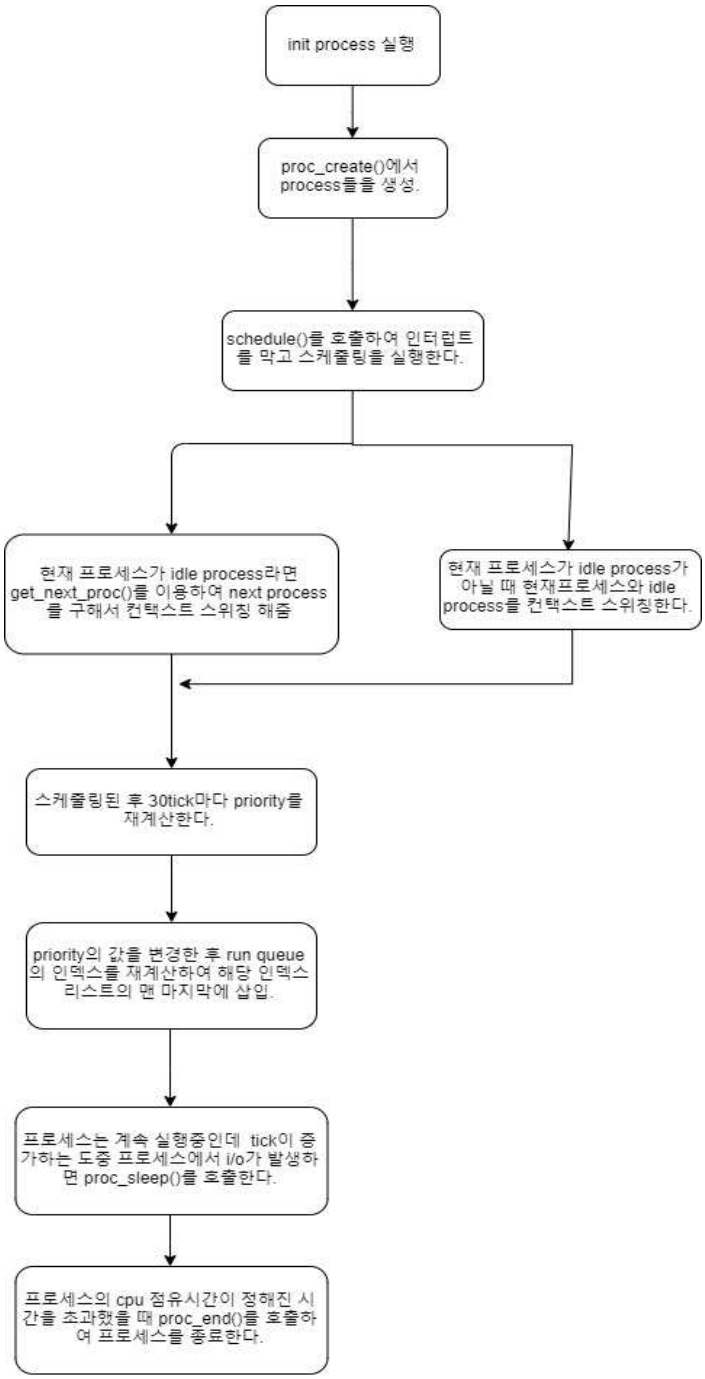
전자정보공학부 20160433 김민정

1. 개요

스케줄링(scheduling)은 다중 프로그래밍을 가능하게 하는 운영 체제의 동작 기법이다. 운영 체제는 프로세스들에게 CPU 등의 자원 배정을 적절히 함으로써 시스템의 성능을 개선할 수 있다. 이러한 실제 스케줄링 기법을 SSUOS에 구현해본다. SSUOS에 기존에 구현된 스케줄러대신 FreeBSD 5.4 이후 구현된 ULE (non- interactive)를 구현해본다. ULE 스케줄러는 과부하시 상호작용성을 증가시키고, CPU를 보다 효율적으로 스케줄링하고, 우선순위를 부여하기 위해 개발된 것이다.이 스케줄링의 구현을 위해 프로세스의 우선순위와 프로세스가 사용한 CPU 시간을 스케줄링에 반영하였다.

