 new 로부터 새로운 레지스터를 로드합니다. swtch()는 cpu 구조체의 멤버인 context 구조체와 proc 구조체의 멤버인 context 구조체에

사용됩니다. 한편 `swtch()`는 `sched()`에도 사용되는데 이를 통해 프로세스 구조체의 `context`와 `mycpu()`의 `scheduler`을 교환할 수 있습니다.

수정 및 작성한 소스코드에 대한 설명



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

자식 프로세스와 부모 프로세스의 priority 를 동일하게 만드는 코드입니다.

현재 활동 텍스트 편집기 11월 12일 17:26 EN

열기(O) [icon] usys.S ~/xv6 저장(S) [icon] [icon] [icon]

```
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(forknexec)
33 SYSCALL(set_proc_priority)
34 SYSCALL(get_proc_priority)
35
```

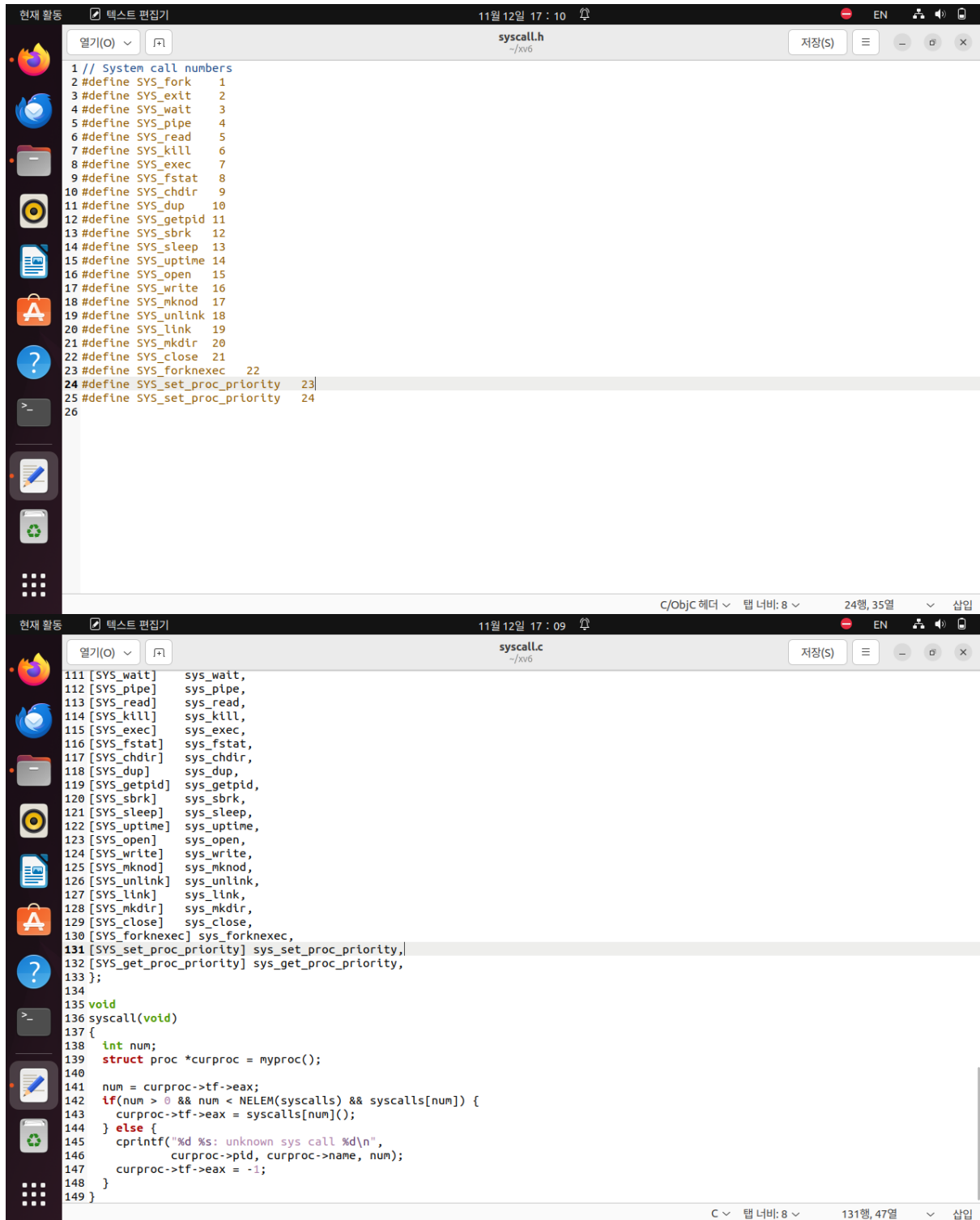
맞는 대괄호가 다음 줄에 있습니다: 33 C 탭 너비: 8 33행, 27열 삽입

