컴퓨터학부 20142385 엄재식

1. 과제 개요

Multi-level Feedback Queue 스케줄링 구현, 0~2 level의 큐로 구성되있다.

스케줄링을 할 때 1번 큐를 우선으로 프로세스를 찾고 1번 큐에 없다면 2번 큐에서 프로세스를 찾는다. Time quantum을 모두 소비하면 0번 큐에서 0번 프로세스로 이동해 schedule 함수를 호출해 스케줄링을 한다.

1. 0번큐

0번 프로세스를 위한 큐, 0번 프로세스는 idle 프로세스로 schedule 함수만 호출하도록 구현

1. 1번큐

0번큐를 제외한 프로세스 생성시 1번큐로 이동, i/o를 마친 프로세스도 1번큐로 이동 (FIFO)

1번큐에서 스케줄링 된 프로세스는 40tick의 time quantum을 할당함

1. 2번큐

1번큐에서 스케줄링을 마친 프로세스가 2번큐로 이동 (Round Robin)

1. 상세설계
2. sched.c

sched\_find\_que 함수에서 list\_empty함수를 사용해 검색할 큐를 찾은 다음 get\_next\_proc함수를 사용해 스케줄링할 프로세스를 검색하도록 코드를 추가함.

get\_next\_proc 함수에서 큐에서 프로세스를 건내 줄 때 list\_remove 함수를 사용해 큐에서 pop해주는 코드를 추가함.

schedule

1. 호출한 프로세스가 0번 프로세스가 아닐 경우 0번 프로세스로 바꿔주는 코드를 추가함
2. 스케줄링 할 때마다 latest를 갱신하여 이전에 스케줄링 된 프로세스가 무엇인지 알게함, lastest의 큐 레벨을 보고 1이면 큐 레벨을 낮추어 2번 큐로 push할 수 있게 만들고 2이면 다시 2번 큐로 push하도록 코드를 추가함
3. sched\_find\_que 함수를 이용해 다음 프로세스 찾은 switch\_process 함수를 이용해 context switching을 한다.

proc\_que\_levelup 함수에서 프로세스 큐의 레벨을 1로 바꿔주고 1번 큐에 push하도록 코드를 추가함

proc\_que\_leveldown 함수에서 프로세스 큐의 레벨을 2로 바꿔주고 2번 큐에 push하도록 코드를 추가함

1. proc.c

proc\_create 함수에서 프로세스를 생성하고 1번 큐로 push하도록 코드를 추가함

proc\_end 와 proc\_sleep 함수가 시작할 때 intr\_disable을 추가해 종료나 i/o 할 동안에 time quantum이 끝나지 않도록 함

