OS assignment1

Design

MLFQ scheduler 구현

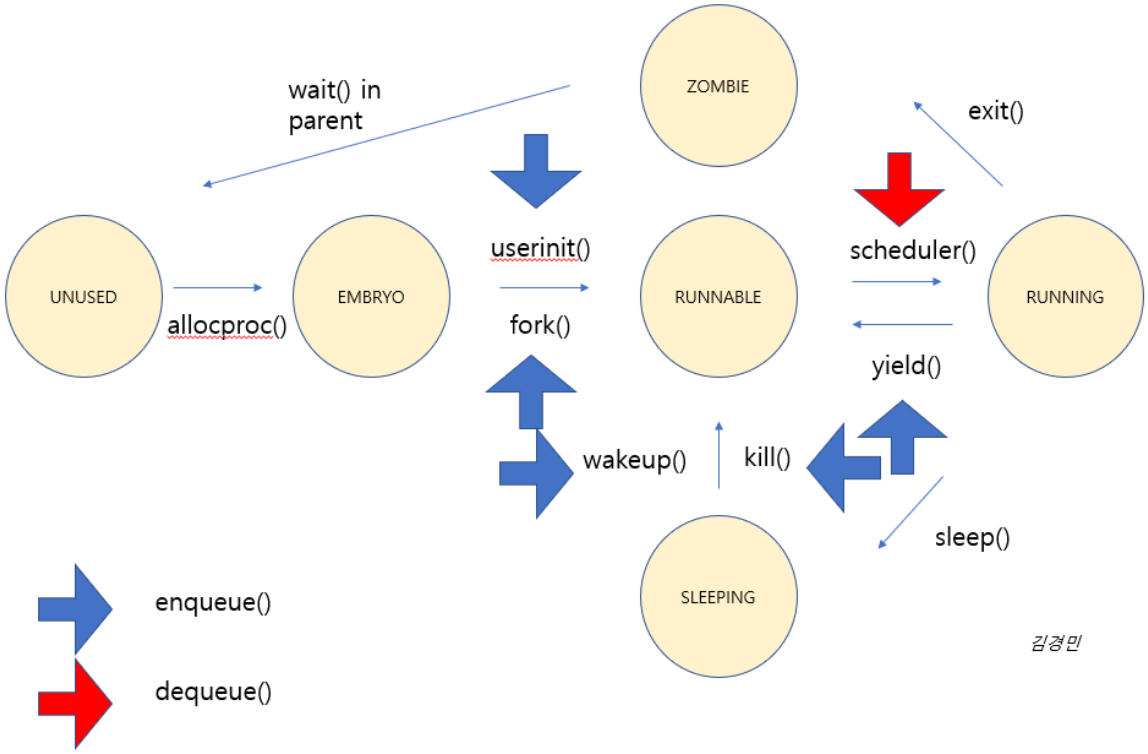
1. process 탐색 방식

- 우선 스케줄링 될 process를 탐색하는 방식을 생각해보았다. 기본적으로 Queue를 이용해야겠다고 생각했으나 구지 Queue를 다 구현해야 할 필요성까진 느끼지 못하였다.
- 그래서 proc포인트의 배열을 선언해 주었고, 해당 Level에 Head만 넣어주었다.
- 그 뒤에 들어오는 process는 process 구조체 안에 proc포인트 next를 이용하여 linkedlist 느낌으로 주렁주렁 매달아 주었다. 이렇게 하면 process의 순서를 통제하기 편하였다.

2. process 선택

- Queue에 어떠한 process를 넣을것인가.

runnable한 프로세스만 넣어준다면 process를 선택하는 과정에서 Level0부터 Level2까지 priority를 고려하지 않는다면 Head에 있는 process를 dequeue해주면 선택이 가능하기 때문에 이러한 방식을 택하였다.



따라서 다음과 같은 상황에서 프로세스를 대기열에 넣는다.

- switch 전 대기열에서 빼기

다음 process를 선택할때 for문 을 통해 Level 0부터 2까지 순회하였다.

Level 0과 1은 priority를 고려하지 않아도 되기때문에 해당배열에 값이 있다면 바로 빼주면 되었다.

Level 2에서는 priority를 고려하여 가장 높은 우선순위를 뺐다.

3. Level변경 및 priority 설정

- trap.c에 ticks가 1 올라갈때마다 proc.h에 선언된 global_tick을 1씩 올려주었다.
- 그리고 RUNNING 상태에서 Time interrupt가 발생되면 process 구조체에 time_allotment 변수의 값을 1증가시킨다.
- process가 1틱을 사용하고 scheduler로 돌아와 LEVEL변경이나 priority를 변경해야할 조건을 만족하면 변경해준다.

4. priority boosting

- global tick이 100에 도달하면 그냥 L0에 다 주렁주렁 매달아주면된다. 이게 바로 Queue로 구현을 했을때의 강점이라고 생각하였다. process 구조체의 값을 초기화 해주면서 그냥 L0 맨 끝 process에 L1을 L1의 맨 끝 process에 L2를 연결 해주면 끝이다.