현재 활동 텍스트 편집기 11월 12일 17:25 EN

열기(O) [icon] user.h ~/xv6 저장(S) [icon] [icon] [icon]

```
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 forknexec(const char *, const char **);
27 int set_proc_priority(int, int);
28 int get_proc_priority(int);
29
30 // ulib.c
31 int stat(const char*, struct stat*);
32 char* strcpy(char*, const char*);
33 void *memmove(void*, const void*, int);
34 char* strchr(const char*, char c);
35 int strcmp(const char*, const char*);
36 void printf(int, const char*, ...);
37 char* gets(char*, int max);
38 uint strlen(const char*);
39 void* memset(void*, int, uint);
40 void* malloc(uint);
41 void free(void*);
42 int atoi(const char*);
43
```

C/ObjC 헤더 탭 너비: 8 27행, 33열 삽입



현재 활동

텍스트 편집기

11월 12일 17:04

EN

🔊 🔌 🖥

열기(O) 🔍

*defs.h
~/xv6

저장(S) ⋮ ⏪ ⏩ ✕

```
159 // timer.c
160 void timerinit(void);
161
162 // trap.c
163 void idtinit(void);
164 extern uint ticks;
165 void tvinit(void);
166 extern struct spinlock tickslock;
167
168 // uart.c
169 void uartinit(void);
170 void uartintr(void);
171 void uartputc(int);
172
173 // vm.c
174 void seginit(void);
175 void kvmalloc(void);
176 pde_t* setupkvm(void);
177 char* uva2ka(pde_t*, char*);
178 int allocuvn(pde_t*, uint, uint);
179 int deallocuvn(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
189 // number of elements in fixed-size array
190 #define NELEM(x) (sizeof(x)/sizeof((x)[0]))
191
192 // forknexec.c
193 int forknexec(const char *, const char **);
194
195 int set_proc_priority(int, int);
196 int get_proc_priority(int);
197
```

C/ObjC 헤더 ▼ 탭 너비: 8 ▼ 195행, 45열 ▼ 삼입

현재 활동

텍스트 편집기

11월 12일 17:00

EN

🔊 🔌 🖥

열기(O) 🔍

sysproc.c
~/xv6

저장(S) ⋮ ⏪ ⏩ ✕

```
110 if (fetchint(uargv + 4*i, (int *)&uarg) < 0)
111     return -1;
112
113 if (uarg == 0) {
114     args[i] = 0;
115     break;
116 }
117
118 if (fetchstr(uarg, &args[i]) < 0)
119     return -1;
120 }
121
122 return forknexec((const char *)path, (const char **)args);
123 }
124 */
125
126 int
127 sys_set_proc_priority(void)
128 {
129     int id;
130     int priority;
131
132     if (argint(0, &id) < 0 || argint(1, &priority) < 0)
133         return -1;
134
135     return set_proc_priority(id, priority);
136 }
137
138 int
139 sys_get_proc_priority(void)
140 {
141     int id;
142
143     if (argint(0, &id) < 0)
144         return -1;
145
146     return get_proc_priority(id);
147 }
148
```

C ▼ 탭 너비: 8 ▼ 139행, 28열 ▼ 삼입

맞는 대괄호가 다음 줄에 있습니다: 139

현재 활동 텍스트 편집기 11월 12일 16:59

proc.c ~/xv6 저장(S)

```
681 cr(pgid) {
682     freevm(pgid);
683     if(ip){
684         iunlockput(ip);
685         end_op();
686     }
687     return -2; // etc err
688 }
689 */
690
691 int set_proc_priority(int id, int priority) {
692     struct proc * p;
693
694     if (p < 1 || p > 10)
695         return -1;
696
697     for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
698         if (p->id == id) {
699             acquire(&ptable.lock);
700             p->priority = priority;
701             release(&ptable.lock);
702             return p->priority;
703         }
704     }
705     return -1;
706 }
707
708 int get_proc_priority(int id) {
709     struct proc * p;
710
711     for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
712         if (p->id == id)
713             return p->priority;
714     }
715     return -1;
716 }
717
718
719
```