proc\_wake 함수에서 i/o가 끝난 함수가 1번큐로 이동하도록 proc\_que\_levelup 함수를 호출함

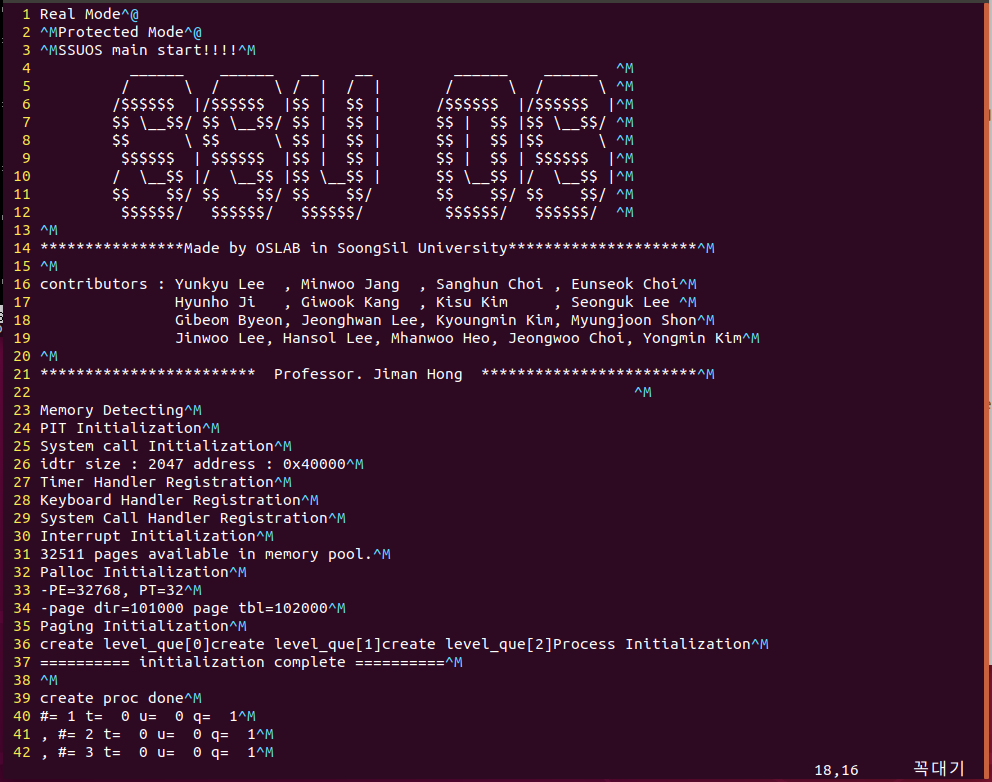
1. interrupt.c

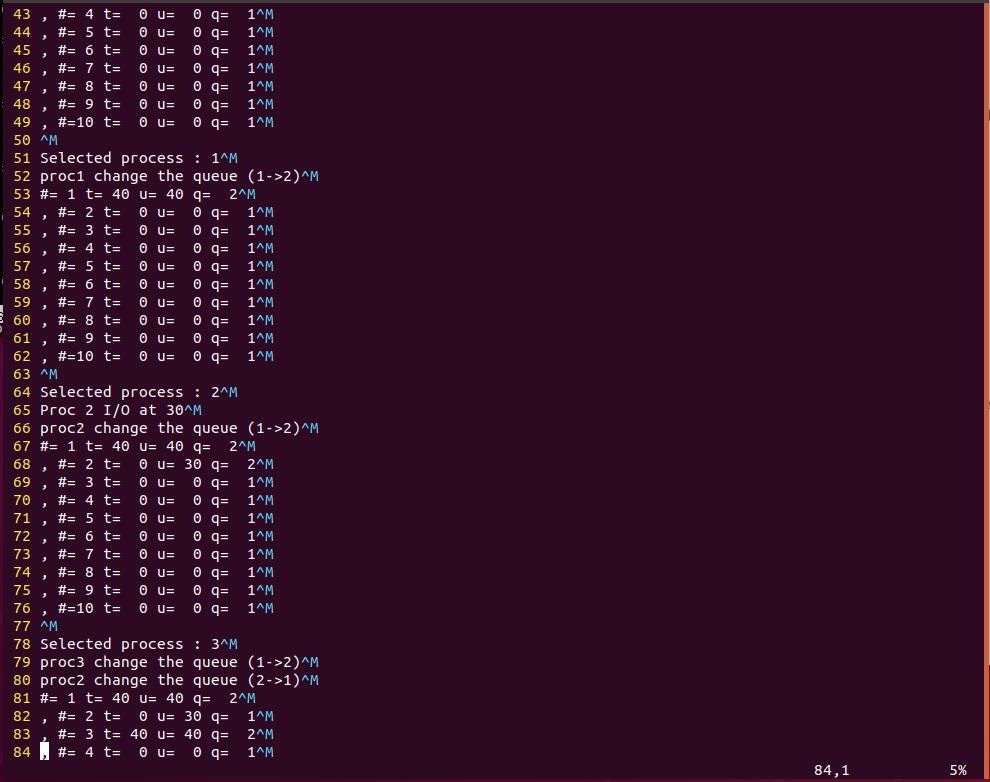
timer\_handler 함수에서 time\_slice를 먼저 검사한 뒤 time\_used와 time\_slice를 증가시키도록 수정하여 i/o 시작 시간과 스케줄 시간이 겹칠 때 i/o 먼저하고 time quantum을 끝내도록 만듬

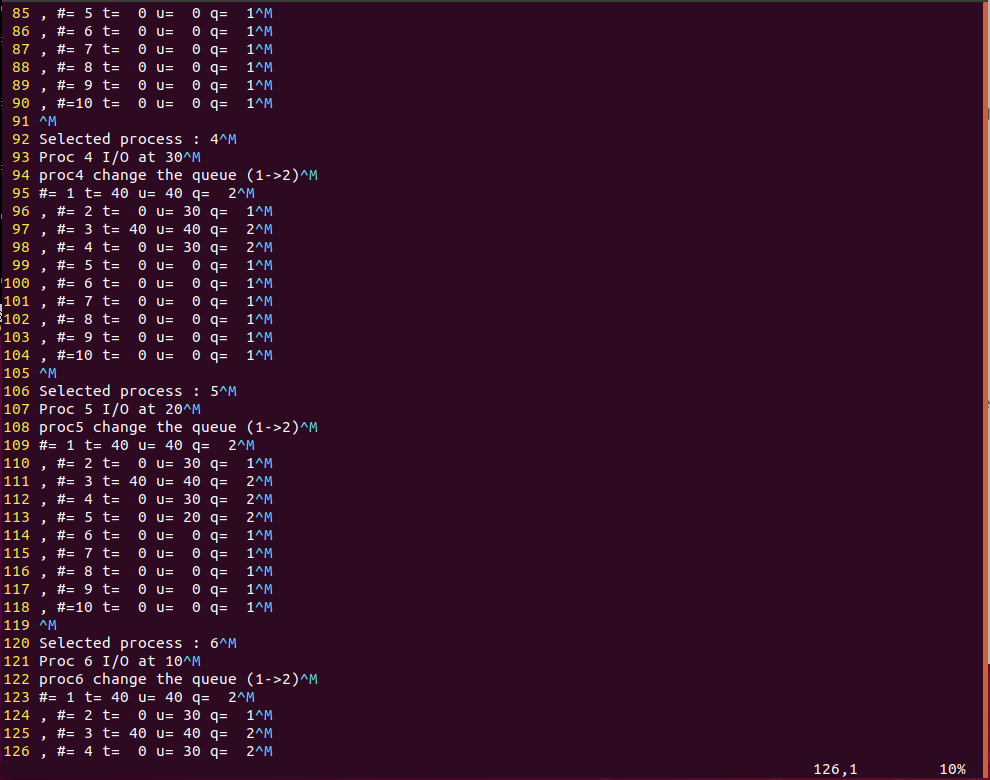
1. 시뮬레이션

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PID**  **(순서)** | **시작시각**  **(tick)** | **종료시각**  **(tick)** | **I/O 제외**  **총 수행시간**  **(tick)** | **I/O 시작 시각**  **(tick)** |
| **1** | **0** | **1210** | **200** | **640** |
| **2** | **40** | **830** | **120** | **70** |
| **3** | **70** | **2180** | **300** | **700** |
| **4** | **110** | **3010** | **300** | **140, 990** |
| **5** | **140** | **3350** | **400** | **160, 1110** |
| **6** | **160** | **2590** | **200** | **170. 340, 980, 1900,** |
| **7** | **170** | **3630** | **500** | **190, 360, 1130** |
| **8** | **190** | **3850** | **600** | **210** |
| **9** | **210** | **3950** | **750** | **240** |
| **10** | **240** | **4030** | **800** | **1450** |