5. schedulerLock

- schedulerLock이 호출되면 proc.c의 sheduler_locked 전역변수가 1이된다.
- 그리고 Lock을 건 process를 정의하기 위해 호출한 process 구조체의 lock_scheduler 변수 값을 1로 바꿔준다.
- 그 후 scheduler 내에서 Lock을 건 process가 실행을 마치고 RUNNABLE일때 p의 값을 그대로 유지하고 switch 전에 process를 선택하는 과정을 생략한다.
- 만약 Level변경이 필요해서 큐의 이동이 필요하다면 그때만 Queue에서 뺴주고 변경된 Level의 Queue에 넣어준다.

6. schedulerUnlock

- schedulerUnlock은 다음과 같은 상황에서 일어난다.

1) priority boosting이 일어난경우

2) Lock을 건 process가 1틱을 사용한 후 RUNNABLE이 아닌경우 (Sleeping 상태에서 cpu를 잡고 있는 것은 상당히 비효율적이라고 생각하였다)

3) Lock을 건 process가 schedulerUnlock을 호출한경우

- schedulerUnlock이 일어나면 scheduler_locked 전역변수의 값을 0으로 바꿔준다.
- 그리고 Lock을 건 process를 찾아서 L0 Queue의 head로 넣어준다.

Implement

1. 기본 선언

```
proc.h

struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  int level;                   // 프로세스의 레벨
  int priority;                //level 2에서 사용되는 프로세스의 우선순위
  int time_quantum;            //프로세스의 time quantum
  int time_allotment;          //프로세스가 해당 Level에서 얼마나 실행되었는지
  int lock_scheduler;          //값이 1이면 Lock을 건 process
  int already_enqueued;        //값이 1이면 이미 Queue에 들어가 있다
  struct proc *next;            //다음 프로세스를 가리키는 포인터
};

struct proc *level_queue[3]; // 각 큐의 헤드
extern int global_tick;     //priority boosting을 위한 tick
```

prco의 구조는 다음과같다

`time_quantum` : 속해 있는 level에서 쓸 수 있는 time_quantum을 나타낸다. level이 변경되면 해당 level의 time_quantum 으로 변경된다.

`already_enqueued` : `enqueue()`가 실행되면 해당 1로 바꿔준다. `dequeue()` 나 `dequeue_specific()` 이 실행되면 0으로 바꿔준다. 혹시라도 queue에 같은 process가 여러개 들어갈 수 있는 경우를 방지해준다.

lock_scheduler : `schedulerLock()` 이 호출되면 1로 `schedulerUnlock()` 이 호출되면 0으로 변경한다. `schedulerUnlock()` 에서 Lock을 건 process를 찾는데 사용된다.

```
proc.c
```

```
static struct proc*
allocproc(void)
{
  ...
  //mlfq 관련 초기화
  p->level = 0;
  p->priority = 3;
  p->time_quantum = 2 * p->level +4;
  p->time_allotment = 0;
  p->next = 0;
  ...
}
```

allocproc()에서 mlfq관련 초기화를 해주었다.

```
proc.

...
int global_tick = 0;
int scheduler_locked=0;
...
```

`scheduler()`에서 사용하는 전역변수를 초기화 해주었다.

---

2. System call

proc.c

- getLevel()

```
proc.

...
int
getLevel(void)
{
  return myproc()->level;
}
...
```

- setPriority

```
proc.c

...
void setPriority(int pid, int priority) {
  if (priority < 0 || priority > 3) {
    cprintf("Invalid priority value. Priority must be between 0 and 3.\n");
    return;
  }// priority 범위 밖의 값이 인자로 들어왔을 경우

  struct proc *p;
  int pid_found = 0;

  acquire(&ptable.lock);
  for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
    if (p->pid == pid) {
      p->priority = priority;
      pid_found = 1;
      break;
    } // 해당 pid값의 process를 찾고 priority 값 변경 후 pid_fount flag 값 변경
  }
  release(&ptable.lock);

  if (!pid_found) {
    cprintf("PID not found.\n");
  }
  return;
}
...
```

입력으로 들어온 pid에 해당하는 process를 ptable을 순회하여 찾고, 찾으면 해당 process의 priority값을 설정해준다. priority값이 0~3범위에 해당하지 않거나 해당 pid의 process를 찾지 못하면 오류 메세지를 띄우고 return해준다.

- schedulerLock()

```
proc.c

...
void
schedulerLock(int password)
{
  if(scheduler_locked == 1){
    cprintf("already locked\n");
    return;
  } //이미 Lock이 걸려있는 경우
```

```
  if(password == 2019041703){
    global_tick = 0;
    scheduler_locked = 1;
    myproc()->lock_scheduler = 1;
  }
  else{
    cprintf("Incorrect password for schedulerLock\n");
    cprintf("Pid : %d , time quantum : %d, Queue_Level : %d\n",
myproc()->pid,myproc()->time_allotment, myproc()->level);
    exit();
  }
}
...
```

`scheduler_locked`가 1이면 이미 Lock이 걸려있는 상태 이므로 오류메세지를 띄워준다.

password가 일치한다면 `scheduler_locked`를 1로 바꿔주고 `global_tick`을 0으로 해주고 현재 실행중인 process의 `lock_scheduler` 값을 1로 해준다.