맞는 대괄호가 다음 줄에 있습니다: 719 C 탭 너비: 8 710행, 32열 삽입

현재 활동 텍스트 편집기 11월 12일 16:54

sysproc.c ~/xv6 저장(S)

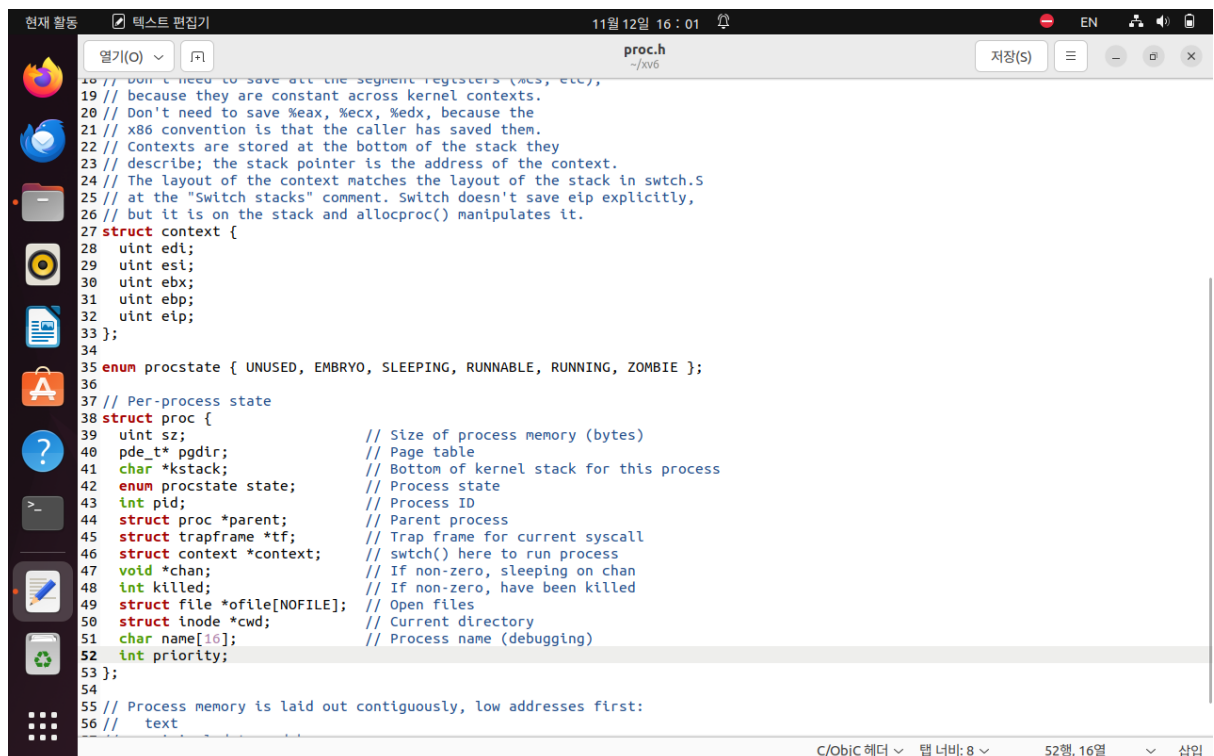
```
99 int i;
100
101 if (argstr(0, &path) < 0 || argint(1, (int *)&uargv) < 0)
102     return -1;
103
104 memset(args, 0, sizeof(args));
105
106 for (i = 0; ; i++) {
107     if (i >= NELEM(args))
108         return -1;
109
110     if (fetchint(uargv + 4*i, (int *)&uarg) < 0)
111         return -1;
112
113     if (uarg == 0) {
114         args[i] = 0;
115         break;
116     }
117
118     if (fetchstr(uarg, &args[i]) < 0)
119         return -1;
120 }
121
122 return forknexec((const char *)path, (const char **)args);
123 }
124 */
125
126 int
127 sys_set_proc_priority(void)
128 {
129     int id;
130     int priority;
131
132     if (argint(0, &id) < 0 || argint(1, &priority) < 0)
133         return -1;
134
135     return set_proc_priority(id, priority);
136 }
137
```