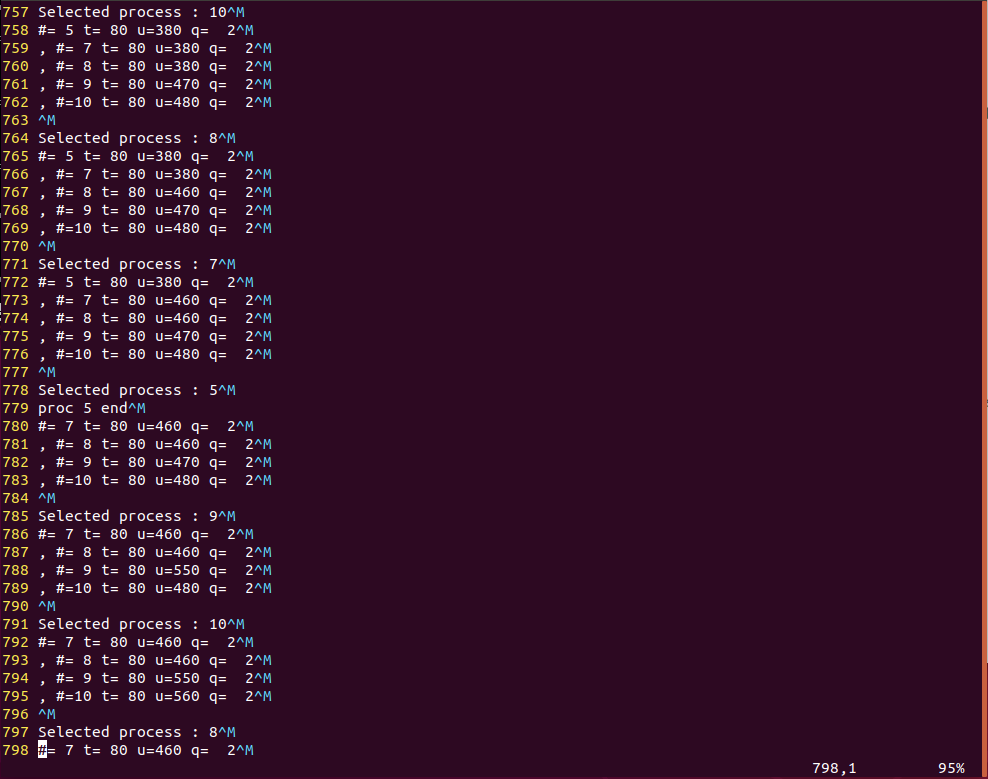
1. 실행결과

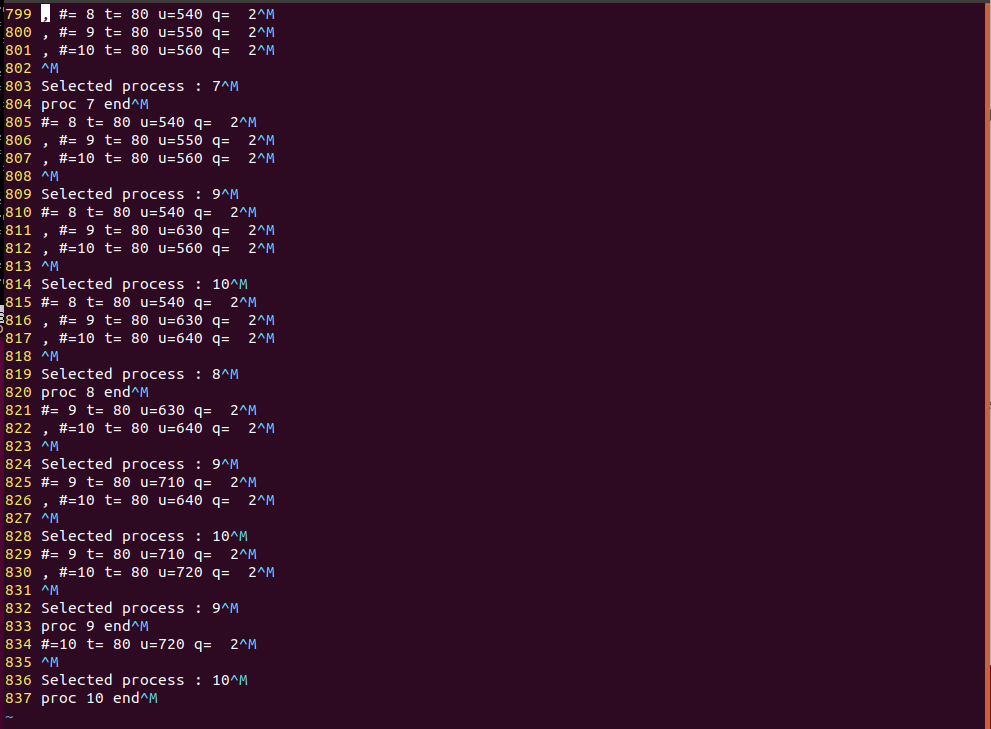






중간생략…





1. 소스코드

sched.c

#include <list.h>  
#include <proc/sched.h>  
#include <mem/malloc.h>  
#include <proc/proc.h>  
#include <proc/switch.h>  
#include <interrupt.h>  
#include <device/console.h>  
  
extern struct list level\_que[QUE\_LV\_MAX];  
extern struct list plist;  
extern struct list slist;  
extern struct process procs[PROC\_NUM\_MAX];  
extern struct process \*idle\_process;  
  
struct process \*latest;  
  
struct process\* get\_next\_proc(struct list \*rlist\_target);  
void proc\_que\_levelup(struct process \*cur);  
void proc\_que\_leveldown(struct process \*cur);  
struct process\* sched\_find\_que(void);  
  
int scheduling;  
  