- schedulerUnlock()

```
proc.c

...
void
schedulerUnlock(int password){
  if(scheduler_locked!=1){
    cprintf("Scheduler is not locked.");
    return;
  } //Lock이 걸려있지 않은 경우

  if(password == 2019041703){
    struct proc *p = 0;
    // Lock이 걸려있는 process를 찾는다
    for (int level = 0; level < 3 && p == 0; level++) {
      for (struct proc *current = level_queue[level]; current != 0; current = current->next) {
        if (current->lock_scheduler == 1) {
          p = current;
          break;
        }
      }
    }

    // process가 있는경우
    if (p != 0) {
      scheduler_locked = 0;
      p->lock_scheduler = 0;
```

```
      if(p->state !=RUNNABLE){
         dequeue_specific(&level_queue[p->level],p);
      } //Runnable이 아니라면 Queue에서 제거
      if(p->state == RUNNABLE|| p->state==RUNNING)){
        p->next = level_queue[0];
        level_queue[0] = p;
        p->level = 0;
        p->time_allotment = 0;
        p->time_quantum = 2 * p->level + 4;
        p->already_enqueued = 1;
      } //RUNNABLE 또는 Running이면 LO의 Head에 넣어주기
    }
  }
    else {
    cprintf("Incorrect password for schedulerUnLock\n");
    cprintf("Pid : %d , time quantum : %d, Queue_Level : %d\n",
myproc()->pid, myproc()->time_allotment, myproc()->level);
    exit();
  }
}
...
```

```
```

lock이 걸려있지 않다면 return

lock이 걸려있다면 Queue를 순회하며 `lock_scheduler`가 1인 Lock을 건 process를 찾는다

그 후 `scheduler_locked`와 process의 `lock_scheduler`를 0으로 해주고

RUNNABLE이 아니라면 queue에서 뺴주고 아니라면 L0의 Head에 넣어준다.

sysproc.c

```
sysproc.c

...
int
sys_yield(void)
{
  yield();
  return 0;
}


int
sys_getLevel(void)
{
```

```
  return getLevel();
}

int
sys_setPriority(void)
{
  int pid, priority;
  if(argint(0, &pid)<0){
    return -1;
  }
  if(argint(1, &priority)<0){
    return -1;
  }
  setPriority(pid, priority);
  return 0;
}

int
sys_schedulerLock(void)
{
  int password;
  if(argint(0,&password)<0) return -1;
  schedulerLock(password);
  return 0;
}

int
sys_schedulerUnlock(void)
{
  int password;
  if(argint(0,&password)<0) return -1;
  schedulerUnlock(password);
  return 0;
}
...
```

user.h

```
user.h

...
int getLevel(void);
void yield(void);
void setPriority(int, int);
void schedulerLock(int);
void schedulerUnlock(int);
...
```

traps.h

```
traps.h

...
#define T_SCHEDLOCK     129
#define T_SCHEDUNLOCK   130
...
```

`schedulerLock()`과 `schedulerUnlock()`은 interrupt로도 처리해야하므로 설정하였다.

defs.h

```
defs.h

...
void            yield(void);
int             getLevel(void);
void            setPriority(int, int);
void            schedulerLock(int);
void            schedulerUnlock(int);
...
```

syscall.h

```
syscall.h

...
#define SYS_yield  23
#define SYS_getLevel  24
#define SYS_setPriority 25
#define SYS_schedulerLock 26
#define SYS_schedulerUnlock 27
```

syscall.c

```
syscall.c

...
extern int sys_yield(void);
extern int sys_getLevel(void);
extern int sys_setPriority(void);
extern int sys_schedulerLock(void);
```

```
extern int sys_schedulerUnlock(void);

...

[SYS_yield]   sys_yield,
[SYS_getLevel]   sys_getLevel,
[SYS_setPriority]   sys_setPriority,
[SYS_schedulerLock] sys_schedulerLock,
[SYS_schedulerUnlock] sys_schedulerUnlock,

...
```

usys.S

```
usys.s

...
SYSCALL(yield)
SYSCALL(getLevel)
SYSCALL(setPriority)
SYSCALL(schedulerLock)
SYSCALL(schedulerUnlock)
...
```

3.enqueue/dequeue

enqueue

```
proc.c

...
void enqueue(struct proc *p, struct proc **queue) {
  if (p->already_enqueued) {
    return; // queue에 이미 process가 있는경우
  }
  if (*queue == 0) {
    // 큐가 비어있으면 프로세스를 큐의 첫 번째 요소로 추가
    *queue = p;
    p->next = 0;
    p->already_enqueued = 1;
  } else {
    struct proc *tmp = *queue;
    // 큐의 마지막 요소를 찾음
```

```
    while (tmp->next) {
      tmp = tmp->next;
    }

    // 마지막 요소 다음에 프로세스를 추가
    tmp->next = p;
    p->next = 0;
    p->already_enqueued = 1;

  }
}
...
```

queue가 비어있으면 process의 level에 해당하는 배열 index에 head로 아니면 마지막 process 뒤에 달아준다.

dequeue

```
proc.c

...

struct proc *dequeue(struct proc **queue) {
  if (*queue == 0) {
    // 큐가 비어있으면 0 반환
    return 0;
  } else {
    struct proc *dequeued_proc = *queue;
    // 큐의 첫 번째 요소를 제거하고 두 번째 요소를 첫 번째 요소로 설정
    *queue = (*queue)->next;
    dequeued_proc->next = 0;

    dequeued_proc->already_enqueued = 0;
    return dequeued_proc;
  }
}
...
```

해당 level의 queue가 비어있으면 0을 반환하고 아니라면 Head에 있는 값을 return 해주고 그 다음값을 head로 지정해준다.