맞는 대괄호가 다음 줄에 있습니다: 127 C 탭 너비: 8 127행, 28열 삽입



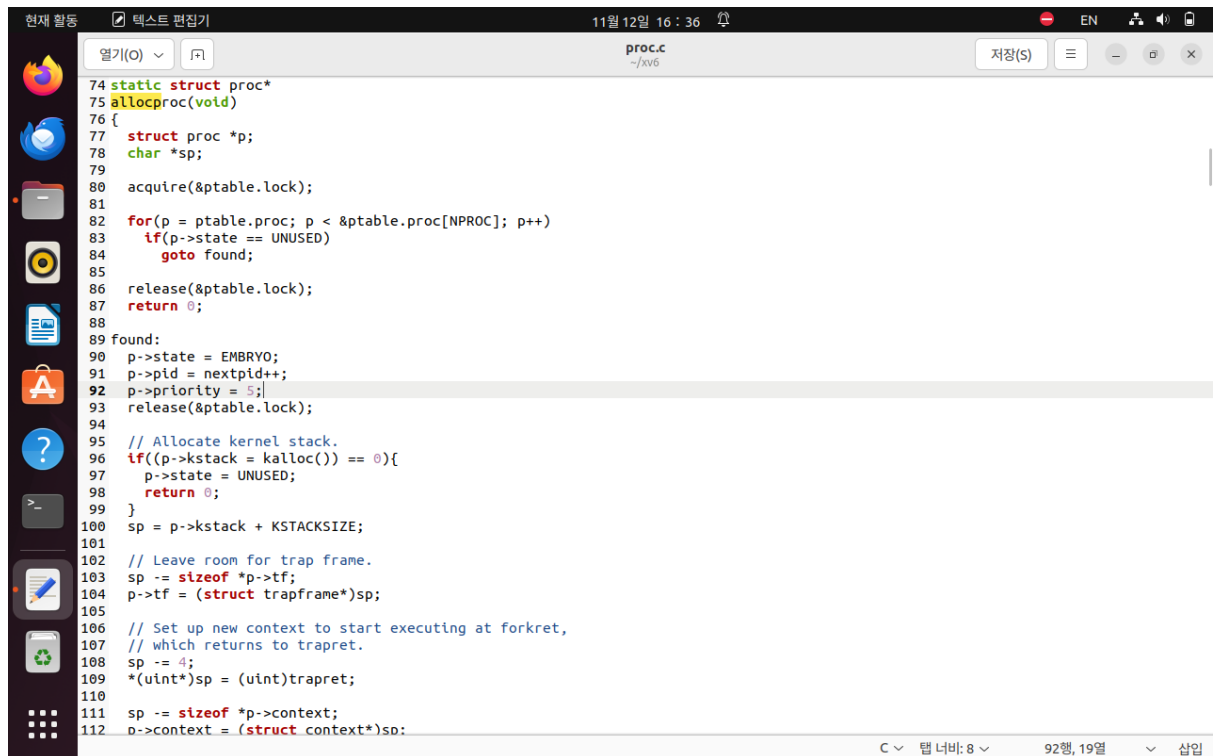
```
671 acquire(&ptable.lock);
672 np->state = RUNNABLE;
673 release(&ptable.lock);
674 wait();
675 return pid;
676 // parent process가 child status를 받으면 정상 return
677 bad:
678 if(pgdir)
679     freevm(pgdir);
680 if(ip){
681     iunlockput(ip);
682     end_op();
683 }
684 return -2; // etc err
685 }
686 }
687 */
688
689 int set_proc_priority(int id, int priority) {
690     struct proc * p;
691     if (p < 1 || p > 10) {
692         return -1;
693     }
694     for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
695         if (p->id == id) {
696             acquire(&ptable.lock);
697             p->priority = priority;
698             release(&ptable.lock);
699             return p->priority;
700         }
701     }
702     return -1;
703 }
```

set_proc_priority 와 get_proc_priority 시스템 콜을 위한 사진입니다.