2.상세설계



void init_proc();
-가장 처음에 실행되어 여러 값들을 초기화시켜주는 함수이다.

pid_t proc_create(proc_func func, struct proc_option *opt, void* aux);
- kernel1~3의 프로세스를 생성시켜주는 함수이다.

void recalculate_priority(void);
-priority를 재계산하고, run queue의 인덱스를 재계산하여 해당 인덱스의 리스트의 맨 마지막에 삽입해주는 함수이다.

void proc_wake(void);
-프로세스를 깨우는 함수이다.

void proc_sleep(unsigned ticks);
-프로세스를 I/O상태(sleep)상태로 만들어주는 함수이다.

void proc_unblock(struct process* proc);
-프로세스의 상태를 PROC_RUN으로 바꿔주고 프로세스를 runq의 해당인덱스의 리스트의 제일 마지막에 삽입해준다.

void kernel1_proc(void* aux);
-커널 1의 프로세스가 언제 I/O되고 종료되는지를 설정해놓은 함수이다.

void kernel2_proc(void* aux);
-커널 2의 프로세스가 언제 I/O되고 종료되는지를 설정해놓은 함수이다.

void kernel3_proc(void* aux);
-커널 3의 프로세스가 언제 I/O되고 종료되는지를 설정해놓은 함수이다.

void proc_getpos(int priority, int *idx);
-해당 프로세스의 run queue의 인덱스를 계산해주는 함수이다.

struct process* get_next_proc(void);
-run queue 배열에서 리스트가 존재하는 곳을 찾아 그 인덱스에서 상태가 PROC_RUN인 프로세스를 next로 설정해주는 함수이다.

void schedule(void);
-스케줄을 수행하는 함수이다.

void timer_handler(struct intr_frame *iframe);
-함수가 호출될 때마다 ticks가 1씩 증가한다. 그리고 pid가 0이 아니고 스케줄링변수가 1이 아닐 때 time_slice와 time_used가 1씩 증가한다. 또한 스케줄링이 된 이후 30ticks마다 cpu 사용량에 따라 recalculate_priority()를 호출하여 priority를 재계산한다.

3. 실행결과

1) [Case 1 결과] 프로그램 수행 시 결과화면의 일부 – strings test.out 이용

(# = pid, p = 우선순위(priority), c = 스케줄 된 이후 CPU사용시간, u = 프로세스의 CPU 총 사용시간)

```
SSUOS main start!!!!

 ______   ______  __   __       ______   ______
/      \ /      \/  | /  |     /      \ /      \
$$$$$$  |$$$$$$  |$$ | $$ |    /$$$$$$  |$$$$$$  |
$$ \__$$/ $$ \__$$/ $$ | $$ |    $$ |  $$ |$$ \__$$/
$$      \ $$      \ $$ | $$ |    $$ |  $$ |$$      \
 $$$$$$  | $$$$$$  |$$ | $$ |    $$ |  $$ | $$$$$$  |
/  \__$$ |/  \__$$ |$$ \__$$ |    $$ \__$$ |/  \__$$ |
$$    $$/ $$    $$/ $$    $$/     $$    $$/ $$    $$/
 $$$$$$/   $$$$$$/   $$$$$$/       $$$$$$/   $$$$$$/
****************Made by OSLAB in SoongSil University**********************
contributors : Yunkyu Lee   , Minwoo Jang   , Sanghun Choi , Eunseok Choi
               Hyunho Ji    , Giwook Kang   , Kisu Kim     , Seonguk Lee
               Gibeom Byeon, Jeonghwan Lee, Kyoungmin Kim, Myungjoon Shon
               Jinwoo Lee, Hansol Lee, Mhanwoo Heo
************************  Professor. Jiman Hong  ************************

Memory Detecting
PIT Initialization
System call Initialization
idtr size : 2047 address : 0x40000
Timer Handler Registration
Keyboard Handler Registration
System Call Handler Registration
Interrupt Initialization
32511 pages available in memory pool.
Interrupt Initialization
Palloc Initialization
-PE=32768, PT=32
-page dir=101000 page tbl=102000
Paging Initialization
Process Initialization
========== initialization complete ==========
#= 1 p=   50 c=  0 u=  0 , #= 2 p=   50 c=  0 u=  0
Selected # = 1
#= 1 p=   59 c= 60 u= 60 , #= 2 p=   50 c=  0 u=  0
Selected # = 2
#= 1 p=   59 c= 60 u= 60 , #= 2 p=   59 c= 60 u= 60
Selected # = 1
#= 1 p=   68 c= 60 u=120 , #= 2 p=   59 c= 60 u= 60
Selected # = 2
Proc 2 I/O at 100
#= 1 p=   68 c= 60 u=120
Selected # = 1
Proc 1 I/O at 140
#= 2 p=   62 c=  0 u=100
Selected # = 2
#= 1 p=   68 c=  0 u=140
Selected # = 1
#= 1 p=   77 c= 60 u=200
Selected # = 1
```