/\*  
    linux multilevelfeedback queue scheduler  
    level 1 que policy is FCFS(First Come, First Served)  
    level 2 que policy is RR(Round Robin).  
\*/  
  
//sched\_find\_que find the next process from the highest queue that has proccesses.  
struct process\* sched\_find\_que(void) {  
    int i,j;  
    struct process \* result = NULL;  
       
    proc\_wake();  
  
        /\*TODO :check the queue whether it is empty or not    
         and find each queue for the next process.  
        \*/  
    while(result == NULL){  
        if(!list\_empty(&level\_que[1]))//1번 큐 찾기  
            result = get\_next\_proc(&level\_que[1]);  
        else if(!list\_empty(&level\_que[2]))//2번 큐 찾기  
            result = get\_next\_proc(&level\_que[2]);      
    }  
    return result;  
}  
  
struct process\* get\_next\_proc(struct list \*rlist\_target) {  
    struct list\_elem \*e;  
  
    for(e = list\_begin (rlist\_target); e != list\_end (rlist\_target);  
        e = list\_next (e))  
    {  
        struct process\* p = list\_entry(e, struct process, elem\_stat);  
          
        if(p->state == PROC\_RUN)  
        {  
            list\_remove(e);//리스트에서 뺌  
            return p;  
        }  
    }  
    return NULL;  
}  
  
void schedule(void)  
{  
    struct process \*cur;  
    struct process \*next;  
    struct process \*tmp;  
    struct list\_elem \*ele;  
    int i = 0, printed = 0;  
  
    scheduling = 1;      
    cur = cur\_process;  
    /\*TODO : if current process is idle\_process(pid 0), schedule() choose the next process (not pid 0).  
    when context switching, you can use switch\_process().    
    if current process is not idle\_process, schedule() choose the idle process(pid 0).  
    complete the schedule() code below.  
    \*/  
  
    if ((cur -> pid) != 0) {  
        //0번 프로세스가 아니면 0번 프로세스로 바꿈  
        ele = list\_begin(&level\_que[0]);  
        next = list\_entry(ele, struct process, elem\_stat);  
        cur\_process = next;  
        switch\_process(cur, next);  
        return;  
    }  
          
        switch (latest -> que\_level){  
            case 1:  
                //이전 프로세스가 1번큐였으면 2번큐로 바꿈  
                proc\_que\_leveldown(latest);  
                break;  
            case 2:  
                //이전 프로세스가 2번큐였으면 2번큐로 다시 푸쉬  
                if(latest->state!=PROC\_ZOMBIE&&latest->state!=PROC\_STOP)  
                    list\_push\_back(&level\_que[2], &latest->elem\_stat);  
                break;  
        }  
    proc\_wake(); //wake up the processes   
      
    //print the info of all 10 proc.  
    for (ele = list\_begin(&plist); ele != list\_end(&plist); ele = list\_next(ele)) {  
        tmp = list\_entry (ele, struct process, elem\_all);  
        if ((tmp -> state == PROC\_ZOMBIE) ||   
            //(tmp -> state == PROC\_BLOCK) ||   
            //    (tmp -> state == PROC\_STOP) ||  
                    (tmp -> pid == 0))     continue;  
            if (!printed) {      
                printk("#=%2d t=%3d u=%3d ", tmp -> pid, tmp -> time\_slice, tmp -> time\_used);  
                printk("q=%3d\n", tmp->que\_level);  
                printed = 1;              
            }  
            else {  
                printk(", #=%2d t=%3d u=%3d ", tmp -> pid, tmp -> time\_slice, tmp->time\_used);  
                printk("q=%3d\n", tmp->que\_level);  
                }  
              
    }  
    //printk("que 1 size %d\n", list\_size(&level\_que[1]));  
    //printk("que 2 size %d\n", list\_size(&level\_que[2]));  
    if (printed)  
        printk("\n");  
    if ((next = sched\_find\_que()) != NULL) {  
        printk("Selected process : %d\n", next -> pid);      
        //프로세스를 next 프로세스로 바꿈          
        intr\_enable();      
        next->time\_slice = 0;  
        latest = next;  
        scheduling = 0;  
        cur\_process = next;  
        switch\_process(cur, next);  
        //프로세스를 0번 프로세스로 바꿈  
        ele = list\_begin(&level\_que[0]);  
        next = list\_entry(ele, struct process, elem\_stat);  
        cur\_process = next;  
        switch\_process(cur, next);  
        intr\_disable();  
        return;  
    }  
    return;  
}  
  
void proc\_que\_levelup(struct process \*cur)  
{  
    /\*TODO : change the queue lv2 to queue lv1.\*/  
    //큐레벨을 1로 바꿔준 다음 1번큐에 푸쉬  
    printk("proc%d ", cur->pid);  
    printk("change the queue (2->1)\n");  
    cur->que\_level = 1;  
    list\_push\_back(&level\_que[1], &cur->elem\_stat);  
}  
  
void proc\_que\_leveldown(struct process \*cur)  
{  
    /\*TODO : change the queue lv1 to queue lv2.\*/  
    //큐레벨을 2로 바꿔준 다음 2번큐에 푸쉬      
    printk("proc%d ", cur->pid);  
    printk("change the queue (1->2)\n");  
    cur->que\_level = 2;  
    if(cur->state!=PROC\_ZOMBIE&&cur->state!=PROC\_STOP)  
        list\_push\_back(&level\_que[2], &cur->elem\_stat);  
}

proc.c

#include <list.h>  
#include <proc/sched.h>  
#include <mem/malloc.h>  
#include <proc/proc.h>  
#include <ssulib.h>  
#include <interrupt.h>  
#include <proc/sched.h>  
#include <syscall.h>  
#include <mem/palloc.h>  
#include <device/console.h>  
  