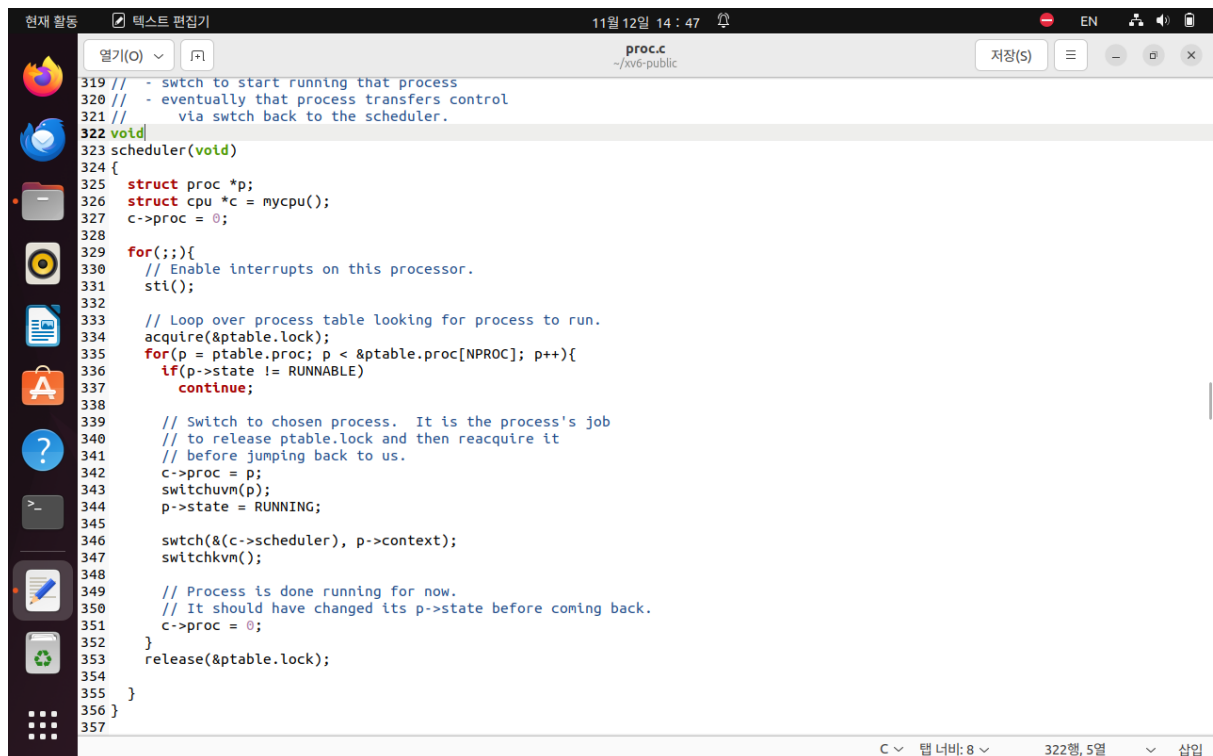
```
18 // Don't need to save all the segment registers (%cs, etc),
19 // because they are constant across kernel contexts.
20 // Don't need to save %eax, %ecx, %edx, because the
21 // x86 convention is that the caller has saved them.
22 // Contexts are stored at the bottom of the stack they
23 // describe; the stack pointer is the address of the context.
24 // The layout of the context matches the layout of the stack in switch.S
25 // at the "Switch stacks" comment. Switch doesn't save eip explicitly,
26 // but it is on the stack and allocproc() manipulates it.
27 struct context {
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     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     int priority;
53 };
54
55 // Process memory is laid out contiguously, low addresses first:
56 // text
```

proc 구조체에 priority 를 추가했습니다.



