컴퓨터학부 20132397 윤건택

1. 과제 개요

이번 과제는 프로세스를 여러개 생성했을 때 시스템의 성능을 개선하기 위해 CPU의 자원을 적절히 배정하는 스케줄링을 익히는 것이 목표이다. 이번 과제에 쓰이는 스케줄링 방식은 Multi-level FeedBack Queue로, 여러개의 준비큐를 두고, 준비 큐마다 다른 CPU 할당 시간을 부여한다. 이 스케줄러는 상위 단계 큐에 우선순위와 적은 tick을 부여하고, 하위 단계 큐에 상위 단계보다 비교적 많은 tick을 부여하여 작업시간이 짧은 프로세스가 우선적으로 처리하게 된다. 그러므로 최하위 단계 큐를 제외한 모든 큐에서는 먼저 들어온 업무를 먼저 처리하는 FCFS 방식을 적용하고, 최하위 단계 큐에서는 프로세스들 사이에 우선순위를 두지않고 순서대로 시간단위로 CPU를 할당하는 Round Robin 방식을 적용한다.

1. 설계

다른 코드들을 변경하기 전,proc\_create() 함수에 PID 0 이후 생성되는 프로세스를 큐레벨 1에 넣어 주는 코드가 구현되어있지 않기 때문에, list.c에 있는 함수 list\_push\_back을 이용하여 level\_que[1]에 생성된 프로세스들을 넣어준다.

그리고 sched.c로 넘어가 schedule()함수 내부에서 현재 PID가 0이 아닐 경우 latest를 cur로 변경하여주고, 그 외의 PID일 경우에는 프로세스의 큐레벨에 따라 현재 프로세스를 알맞는 큐레벨에 삽입한다. 큐레벨 리스트에 있는 것을 사용해야 하기에 이또한 list.c에 있는 list\_push\_back과 list\_pop\_front함수를 이용한다.

sched\_find\_que 함수에서는 큐 리스트가 비어있는지 아닌지에 대한 판단과 비어있지않다면 리스트에 있는 맨 첫번째 프로세스를 반환하는 기능을 구현해야한다. 우선 list\_empty를 통해 큐 리스트의 비어있는 여부를 확인하고, get\_next\_proc함수를 통해 프로세스를 result 값에 넣어 반환한다.

현재 프로세스 상황 출력 구현은 printk로 구현한다. proc\_sleep()하는 시점이 I/O가 출력되는 시점이므로 함수 내부에 printk로 PID, time\_used, 큐 레벨을 출력하고, proc\_wake() 함수가 사용되는 시점이 큐레벨이 2에서 1로 올라가는 경우이므로 이 함수 내부에 큐 레벨이 변경되었다는 내용을 출력하도록 한다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PID  (순서) | 시작시간  (tick) | 종료시각  (tick) | I/O 제외  총 수행시간  (tick) | I/O 시작 시각  (tick) |
| 1 | 0 | 1250 | 200 | 640 |
| 2 | 40 | 870 | 120 | 70 |
| 3 | 70 | 2220 | 300 | 700 |
| 4 | 110 | 3050 | 300 | 140,1030 |
| 5 | 140 | 3390 | 400 | 160,1150 |
| 6 | 160 | 2630 | 200 | 170,340,1020,1940 |
| 7 | 170 | 3670 | 500 | 190,360,1170 |
| 8 | 190 | 3890 | 600 | 210 |
| 9 | 210 | 4090 | 750 | 240 |
| 10 | 240 | 4170 | 800 | 1490 |

1. 실행 결과







1. 소스 코드

**interrupt.c**

…

void timer\_handler(struct intr\_frame \*iframe)

{

ticks++;

//sleep 상태 프로세스 1틱 증가시키는 것 방지

if ((cur\_process -> pid != 0) && (!scheduling) && cur\_process->state == PROC\_RUN) { cur\_process->time\_used++;

cur\_process->time\_slice++;

if(cur\_process->que\_level == 1){//큐 레벨이 1일 경우 틱 40 부여

if(cur\_process->time\_slice >= TIMER\_MAX)

do\_sched\_on\_return();