  
#define STACK\_SIZE 512  
  
struct list plist;                  
struct list level\_que[QUE\_LV\_MAX];  
struct list slist;                  
struct list dlist;                  
struct process procs[PROC\_NUM\_MAX];   
struct process \*cur\_process;  
struct process \*idle\_process;   
int pid\_num\_max;  
uint32\_t process\_stack\_ofs;   
static int lock\_pid\_simple;   
static int lately\_pid;   
  
bool more\_prio(const struct list\_elem \*a, const struct list\_elem \*b, void \*aux);  
bool less\_time\_sleep(const struct list\_elem \*a, const struct list\_elem \*b, void \*aux);  
pid\_t getValidPid(int \*idx);  
void proc\_start(void);  
void proc\_end(void);  
  
  
void kernel1\_proc(void \*aux);  
void kernel2\_proc(void \*aux);  
void kernel3\_proc(void \*aux);  
void kernel4\_proc(void \*aux);  
void kernel5\_proc(void \*aux);  
void kernel6\_proc(void \*aux);  
void kernel7\_proc(void \*aux);  
void kernel8\_proc(void \*aux);  
void kernel9\_proc(void \*aux);  
void kernel10\_proc(void \*aux);  
  
extern int scheduling;  
  
void init\_proc()  
{  
    int i, j;  
    process\_stack\_ofs = offsetof (struct process, stack);   
  
    lock\_pid\_simple = 0;  
    lately\_pid = -1;  
  
    list\_init(&plist);  
  
    for (i = 0; i < 3 ; i++) {  
            list\_init(&level\_que[i]);  
            printk("create level\_que[%d]", i);  
    }  
  
    list\_init(&slist);  
    list\_init(&dlist);  
  
    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 = &procs[0];  
    idle\_process = &procs[0]; //procs 0 is always idle  
  
    cur\_process->pid = pid;  
    cur\_process->parent = NULL;  
    cur\_process->state = PROC\_RUN;  
  
    cur\_process -> nice = 0;  
    cur\_process -> rt\_priority = 0;  
    cur\_process -> priority = cur\_process -> nice + cur\_process -> rt\_priority;  
  
    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;  
  
    cur\_process->que\_level = 0 ;  
    list\_push\_back(&plist, &cur\_process->elem\_all);  
    list\_push\_back(&level\_que[0], &cur\_process->elem\_stat);  
  
}  
  
pid\_t getValidPid(int \*idx) {  
    pid\_t pid = -1;  
    int i;  
  
    while(lock\_pid\_simple);  
  
    lock\_pid\_simple++;  
  
    for(i = 0; i < PROC\_NUM\_MAX; i++)  
    {  
        int tmp = i + lately\_pid + 1;  
        if(procs[tmp % PROC\_NUM\_MAX].state == PROC\_UNUSED) {   
            pid = lately\_pid + 1;  
            \*idx = tmp % PROC\_NUM\_MAX;  
            break;  
        }  
    }  
  
    if(pid != -1)  
        lately\_pid = pid;      
  
    lock\_pid\_simple = 0;  
  
    return pid;  
}  
  
pid\_t proc\_create(proc\_func func, struct proc\_option \*opt, void\* aux)  
{  
    struct process \*p;  
    int idx;  
    int i,j;  
  
    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 -> nice = opt -> nice; // not use.  
        p -> rt\_priority = opt -> rt\_priority;   //not use.  
    }  
    else {  
        p -> nice = 20;  
        p -> rt\_priority = (unsigned char)45;  
    }  
  
    p -> priority = p -> nice + p -> rt\_priority; //not use.  
    p -> que\_level = 1;   
    p -> old\_proc = 0;  
    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;  
  
    list\_push\_back(&plist, &p->elem\_all);  
    //큐레벨에 프로세스 푸쉬  
    list\_push\_back(&level\_que[p->que\_level], &p->elem\_stat);  
     //TODO : when create first , input the level 1 queue  
  
    intr\_set\_level (old\_level);  
    return p->pid;  
}  
  
void\* getEIP()  
{  
    return \_\_builtin\_return\_address(0);  
}  
  
void  proc\_start(void)  
{  
    intr\_enable ();  
    return;  
}  
  
void proc\_free(void)  
{  
    uint32\_t pt = \*(uint32\_t\*)cur\_process->pd;  
    cur\_process->parent->child\_pid = cur\_process->pid;  
    cur\_process->parent->simple\_lock = 0;  
  
    cur\_process->state = PROC\_ZOMBIE;  
    list\_push\_back(&dlist, &cur\_process->elem\_stat);   
  
    palloc\_free\_page(cur\_process->stack);   
    palloc\_free\_page((void\*)pt);  
    palloc\_free\_page(cur\_process->pd);  
  
    list\_remove(&cur\_process->elem\_stat);  
    list\_remove(&cur\_process->elem\_all);  
}  
  
void proc\_end(void)  
{  
    intr\_disable();//종료할거니까 멈춰라  
    proc\_free();  
    printk("proc %d end\n", cur\_process->pid);  
    schedule();  
    return;  
}  
  
void proc\_wake(void)  
{  
    struct process\* p;  
    int que\_level;  
    int old\_level;  
    unsigned long long t = get\_ticks();  
    struct list\_elem \*e;  
        for(e = list\_begin (&slist); e != list\_end (&slist);  
        e = list\_next (e))  
    {  
        struct process\* p = list\_entry(e, struct process, elem\_stat);  
    }  
       while(!list\_empty(&slist))  
    {  
            p = list\_entry(list\_front(&slist), struct process, elem\_stat); //pop the sleep list of sleeping proc  
        if(p->time\_sleep > t)  
            break;  
  
        list\_remove(&p->elem\_stat);      
        //i/o가 끝나면 1번큐로 바꿈          
        proc\_que\_levelup(p);  
        p->state = PROC\_RUN;  
    }  
      