dequeue_specific

```
proc.c

...
void
dequeue_specific(struct proc **queue, struct proc *p_specific)
{
  struct proc *p = 0;
  struct proc *prev = 0;

  if(p_specific->already_enqueued == 0){
    return;
  }//p가 큐에 들어가 있지 않은 상태

  for (p = *queue; p != 0; prev = p, p = p->next) {
    if (p == p_specific) {
      if (prev) {
        prev->next = p->next;
      } else { //찾고자 하는 값이 첫번째 인경우
        *queue = p->next;
      }
      p->next = 0;
      p->already_enqueued = 0;
      break;
    }
  }
}
...
```

특정 process를 해당 Level에서 추출할때 쓰는 함수이다. level2에서 priority가 가장 높은 process를 찾을때 와 scheduler lock이 걸렸을때 lock을 건 process의 `level` 변경이 필요하여 queue에서 빼줄때 와 `schedulerUnlock()` 에서쓴다.

---

4. scheduler

process 선택

```
proc.c

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

    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    if(p&&scheduler_locked==1){
```

```
      //락이 걸려있으면 dequeue를 해주지 않고 기존의 p값을 그대로 가져감
    }
    else{
      for(int level = 0; level < 3; level++) {
        struct proc *p_high_priority = 0;
        //level2에서 우선적으로 실행되야 할 process 선택
        if (level == 2) {
          for (p = level_queue[level]; p != 0; p = p->next) {
            if (p_high_priority == 0 || p->priority < p_high_priority->priority) {
              p_high_priority = p;
            }
          }
          if (p_high_priority != 0) {
            dequeue_specific(&level_queue[level], p_high_priority);
            p = p_high_priority;
            break;
          }
        } else {
          p = dequeue(&level_queue[level]);
          if (p != 0) {
            break;
          }
        }
      }
    }

    if(p == 0) {
      release(&ptable.lock);
      continue;
    }
...
```

- 만약 schedulerlock이 걸려있으면 기존의 p값을 그대로 사용한다.
- 아니라면 level 0부터 순회한다. level0나 1에 값이 있으면 그대로 head에 있는 값을 dequeue해서 p로 정한다
- level 2까지 왔으면 priority가 높은 것을 p로 정해준다
- 만약에 p가 선택되지 않는다면 다시 for문 첫부분으로 이동한다.

context switching

```
    proc.c

...
    c->proc = p;
```

```
    switchuvm(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;

    if (scheduler_locked == 1 && p->state != RUNNABLE) {
      //scheduler_lockde이 되어있는데 p가 RUNNABLE이 아니라면 lock을 해제함
      schedulerUnlock(2019041703);
    }
...
```

선택된 p를 context switching 해준다.

context switching 후 lock이 걸려있는 상태인데 p가 Runnable이 아니라면 lock을 해제한다

Level change

```
proc.

...
    if (p->level < 2 && p->time_allotment >= p->time_quantum)
    {//Level 0과 1에서 timequantum 을 다 사용한 경우
      if(scheduler_locked==1)
      {//lock이 걸려있으면 우선 queue에서 빼줌
        dequeue_specific(&level_queue[p->level],p);
      }
      p->level++;
      p->time_quantum = 2 * p->level + 4;
      p->time_allotment = 0;
    }
    else if (p->level == 2 && p->time_allotment >= p->time_quantum)
    {//leve2에서 timequantum을 다 사용한 경우 priority 변경
      p->priority = (p->priority - 1 >= 0) ? p->priority - 1 : 0;
      p->time_allotment = 0;
    }
    if(p->state == RUNNABLE) {
      enqueue(p, &level_queue[p->level]);
    }
    if(scheduler_locked!=1)
    {//schedulerlock이 걸려있지 않다면 p초기화
      p = 0;
    }
...
```

level 변경이 필요한 경우 level 변경을 진행해준다

만약 lock이 걸려있는 상태에서 lock을 건 process의 level변경이 필요하다면 해당 process의 기존 level의 queue에서 빼준 후 변경도니 level의 queue에서 넣어준다.

```
if(p->state == RUNNABLE) {
      enqueue(p, &level_queue[p->level]);
    }
```

이 코드에서 lock이 걸려있든 걸려있지 않든 동일하게 들어간다.

그 후 lock이 걸려있지 않다면 `p`를 0으로 초기화 해준다. 이 부분은 만약 `runnable`한걸 못찾는다면 p가 null을 가지고 있을 것 이기 때문에 deadlock을 막기 위함이다. lock이 걸려있는 상태에서 p를 초기화 해주지 않는 이유는 lock이 걸려있는 상태에서 process 선택과정에서 dequeue를 해주지 않고 기존의 p값을 그대로 실행시키기 때문이다.