2) [Case 2 결과] 프로그램 수행 시 결과화면의 일부 – strings test.out 이용

(# = pid, p = 우선순위(priority), c = 스케줄 된 이후 CPU사용시간, u = 프로세스의 CPU 총 사용시간)

```
        $$    $$/ $$    $$/ $$    $$/       $$    $$/ $$    $$/
         $$$$$$/   $$$$$$/   $$$$$$/         $$$$$$/   $$$$$$/
****************Made by OSLAB in SoongSil University**********************
contributors : Yunkyu Lee  , Minwoo Jang  , Sanghun Choi , Eunseok Choi
               Hyunho Ji   , Giwook Kang  , Kisu Kim     , Seonguk Lee
               Gibeom Byeon, Jeonghwan Lee, Kyoungmin Kim, Myungjoon Shon
               Jinwoo Lee, Hansol Lee, Mhanwoo Heo
************************  Professor. Jiman Hong  ************************

Memory Detecting
PIT Initialization
System call Initialization
idtr size : 2047 address : 0x40000
Timer Handler Registration
Keyboard Handler Registration
System Call Handler Registration
Interrupt Initialization
32511 pages available in memory pool.
Interrupt Initialization
Palloc Initialization
-PE=32768, PT=32
-page dir=101000 page tbl=102000
Paging Initialization
Process Initialization
========== initialization complete ==========
#= 1 p=   50 c=  0 u=  0 , #= 2 p=   50 c=  0 u=  0 , #= 3 p=   30 c=  0 u=  0
Selected # = 3
Proc 3 I/O at 50
#= 1 p=   50 c=  0 u=  0 , #= 2 p=   50 c=  0 u=  0
Selected # = 1
#= 1 p=   59 c= 60 u= 60 , #= 2 p=   50 c=  0 u=  0 , #= 3 p=   33 c=  0 u= 50
Selected # = 3
Proc 3 I/O at 100
#= 1 p=   59 c= 60 u= 60 , #= 2 p=   50 c=  0 u=  0
Selected # = 2
#= 1 p=   59 c= 60 u= 60 , #= 2 p=   59 c= 60 u= 60 , #= 3 p=   36 c=  0 u=100
Selected # = 3
#= 1 p=   59 c= 60 u= 60 , #= 2 p=   59 c= 60 u= 60
Selected # = 1
#= 1 p=   68 c= 60 u=120 , #= 2 p=   59 c= 60 u= 60
Selected # = 2
Proc 2 I/O at 100
#= 1 p=   68 c= 60 u=120
Selected # = 1
Proc 1 I/O at 140
#= 2 p=   62 c=  0 u=100
Selected # = 2
#= 1 p=   68 c=  0 u=140
Selected # = 1
#= 1 p=   77 c= 60 u=200
Selected # = 1
```