}  
  
void proc\_sleep(unsigned ticks)  
{  
    intr\_disable();//i/o 할거니까 멈춰라  
    unsigned long cur\_ticks = get\_ticks();  
    //프로세스 번호와 i/o 출력      
    scheduling = 1;  
    printk("Proc %d ",cur\_process->pid);  
    printk("I/O at %d\n", cur\_process->time\_used);  
    scheduling = 0;  
    cur\_process->time\_sleep =  ticks + cur\_ticks;  
    cur\_process->state = PROC\_STOP;  
    cur\_process->time\_slice = 0;  
    struct list\_elem \*e;  
      
    list\_remove(&cur\_process->elem\_stat); //remove queue list  
  
    list\_insert\_ordered(&slist, &cur\_process->elem\_stat,  
            less\_time\_sleep, NULL); //order by less time sleep   
  
    list\_sort (&slist,less\_time\_sleep, NULL);  
    schedule();  
}  
  
void proc\_block(void) //io   
{  
    cur\_process->state = PROC\_BLOCK;  
    schedule();      
}  
  
void proc\_unblock(struct process\* proc)  
{  
    enum intr\_level old\_level;  
    list\_push\_back(&level\_que[proc->que\_level],&proc->elem\_stat);  
    proc->state = PROC\_RUN;  
}       
  
bool less\_time\_sleep(const struct list\_elem \*a, const struct list\_elem \*b,void \*aux)  
{  
    struct process \*p1 = list\_entry(a, struct process, elem\_stat);  
    struct process \*p2 = list\_entry(b, struct process, elem\_stat);  
  
    return p1->time\_sleep < p2->time\_sleep;  
}  
  
bool more\_prio(const struct list\_elem \*a, const struct list\_elem \*b,void \*aux)  
{  
    struct process \*p1 = list\_entry(a, struct process, elem\_stat);  
    struct process \*p2 = list\_entry(b, struct process, elem\_stat);  
      
    return p1->priority > p2->priority;  
}  
  
void kernel1\_proc(void\* aux)  
{  
    int passed = 0;  
    while(1)  
    {  
        if ((cur\_process -> time\_used >= 80) && (!passed)) {  
          
            proc\_sleep(60);  
            passed = 1;  
              
        }  
        if (cur\_process -> time\_used >= 200)  
            proc\_end();      
    }  
}  
  
void kernel2\_proc(void\* aux)  
{  
    int passed = 0;  
    while(1)  
    {  
        if ((cur\_process -> time\_used >= 30) && (!passed)) {  
            proc\_sleep(30);  
            passed = 1;  
              
        }  
  
        if (cur\_process -> time\_used >= 120) {  
            proc\_end();  
        }  
    }  
}  
  
void kernel3\_proc(void\* aux)  
{  
    int passed = 0;  
  
  
    while(1)  
    {  
        if ((cur\_process -> time\_used >= 100) && (!passed)) {  
            proc\_sleep(200);  
            passed = 1;  
          
        }  
  
        if (cur\_process -> time\_used >= 300) {  
            proc\_end();  
        }  
    }  
}  
  
void kernel4\_proc(void\* aux)  
{  
    int passed1=0, passed2 = 0 ;  
  
    while(1)  
    {  
        if ((cur\_process -> time\_used >= 30) && (!passed1)){  
            proc\_sleep(250);  
            passed1 = 1 ;  
          
        }  
          
        if ((cur\_process -> time\_used >= 80) && (!passed2)){  
            proc\_sleep(300);  
            passed2 =1;  
              
        }  
  
        if (cur\_process -> time\_used >=300){  
            proc\_end();  
        }  
    }  
   
}  
void kernel5\_proc(void\* aux)  
{  
    int passed1=0, passed2 = 0 ;  
    while(1)  
    {  
        if ((cur\_process -> time\_used >= 20) && (!passed1)){  
            proc\_sleep(350);  
            passed1 = 1;  
              
        }  
        if ((cur\_process -> time\_used >= 100) && (!passed2)){  
            proc\_sleep(50);  
            passed2 = 1;  
          
        }  
        if (cur\_process -> time\_used >=400)  
            proc\_end();  
    }  
}  
  
void kernel6\_proc(void\* aux)  
{  
    int passed1=0, passed2 = 0, passed3 = 0, passed4 = 0;  
    while(1)  
    {  
        if ((cur\_process -> time\_used >= 10) && (!passed1)){  
            proc\_sleep(10);  
            passed1 = 1;  
              
        }  
        if ((cur\_process -> time\_used >= 30) && (!passed2)){  
            proc\_sleep(20);  
            passed2 = 1;  
              
        }  
        if ((cur\_process -> time\_used >= 100) && (!passed3)){  
            proc\_sleep(200);  
            passed3 = 1;  
          
        }  
        if ((cur\_process -> time\_used >= 150) && (!passed4)){  
            proc\_sleep(10);  
            passed4 = 1;  
              
        }  
  
        if (cur\_process -> time\_used >=200)  
            proc\_end();  
    }  
}  
  
void kernel7\_proc(void\* aux)  
{  
    int passed1=0, passed2 = 0, passed3 = 0;  
    while(1)  
    {  
        if ((cur\_process -> time\_used >= 20) && (!passed1)){  
            proc\_sleep(20);  
            passed1 = 1;  
              
        }  
        if ((cur\_process -> time\_used >= 40) && (!passed2)){  
            proc\_sleep(200);  
            passed2 = 1;  
              
        }  
        if ((cur\_process -> time\_used >= 100) && (!passed3)){  
            proc\_sleep(10);  
            passed3 = 1;  
              