priority boosting

```
proc.c

...
//priority boosting이 필요한경우
    if (global_tick >= 100)
    {
      struct proc *current = level_queue[0];

      // level_queue[0]를 모두 초기화하고 끝으로 이동
      while (current != 0 && current->next != 0) {
        current->priority = 3;
        current->time_allotment = 0;
        current = current->next;
      }
      if (current != 0) {
        //level_queue[0]의 맨 마지막 값 초기화
        current->priority = 3;

        current->time_allotment = 0;
      }

      // level_queue[1] 및 level_queue[2]의 프로세스를 level_queue[0]로 이동
      for (int level = 1; level <= 2; level++) {
        struct proc *next_level_process = level_queue[level];
```

```
        while (next_level_process != 0) {
          next_level_process->level = 0;
          next_level_process->priority = 3;
          next_level_process->time_allotment = 0;
          next_level_process->time_quantum = 2 * next_level_process->level + 4;

          if (current == 0) {
            //leve_queue[0]이 비어있는 경우
            level_queue[0] = next_level_process;
            current = next_level_process;
          }
          else {
            current->next = next_level_process;
            current = next_level_process;
          }
          next_level_process = next_level_process->next;
        }
        level_queue[level] = 0; // 현재 레벨 큐를 초기화
      }
      if(scheduler_locked==1){
        //schedulerlock이 걸려있으면 해제
        schedulerUnlock(2019041703);
      }
      global_tick = 0; // global_tick초기화
    }
    release(&ptable.lock);
  }
}
```

- global tick이 100이 되면 level0 에 있는 process의 `priority`와 `time_allotment`를 초기화해준 후 level 1과 level2에 process들을 level 0끝에 쭉 잇는다.
- lock이 걸려있는 경우 lock을 해제해준다
- `global_tick` 값을 0으로 초기화해준다.

---

5. trap

```
trap.c

...
if(tf->trapno == T_SCHEDLOCK){
    schedulerLock(2019041703);
    exit();
  }

  if(tf->trapno == T_SCHEDUNLOCK){
    schedulerUnlock(2019041703);
    exit();
```

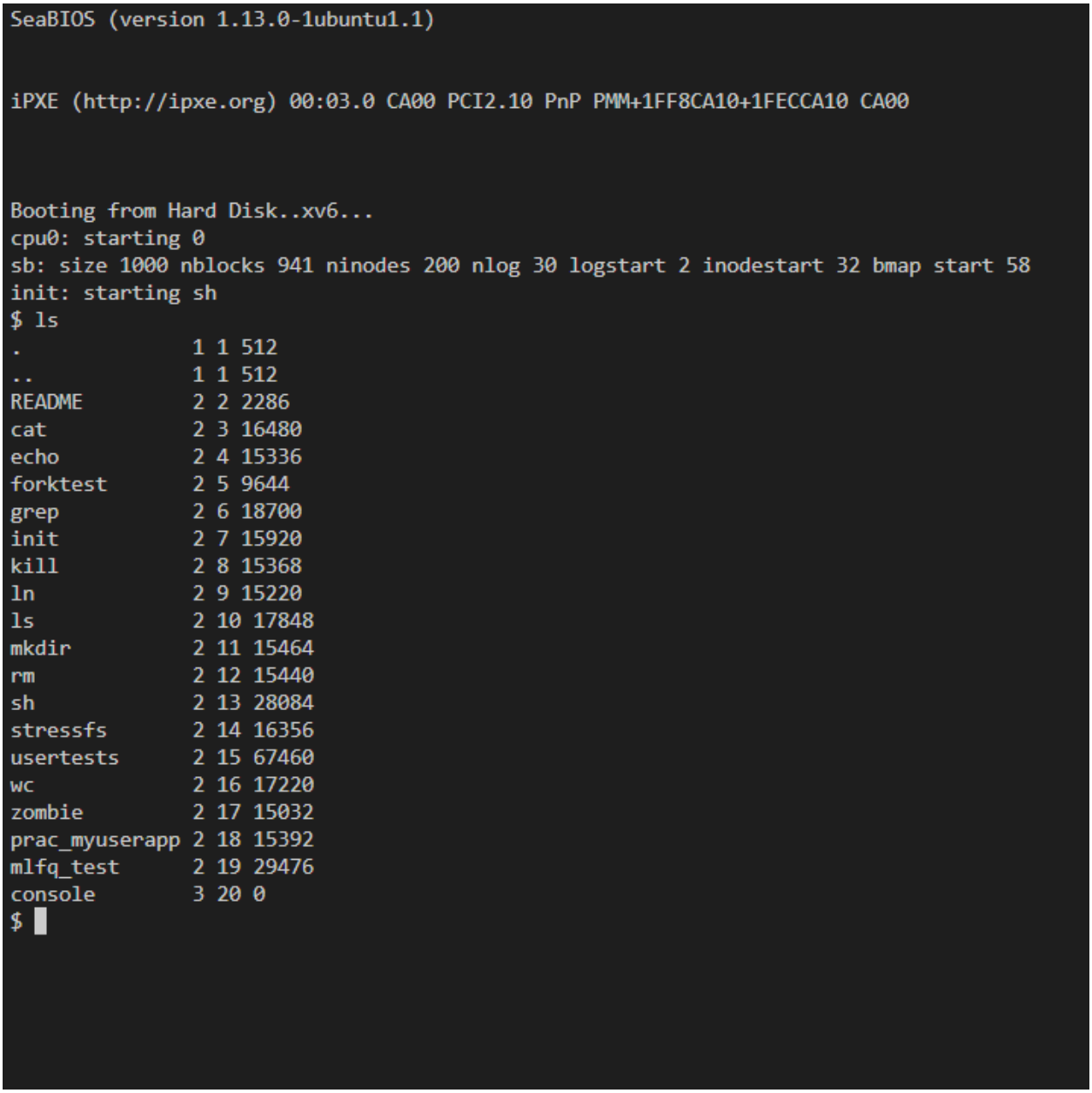
```
  }
...
```

128번과 129번이 발생하면 각각 `schedulerLock()` 과 `schedulerUnlock()`을 호출한다.