4. 소스코드

-proc.c

1)

```
void init_proc()
{
	int i;
	process_stack_ofs = offsetof (struct process, stack);

	lock_pid_simple = 0;
	lately_pid = -1;

	list_init(&plist);
	list_init(&rlist);
	list_init(&slist);

	for(int j=0;j<25;j++)
		list_init(&runq[j]);

	for (i = 0; i < PROC_NUM_MAX; i++)
	{
		procs[i].pid = i;
		procs[i].state = PROC_UNUSED;
		procs[i].parent = NULL;
	}

	pid_t pid = getValidPid(&i);
	cur_process = (int *)&procs[0];
	idle_process = (int *)&procs[0];

	cur_process->pid = pid;
	cur_process->parent = NULL;
	cur_process->state = PROC_RUN;

	cur_process -> priority = 99;

	cur_process->stack = 0;
	cur_process->pd = (void*)read_cr3();
	cur_process -> elem_all.prev = NULL;
	cur_process -> elem_all.next = NULL;
	cur_process -> elem_stat.prev = NULL;
	cur_process -> elem_stat.next = NULL;

	/* You should modify this function... */
	list_push_back(&plist, &cur_process->elem_all);
```

```
        list_push_back(&runq[24], &cur_process->elem_stat); //rlist가 아니라 runq[]에 프로세스를 삽입한다.
}
2)
pid_t proc_create(proc_func func, struct proc_option *opt, void* aux)
{
        struct process *p;
        int idx;
        enum intr_level old_level = intr_disable();

        pid_t pid = getValidPid(&idx);
        p = &procs[pid];
        p->pid = pid;
        p->state = PROC_RUN;

        if(opt != NULL)
                p -> priority = opt -> priority;
        else
                p -> priority = (unsigned char)99;

        p->time_used = 0;
        p->time_slice = 0;
        p->parent = cur_process;
        p->simple_lock = 0;
        p->child_pid = -1;
        p->pd = pd_create(p->pid);

        //init stack
        int *top = (int*)palloc_get_page();
        int stack = (int)top;
        top = (int*)stack + STACK_SIZE - 1;

        *(--top) = (int)aux;                //argument for func
        *(--top) = (int)proc_end;           //return address from func
        *(--top) = (int)func;               //return address from proc_start
        *(--top) = (int)proc_start; //return address from switch_process

        //process call stack :
        //switch_process > proc_start > func(aux) > proc_end

        *(--top) = (int)((int*)stack + STACK_SIZE - 1); //ebp
        *(--top) = 1; //eax
        *(--top) = 2; //ebx
        *(--top) = 3; //ecx
        *(--top) = 4; //edx
```

```
        *(--top) = 5; //esi
        *(--top) = 6; //edi

        p -> stack = top;
        p -> elem_all.prev = NULL;
        p -> elem_all.next = NULL;
        p -> elem_stat.prev = NULL;
        p -> elem_stat.next = NULL;

        /* You should modify this function... */
        proc_getpos(p->priority,&idx);//idx에 index값을 저장한다.
        list_push_back(&plist, &p->elem_all);
        list_push_back(&runq[idx], &p->elem_stat); //rlist 대신에 runq[]에 프로세스를 삽입한다.

        intr_set_level (old_level);
        return p->pid;
}

3)
void recalculate_priority(void)
{
        int idx;
        struct process *p = cur_process;


        /* Your code goes here... */
        p->priority += p->time_slice/10;
        proc_getpos(p->priority, &idx);//proc_getpos()이용해서 runq의 인덱스를 재계산한다.

        list_push_back(&runq[idx], &p->elem_stat);//재계산한 runq의 인덱스부분의 마지막에 프로세스를 삽입한다.
}

4)
void proc_wake(void)
{
        struct process* p;
        int idx;
        int old_level;
        unsigned long long t = get_ticks();

        /* You should modify this function... */
        while(!list_empty(&slist)) //sleep process list가 empty가 아니면
        {
                p = list_entry(list_front(&slist), struct process, elem_stat);
```

```
                if(p->time_sleep > t) // i/o가 아직 끝나지 않았을 때
                        break;

                list_remove(&p->elem_stat);//slist에서 프로세스를 삭제한다.

                proc_getpos(p->priority, &idx);//해당 프로세스가 들어갈 index를 계산
                list_push_back(&runq[idx], &p->elem_stat);//재계산한 인덱스를 이용하여 runq에 프로세스를 삽입
                p->state = PROC_RUN; //프로세스의 상태를 PROC_RUN으로 설정한다.
        }
}
```