        }  
        if (cur\_process -> time\_used >=500)  
            proc\_end();  
    }  
}  
  
void kernel8\_proc(void\* aux)  
{  
    int passed = 0;  
    while(1)  
    {  
        if ((cur\_process -> time\_used >= 20) && (!passed)){  
            proc\_sleep(200);  
            passed = 1;  
              
        }  
  
        if (cur\_process -> time\_used >=600)  
            proc\_end();  
    }  
}  
  
void kernel9\_proc(void\* aux)  
{  
    int passed = 0;  
    while(1)  
    {  
        if ((cur\_process -> time\_used >= 30) && (!passed)){  
            proc\_sleep(20);  
            passed = 1;  
              
        }  
  
        if (cur\_process -> time\_used >=750)  
            proc\_end();  
    }  
}  
  
void kernel10\_proc(void\* aux)  
{  
    int passed = 0;  
    while(1)  
    {  
        if ((cur\_process -> time\_used >= 200) && (!passed)){  
            proc\_sleep(100);  
            passed = 1;  
              
        }  
  
        if (cur\_process -> time\_used >=800)  
            proc\_end();  
    }  
}  
  
void idle(void\* aux)  
{  
    proc\_create(kernel1\_proc, NULL, NULL);  
    proc\_create(kernel2\_proc, NULL, NULL);  
    proc\_create(kernel3\_proc, NULL, NULL);  
    proc\_create(kernel4\_proc, NULL, NULL);  
    proc\_create(kernel5\_proc, NULL, NULL);  
    proc\_create(kernel6\_proc, NULL, NULL);  
    proc\_create(kernel7\_proc, NULL, NULL);  
    proc\_create(kernel8\_proc, NULL, NULL);  
    proc\_create(kernel9\_proc, NULL, NULL);  
    proc\_create(kernel10\_proc, NULL, NULL);  
    printk("create proc done\n");  
    struct list\_elem \*ele;  
    while(1) {    
        schedule();      
    }  
}  
  
  
  
void proc\_print\_data()  
{  
    int a, b, c, d, bp, si, di, sp;  
  
    //eax ebx ecx edx  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(a));  
  
    \_\_asm\_\_ \_\_volatile("mov %ebx ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(b));  
      
    \_\_asm\_\_ \_\_volatile("mov %ecx ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(c));  
      
    \_\_asm\_\_ \_\_volatile("mov %edx ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(d));  
      
    //ebp esi edi esp  
    \_\_asm\_\_ \_\_volatile("mov %ebp ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(bp));  
  
    \_\_asm\_\_ \_\_volatile("mov %esi ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(si));  
  
    \_\_asm\_\_ \_\_volatile("mov %edi ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(di));  
  
    \_\_asm\_\_ \_\_volatile("mov %esp ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(sp));  
  
    printk(    "\neax %o ebx %o ecx %o edx %o"\  
            "\nebp %o esi %o edi %o esp %o\n"\  
            , a, b, c, d, bp, si, di, sp);  
}  
  
void hexDump (void \*addr, int len) {  
    int i;  
    unsigned char buff[17];  
    unsigned char \*pc = (unsigned char\*)addr;  
  
    if (len == 0) {  
        printk("  ZERO LENGTH\n");  
        return;  
    }  
    if (len < 0) {  
        printk("  NEGATIVE LENGTH: %i\n",len);  
        return;  
    }  
  
    for (i = 0; i < len; i++) {  
        if ((i % 16) == 0) {  
            if (i != 0)  
                printk ("  %s\n", buff);  
  
            printk ("  %04x ", i);  
        }  
  
        printk (" %02x", pc[i]);  
  
        if ((pc[i] < 0x20) || (pc[i] > 0x7e))  
            buff[i % 16] = '.';  
        else  
            buff[i % 16] = pc[i];  
        buff[(i % 16) + 1] = '\0';  
    }  
  
    while ((i % 16) != 0) {  
        printk ("   ");  
        i++;  
    }  
  
    printk ("  %s\n", buff);  
}

interrupt.c

#include <interrupt.h>  
#include <device/console.h>  
#include <type.h>  
#include <device/pit.h>  
#include <proc/sched.h>  
#include <device/io.h>  
#include <device/kbd.h>  
#include <proc/proc.h>  
#include <ssulib.h>  
#include <proc/switch.h>  
#include <syscall.h>  
  
#pragma pack(push, 1)  
  
typedef struct \_ID  
{  
    unsigned short LowOffset;  
    unsigned short CodeSelector;  
    unsigned short Type;  
    unsigned short HighOffset;  
} \_ID;  
  
typedef struct \_IDTR  
{  
    unsigned short Size;  
    unsigned long Address;  
} \_IDTR;  
#pragma pack(pop)  
  
#ifndef ASSERT  
#define ASSERT(CONDITION) ( (void\*) 0)  
#endif  
  
static unsigned long ticks;  
intr\_handler\_func \*handlers[INTR\_MAX];  
bool sched\_on\_return;  
static bool in\_external\_intr;    
\_ID     \*idt;  
extern int scheduling;  
unsigned long proc\_ticks;  
extern struct list level\_que[QUE\_LV\_MAX];  
  
typedef void intr\_stub\_func (void);  
void default\_handler(struct intr\_frame \*iframe);  
void timer\_handler(struct intr\_frame \*iframe);  
void syscall\_handler(struct intr\_frame \*iframe);  
void intr\_exit (void);  
extern intr\_stub\_func \*intr\_stubs[INTR\_MAX];  
  
int init\_intr(void)  
{  
    int i;  
    \_IDTR   IDTR;  
    ticks = 0;  
    in\_external\_intr = false;  
    sched\_on\_return = false;  
    enum intr\_level old\_level = intr\_disable();  
  