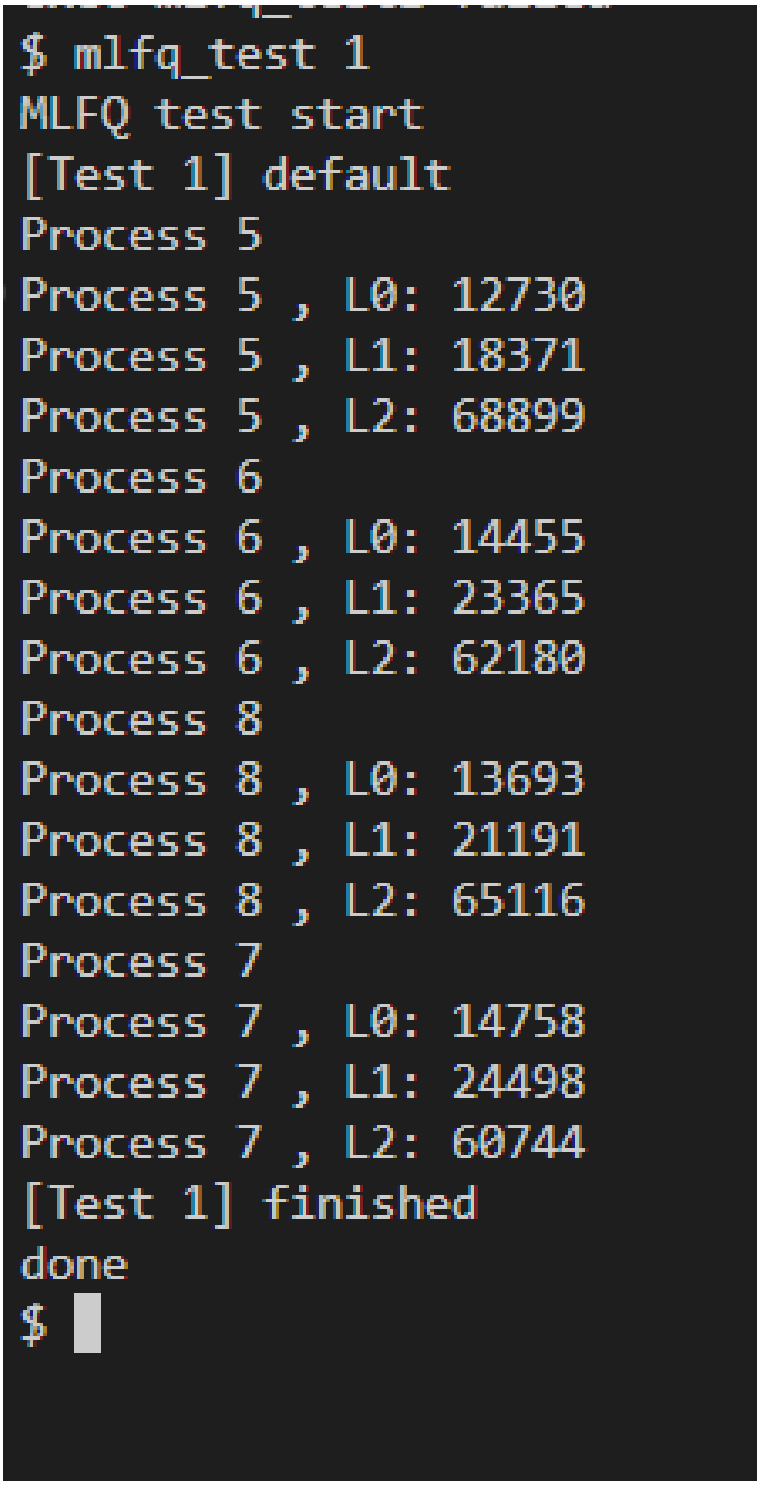
```
trap.c

...
case T_IRQ0 + IRQ_TIMER:
    if(cpuid() == 0){
      acquire(&tickslock);
      ticks++;
      global_tick++;
      wakeup(&ticks);
      release(&tickslock);
    }
    lapiceoi();

    if(myproc() && myproc()->state == RUNNING && tf->trapno == T_IRQ0+IRQ_TIMER) {
      myproc()->time_allotment++; // time_allotment 증가

    }

    break;
...

if(myproc() && myproc()->state == RUNNING &&
     tf->trapno == T_IRQ0+IRQ_TIMER)
    yield();
```

`ticks`가 올라가면 `global_tick`도 같이 올려주고 process가 `RUNNING`상태에서 `time interrupt`가 발생하면 해당 process의 `time_allotment`를 올려준다. 그 후 `yield()` 를 호출한다.