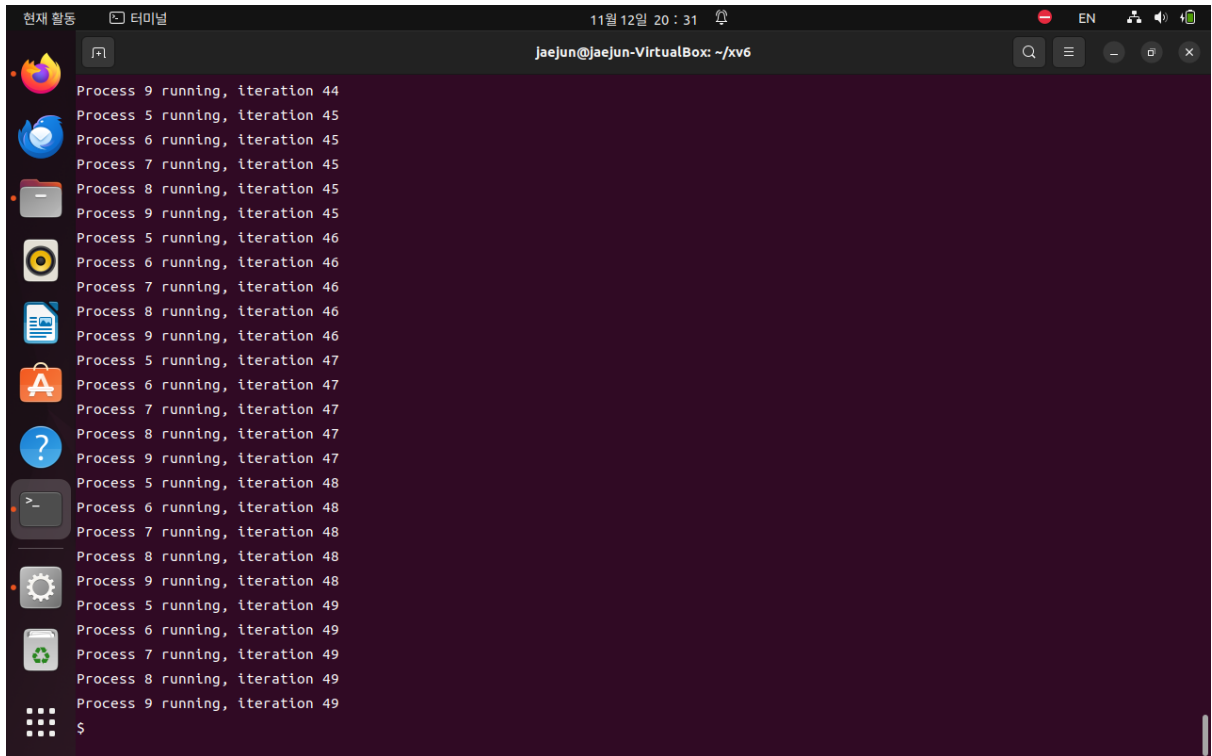
```
74 static struct proc*
75 allocproc(void)
76 {
77     struct proc *p;
78     char *sp;
79
80     acquire(&table.lock);
81
82     for(p = table.proc; p < &table.proc[NPROC]; p++)
83         if(p->state == UNUSED)
84             goto found;
85
86     release(&table.lock);
87     return 0;
88
89 found:
90     p->state = EMBRYO;
91     p->pid = nextpid++;
92     p->priority = 5;
93     release(&table.lock);
94
95     // Allocate kernel stack.
96     if((p->kstack = kalloc()) == 0){
97         p->state = UNUSED;
98         return 0;
99     }
100     sp = p->kstack + KSTACKSIZE;
101
102     // Leave room for trap frame.
103     sp -= sizeof *p->tf;
104     p->tf = (struct trapframe*)sp;
105
106     // Set up new context to start executing at forkret,
107     // which returns to trapret.
108     sp -= 4;
109     *(uint*)sp = (uint)trapret;
110
111     sp -= sizeof *p->context;
112     p->context = (struct context*)sp;
```

priority 생성시 디폴트값을 5 로설정했습니다.