    \_\_asm\_\_ \_\_volatile("sidt %0"::"m" (IDTR));  
    printk("idtr size : %d address : 0x%x\n", IDTR.Size, IDTR.Address);  
    idt = (\_ID \*)IDTR.Address;  
  
    for(i = 0; i < INTR\_MAX; i++)  
    {  
        idt[i].LowOffset = (unsigned short)((unsigned long)intr\_stubs[i] & 0xffff);  
        idt[i].CodeSelector = (unsigned short)CODE\_SEGMENT;  
        idt[i].Type = (unsigned short)0x8e00;       
        idt[i].HighOffset = (unsigned short)(((unsigned long)intr\_stubs[i] >> 16) & 0xffff);  
  
        handlers[i] = default\_handler;  
    }  
  
    reg\_handler(32, timer\_handler);  
    printk("%s", "Timer Handler Registration\n");  
    printk("%s", "Keyboard Handler Registration\n");  
  
    reg\_handler(0x30, syscall\_handler);  
    printk("%s", "System Call Handler Registration\n");  
  
    intr\_set\_level (old\_level);  
    return 1;  
}  
  
void intr\_common(struct intr\_frame \*iframe)  
{  
    in\_external\_intr = iframe->vec\_no >= 0x20 && iframe->vec\_no < 0x30;  
  
    if(in\_external\_intr)  
    {  
        ASSERT (intr\_get\_level () == INTR\_OFF);  
        sched\_on\_return = false;  
    }  
  
    if(handlers[iframe->vec\_no] != NULL)  
        handlers[iframe->vec\_no](iframe);  
    else  
        default\_handler(iframe);  
  
    if(in\_external\_intr)  
    {  
        ASSERT (iframe->vec\_no >= 0x20 && iframe->vec\_no < 0x30);  
        in\_external\_intr = false;  
        outb(0x20, 0x20);  
  
        if(iframe->vec\_no >= 0x28)  
            outb(0xa0, 0x20);  
  
        if(sched\_on\_return)  
            schedule();  
    }  
}  
  
void do\_sched\_on\_return(void)  
{  
    ASSERT (intr\_context ());  
    sched\_on\_return = true;  
}  
  
void reg\_handler(unsigned short idx, intr\_handler\_func \*handler)   
{  
    enum intr\_level old\_level = intr\_disable();  
    handlers[idx] = handler;  
  
    intr\_set\_level (old\_level);  
    return;  
}  
  
#define FLAG\_MBS  0x00000002  
#define FLAG\_IF   0x00000200  
  
enum intr\_level intr\_get\_level (void)   
{  
    uint32\_t flags;  
    asm volatile ("pushfl; popl %0" : "=g" (flags));  
    return flags & FLAG\_IF ? INTR\_ON : INTR\_OFF;  
}  
  
enum intr\_level intr\_set\_level (enum intr\_level level)   
{  
    return level == INTR\_ON ? intr\_enable () : intr\_disable ();  
}  
  
enum intr\_level intr\_enable (void)   
{  
    enum intr\_level old\_level = intr\_get\_level ();  
    ASSERT (!intr\_context ());  
  
    asm volatile ("sti");  
  
    return old\_level;  
}  
  
enum intr\_level intr\_disable (void)   
{  
    enum intr\_level old\_level = intr\_get\_level ();  
  
    asm volatile ("cli" : : : "memory");  
  
    return old\_level;  
}  
  
bool intr\_context (void)  
{  
    return in\_external\_intr;  
}  
  
unsigned long get\_ticks(void)  
{  
    return ticks;  
}  
  
void default\_handler(struct intr\_frame \*iframe)  
{  
    printk("\n\nint intr %d\n\n", iframe->vec\_no);  
}  
  
void timer\_handler(struct intr\_frame \*iframe)  
{  
    ticks++;  
    if ((cur\_process -> pid != 0) && (!scheduling) && (cur\_process->state == PROC\_RUN)) {          
        //i/o시작 시간과 스케줄 시간이 겹치는 것을 고려해 구조 바꿈      
        if(cur\_process->time\_slice >= TIMER\_MAX\*cur\_process->que\_level)  
            do\_sched\_on\_return();  
        else  
        {  
            cur\_process->time\_used++;      
            cur\_process->time\_slice++;  
        }  
    }  
  
#ifdef SCREEN\_SCROLL  
    static unsigned long refresh\_ticks = 0;  
    if(++refresh\_ticks >= (PIT\_FRQ\_HZ/REFRESH\_FPS)) {  
        refresh\_ticks = 0;  
        refreshScreen();  
    }  
#endif  
}  
  
void syscall\_handler(struct intr\_frame \*iframe)  
{  
    long sys\_num, arg\_num, ret;  
    int \*pt;  
    int i;  
  
#define SYSCALL\_ARG\_MAX 3  
    long arg[SYSCALL\_ARG\_MAX];  
  
    pt = (int\*) (((int)&iframe->ss) - 4);  
    sys\_num = \*pt++;  
    arg\_num = syscall\_tbl[sys\_num][1];  
  
    for(i = 0; i<arg\_num; i++)  
        arg[i] = \*pt++;  
  
    switch(arg\_num)  
    {  
        case 0:  
            ret = ((int(\*)(void))syscall\_tbl[sys\_num][0])();  
            break;  
        case 1:  
            ret = ((int(\*)(int))syscall\_tbl[sys\_num][0])(arg[0]);  
            break;  
        case 2:  
            ret = ((int(\*)(int, int))syscall\_tbl[sys\_num][0])(arg[0], arg[1]);  
            break;  
        case 3:  
            ret = ((int(\*)(int, int, int))syscall\_tbl[sys\_num][0]) (arg[0], arg[1], arg[2]);  
            break;  
    }  
      
    \*(pt + 3) = ret;          
}