Result


SeaBIOS (version 1.13.0-1ubuntu1.1)
iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ ls
. 1 1 512
.. 1 1 512
README 2 2 2286
cat 2 3 16480
echo 2 4 15336
forktest 2 5 9644
grep 2 6 18700
init 2 7 15920
kill 2 8 15368
ln 2 9 15220
ls 2 10 17848
mkdir 2 11 15464
rm 2 12 15440
sh 2 13 28084
stressfs 2 14 16356
usertests 2 15 67460
wc 2 16 17220
zombie 2 17 15032
prac_myuserapp 2 18 15392
mlfq_test 2 19 29476
console 3 20 0
$

부팅 후 ls 입력한 화면

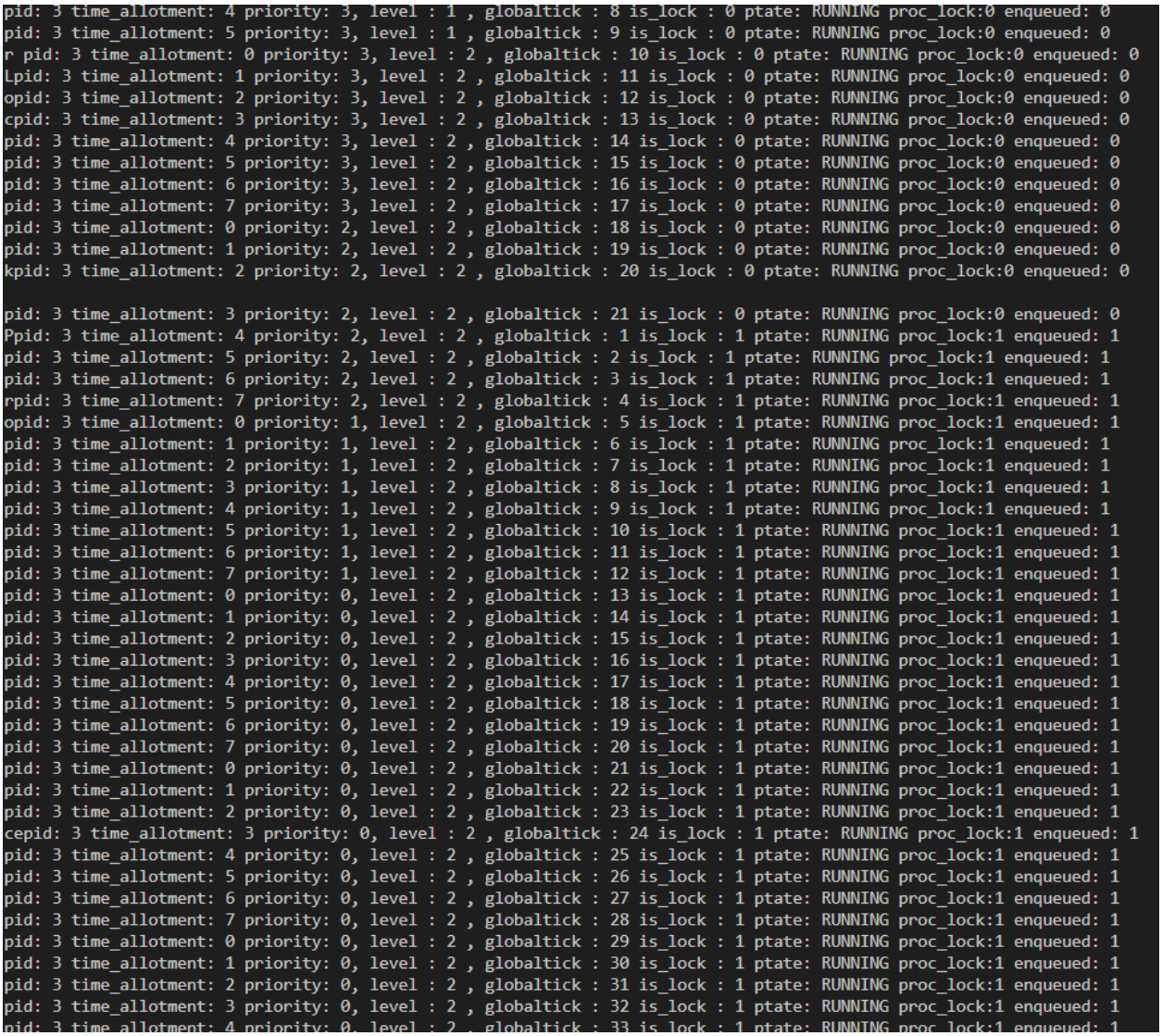


10만번 증가 연산 수행 결과

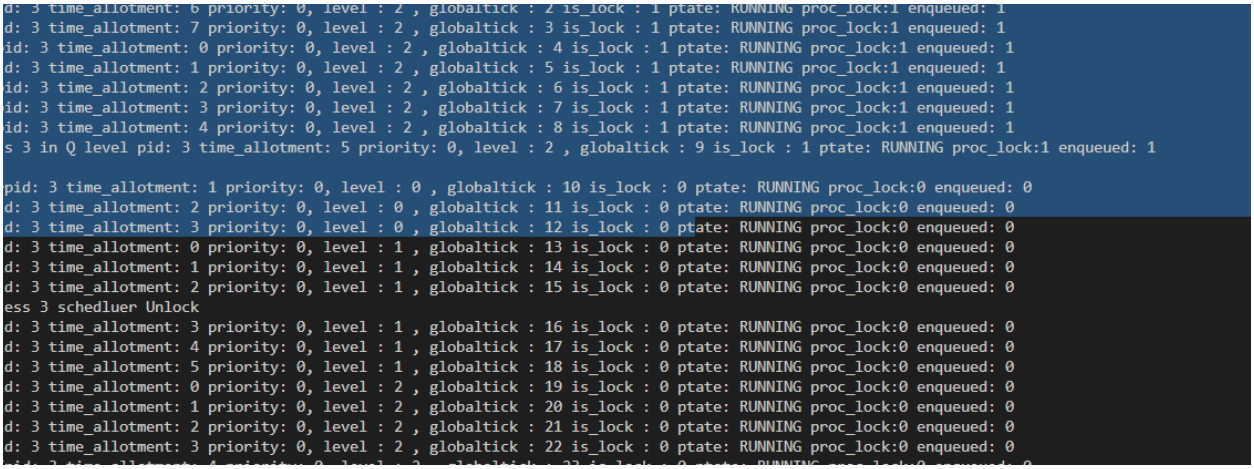
terminal에서 mlfq_test 1

여기서 보면 늦게 종료되는 process가 L2에 적게 있는 것을 알 수 있다. 이는 오래 실행될수록 priority boosting이 많이 일어나기 때문이다.

또한 가끔 L0와 L1의 담기는 값의 비율이 달라질때가 있는데 test를 실행하기 전 기존의 `global_tick`에 따라 priority boosting이 짧은 시간내에 일어날 수도 있기 때문이라고 생각하였다.



lock이 걸렸을 때 p의 `lock_scheduler`가 1이 된것을 확인할 수 있고 `global_tick`이 초기화 됨을 볼 수 있다.



lock이 해제 되면 `lock_scheduler`가 0이 되고 L0 queue에 맨앞으로 간 것을 확인할 수 있다.

print는 해당라인 에서 진행하였고,

```
proc.c

...
const char* state_to_string(enum procstate state) {
  switch (state) {
    case UNUSED:
      return "UNUSED";
    case EMBRYO:
      return "EMBRYO";
    case SLEEPING:
      return "SLEEPING";
    case RUNNABLE:
      return "RUNNABLE";
    case RUNNING:
      return "RUNNING";
    case ZOMBIE:
      return "ZOMBIE";
```

```
    default:
      return "UNKNOWN";
  }
}
...
```

p의 `state`는 다음과 같은 방식으로 출력하였다.

Trouble shooting

- global tick을 어디에 써줄지

`global tick`을 처음에 context switching 이 일어난 후 바꿔 주었는데 `sleeping` 상태 처럼 `time interrupt`가 발생하지 않는 경우를 고려하지 못했었다.

```
trap.c

...
case T_IRQ0 + IRQ_TIMER:
    if(cpuid() == 0){
      acquire(&tickslock);
      ticks++;
      global_tick++;
      wakeup(&ticks);
      release(&tickslock);
    }
...
```