5)

```
extern int scheduling; //extern으로 scheduling 변수 선언하였다.
void proc_sleep(unsigned ticks)
{
        scheduling = 1;//time_slice와 time_used의 값의 증가를 막기 위해 설정해놓았다.
        unsigned long cur_ticks = get_ticks();

        /* You should modify this function... */
        printk("Proc %d I/O at %d\n", cur_process->pid, cur_process->time_used);//어떤 프로세스가 I/O중인지를
알려주기 위해 출력문을 삽입했다.
        cur_process->time_sleep =  ticks + cur_ticks;//깨어날 시간을 설정하였다.
        cur_process->state = PROC_STOP;//프로세스의 상태를 PROC_STOP으로 설정한다.

        cur_process->time_slice = 0;

        list_remove(&cur_process->elem_stat);
        list_insert_ordered(&slist, &cur_process->elem_stat,less_time_sleep, NULL);
        schedule();
}
```

6)

```
void proc_unblock(struct process* proc)
{
        enum intr_level old_level;
        int idx;

        proc_getpos(proc->priority, &idx); //프로세스의 인덱스를 재계산해준다.
        /* You shoud modify this function... */
        list_push_back(&runq[idx], &proc->elem_stat); //재계산한 인덱스를 이용하여 runq에 프로세스를 삽입
        proc->state = PROC_RUN;
}
```

7)

```
void kernel1_proc(void* aux)
{
	int passed = 0;
	while(1)
	{
		/* Your code goes here... */
		if((cur_process->time_used >= 140) && (!passed)){ // i/o가 발생한 생태이고 kernel1이 실행된 적 없
으면
			proc_sleep(60);
			passed = 1;
		}

		if(cur_process->time_used >= 200){ // i/o를 제외한 총 수행시간이 200초 이상이면 프로세스를 종료한
다.
			proc_end();
		}

	}

}
```

8)

```
void kernel2_proc(void* aux)
{
	int passed = 0;
	while(1)
	{
		/* Your code goes here... */
		if((cur_process->time_used >= 100) && (!passed)){ // i/o가 발생한 생태이고 kernel2가 실행된 적 없
으면
			proc_sleep(60);
			passed = 1;
		}

		if(cur_process->time_used >= 120){ // i/o를 제외한 총 수행시간이 120초 이상이면 프로세스를 종료한
다.
			proc_end();
		}
	}
}
```

9)

```
void kernel3_proc(void* aux)
```

```
{
        int passed1 = 0, passed2 = 0;

        while(1)
        {
                /* Your code goes here... */
                if((cur_process->time_used >= 50) && (!passed1)){ // 첫 번째 i/o가 발생한 상태이고 kernel3이 실행
된 적 없으면
                        proc_sleep(60);
                        passed1 = 1;
                }

                if((cur_process->time_used >= 100) && (!passed2)){ // 두 번째 i/o가 발생한 상태이고 kernel3이 실
행된 적 없으면
                        proc_sleep(60);
                        passed2 = 1;
                }

                if(cur_process->time_used >= 150){ // i/o를 제외한 총 수행시간이 150초 이상이면 프로세스를 종료한
다.
                        proc_end();
                }
        }
}
```

10)

```
void proc_getpos(int priority, int *idx) //해당 프로세스의 run queue의 인덱스를 재계산하는 함수
{
        *idx = priority/4;
}
```