}

else if(cur\_process->que\_level == 2){//큐 레벨이 2일 경우 틱 80 부여

if(cur\_process->time\_slice >= 80)

do\_sched\_on\_return();

}

}

…

**proc.c**

…

pid\_t proc\_create(proc\_func func, struct proc\_option \*opt, void\* aux)

{

struct process \*p;

int idx;

int i,j;

…

//생성된 프로세스 큐레벨1 리스트에 삽입

list\_push\_back(&level\_que[1], &p->elem\_stat);

intr\_set\_level (old\_level);

return p->pid;

}

…

void proc\_wake(void)

{

struct process\* p;

int que\_level;

int old\_level;

unsigned long long t = get\_ticks();

struct list\_elem \*e;

intr\_disable();

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);

que\_level = 1;

p->que\_level = 1;

list\_push\_back(&level\_que[que\_level], &p->elem\_stat);

p->state = PROC\_RUN;

//큐레벨 변경 메세지 출력

printk("proc%d change the queue(2->1)\n", p->pid);

}

intr\_enable();

}

void proc\_sleep(unsigned ticks)

{

unsigned long cur\_ticks = get\_ticks();

intr\_disable();

//I/O중인 프로세스의 ID와 큐레벨 출력

printk("Proc %d I/O at %d\n", cur\_process->pid, cur\_process->time\_used);

printk("Proc %d I/O Que is %d\n", cur\_process->pid, cur\_process->que\_level);

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();

}

**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.

\*/

//큐레벨1의 리스트가 비지 않았을 경우

if(!list\_empty(&level\_que[1]))

//큐레벨1 리스트에서 첫번째 프로세스 가져옴

result = get\_next\_proc(&level\_que[1]);

//큐레벨2의 리스트가 비지 않았을 경우

else if(!list\_empty(&level\_que[2]))

//큐레벨2 리스트에서 첫번째 프로세스 가져옴

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)

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) {//현재 PID가 0이 아닐 경우

latest = cur;//latest를 cur로 변경

latest->time\_slice = 0;//time\_slice 값 초기화

scheduling = 0;//tick 증가로 명시

cur\_process = idle\_process;//현재 프로세스를 idle로 지졍

switch\_process(cur, idle\_process);//현재 프로세스를 idle로 변경

return;

}

switch (latest -> que\_level){

//현재 큐레벨이 1일 경우

case 1:

//latest의 상태가 PROC\_RUN일 때

if(latest->state == PROC\_RUN){

//큐레벨1 리스트 첫번째 프로세스를 큐레벨2 리스트 마지막으로 보냄

list\_push\_back(&level\_que[2], list\_pop\_front(&level\_que[1]));

//보낸 프로세스의 que\_level값 2로 변경

latest->que\_level = 2;

intr\_disable();

//큐레벨 변경 메세지 출력

printk("proc%d change the queue(1->2)\n", latest->pid);

intr\_enable();

}

break;

//현재 큐레벨이 2일 경우

case 2:

//latest의 상태가 PROC\_RUN일 때

if(latest->state == PROC\_RUN)

//큐레벨2 리스트 첫번째 프로세스를 큐레벨2 리스트 마지막으로 보냄

list\_push\_back(&level\_que[2], list\_pop\_front(&level\_que[2]));

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);

}

}

if (printed)

printk("\n");

//큐에 프로세스가 존재 할 경우 next에 프로세스 지정

if ((next = sched\_find\_que()) != NULL) {

intr\_disable();

//현재 선택된 프로세스 아이디 출력

printk("Selected process : %d\n", next -> pid);

intr\_enable();

cur\_process = next;//현재 프로세스를 next로 변경

scheduling = 0;//tick 증가 가능 명시

switch\_process(cur, next);//현재 프로세스를 next로 변경

return;

}

return;

}

void proc\_que\_levelup(struct process \*cur)

{

/\*TODO : change the queue lv2 to queue lv1.\*/

cur->que\_level = 1;

}

void proc\_que\_leveldown(struct process \*cur)

{

/\*TODO : change the queue lv1 to queue lv2.\*/

cur->que\_level = 2;

}