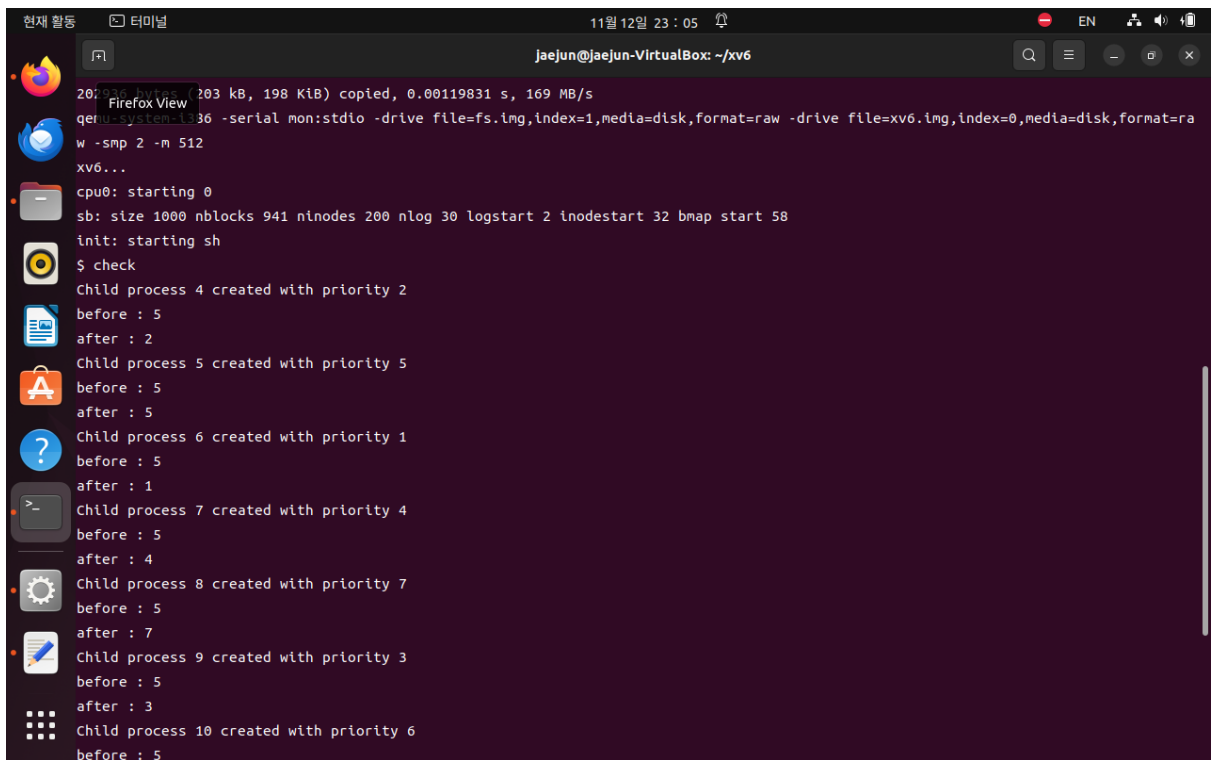


```
319 // - switch to start running that process
320 // - eventually that process transfers control
321 //   via switch back to the scheduler.
322 void
323 scheduler(void)
324 {
325     struct proc *p;
326     struct cpu *c = mycpu();
327     c->proc = 0;
328
329     for(;;){
330         // Enable interrupts on this processor.
331         sti();
332
333         // Loop over process table looking for process to run.
334         acquire(&table.lock);
335         for(p = table.proc; p < &table.proc[NPROC]; p++){
336             if(p->state != RUNNABLE)
337                 continue;
338
339             // Switch to chosen process. It is the process's job
340             // to release ptable.lock and then reacquire it
341             // before jumping back to us.
342             c->proc = p;
343             switchvm(p);
344             p->state = RUNNING;
345
346             switch(&(c->scheduler), p->context);
347             switchkvm();
348
349             // Process is done running for now.
350             // It should have changed its p->state before coming back.
351             c->proc = 0;
352         }
353         release(&table.lock);
354     }
355 }
356 }
357
```

스케줄러에 priority 를 적용한 모습입니다.



프로세스가 계속 작동하는 것을 보여주는 사진입니다.



priority 가 변경된 것을 확인할 수 있는 사진입니다.

과제 수행 중 발생한 문제점과 해결 방법

프로세스를 체크할 때 check 와 scheduler 과 번갈아가면서 수행돼 sleep()과 wati()을 이용해 해결했습니다.

참조

xv6-book-rev11, Russ Cox(2018)

xv6-source-rev11