-sched.c

11)

```
struct process* get_next_proc(void) {

        bool found = false;
        struct process *next = NULL;
        struct list_elem *elem;

        /*
           You shoud modify this function...
           Browse the 'runq' array
         */
```

```
        for(int i=0;i<RQ_NQS;i++){ //runq배열을 처음부터 돈다
                if(list_empty(&runq[i])) //각 배열 안에 list가 존재하지 않으면 continue;
                        continue;

                for(elem = list_begin(&runq[i]); elem != list_end(&runq[i]); elem = list_next(elem)) //runq[i]에 list
가 존재하면 그 list들을 처음부터 돈다.
                {
                        struct process *p = list_entry(elem, struct process, elem_stat);

                        if(p->state == PROC_RUN) //상태가 PROC_RUN인 것을 찾아서 리턴한다.
                                return p;
                }
        }

        return next;
}

12)
void schedule(void)
{
        struct process *cur;
        struct process *next;
        struct process *tmp;
        struct list_elem *elem;

        int printed = 0;
        /* You shoud modify this function.... */
        intr_set_level(0); // context switching 도중에는 인터럽트핸들러가 수행되지 않도록 함
        scheduling = 1;

        cur = cur_process;
        if(cur_process->pid != 0){ //현재 프로세스가 idle process가 아닐 때
                cur_process=idle_process;//idle_process를 현재 프로세스로 설정한다.
                latest=cur;//cur 프로세스가 가장 최근에 cpu를 점유한 프로세스이므로 latest에 값을 저장하고
                switch_process(cur,idle_process);//cur과 idle_process를 context switching한다.
                intr_set_level(1); //context switching이 끝났으니 인터럽트 핸들러가 수행되도록 한다.
                return;
        }

        proc_wake();

        for (elem = list_begin(&plist); elem != list_end(&plist); elem = list_next(elem)) { //plist를 처음부터 돈다.
                tmp = list_entry (elem, struct process, elem_all);
```

```
                //프로세스의 상태가 PROC_ZOMBIE, PROC_BLOCK, PROC_STOP, PROC_UNUSED인 경우는 continue.
                if ((tmp -> state == PROC_ZOMBIE) || (tmp -> state == PROC_BLOCK) || (tmp -> state == PROC_STOP) || (tmp -> state == PROC_UNUSED) || (tmp -> pid == 0))
                        continue;
                if (!printed) {
                        printk("#=%2d p= %4d c=%3d u=%3d ", tmp -> pid, tmp->priority, tmp -> time_slice, tmp->time_used);
                        printed=1;
                }
                else{
                        printk(", #=%2d p= %4d c=%3d u=%3d ", tmp -> pid, tmp->priority, tmp -> time_slice, tmp->time_used);
                }

        }
        if (printed)
                printk("\n");

        if((next = get_next_proc())!= NULL){ // 지금 프로세스의 다음에 CPU를 점유할 프로세스를 찾아서 next에 저장한다.
                if(next->pid != 0)
                        printk("Selected # = %d\n", next->pid); //next에 저장된 프로세스가 무엇인지 알려준다.
                cur_process = next; //현재 프로세스를 바꿔준다.
                cur_process->time_slice = 0; //time_slice를 초기화한다.
                scheduling=0;
                switch_process(cur, next);//cur과 next를 context switching한다.
                intr_set_level(1); //context switching이 끝났으므로 인터럽트 핸들러가 수행되도록 한다.
                return;
        }
        return;

}
```

-interrupt.c

13)

```
void timer_handler(struct intr_frame *iframe)
{
        ticks++; //함수가 호출될때마다 계속 1씩 증가한다.
        if((cur_process -> pid != 0) && (!scheduling)){ // pid가 0이 아니고 scheduling이 0일 때
                cur_process->time_slice++; //매 클럭 ticks마다 time_slice를 1씩 증가시킨다.
                cur_process->time_used++; //매 클럭 ticks마다 time_used를 1씩 증가시킨다.

                //스케줄링 된 이후 매 30tick마다 cpu 사용량에 따라 priority를 재계산한다..
                if(cur_process -> time_slice%30 == 0) {
```

```
                recalculate_priority();

        }

        if(cur_process->time_slice >= TIMER_MAX)//기본적으로 60 ticks마다 스케줄링된다.
                do_sched_on_return();
    }
#ifdef SCREEN_SCROLL
    static unsigned long refresh_ticks = 0;
    if(++refresh_ticks >= (PIT_FRQ_HZ/REFRESH_FPS)) {
        refresh_ticks = 0;
        refreshScreen();
    }
#endif
}
```