이와 같이 `interrupt`가 발생하는 순간에 올려주는 것으로 해결했다.

- Lock을 걸었을때 좀비 처리

lock을 걸고 난 후 해당 process가 `zombie` 상태가 되는 경우가 종종 있었는데 이러한 특수 경우를

context switching이 일어 난 후 처리해 주었다.

```
proc.c

...
if (scheduler_locked == 1 && p->state != RUNNABLE) {
      //scheduler_lockde이 되어있는데 p가 RUNNABLE이 아니라면 lock을 해제함
      schedulerUnlock(2019041703);
    }
...
```

- Lock을 걸었을 때 queue와 dequeue

lock을 걸고 난 후 dequeue가 일어나지 않기 때문에 level이 변경되었을 때 기존 queue에서 빼주고 다시 변경된 level에 넣어주는 것을 생각 해야 했다.

```
proc.c

...
if (p->level < 2 && p->time_allotment >= p->time_quantum)
    {//Level 0과 1에서 timequantum 을 다 사용한 경우
      if(scheduler_locked==1)
      {//lock이 걸려있으면 우선 queue에서 빼줌
        dequeue_specific(&level_queue[p->level],p);
      }
...
```

- p=0 초기화 trouble

```
proc.c

...
if(scheduler_locked!=1)
    {//schedulerlock이 걸려있지 않다면 p초기화
      p = 0;
    }
...
```

scheduler 내에서 p를 초기화 해주지 않는다면 mlfq가 멈추는 오류가 발생했었다.

이유는 runnable한 process가 없는 경우 p가 전에 실행되었던 process를 가리켜 deadlock에 걸린 것이였다.

- Unlock 이 일어나는 경우

Unlock이 일어나는 경우가 총 3개 가 있다

1. process가 실행되는 도중 호출하는경우
2. lock을 건 process가 runnable하지 않은 경우
3. priority boosting 이 일어나는경우

이 세가지 경우에 dequeue와 L0 맨앞에 넣는 것을 모두 만족 시키지 않는다면 zombie exit 에러가 종종 발생하였다.

```
proc.c

...
if (p != 0) {
      scheduler_locked = 0;
      p->lock_scheduler = 0;

      if(p->state !=RUNNABLE){
         dequeue_specific(&level_queue[p->level],p);
      } //Runnable이 아니라면 Queue에서 제거
      if(p->state == RUNNABLE|| p->state==RUNNING){
        p->next = level_queue[0];
        level_queue[0] = p;
        p->level = 0;
        p->time_allotment = 0;
        p->time_quantum = 2 * p->level + 4;
        p->already_enqueued = 1;
      } //RUNNABLE 또는 Running이면 LO의 Head에 넣어주기
    }
...
```

이에 대한 처리를 확실하게 해줘야 했다. 우선 priority boosting 이 발생한 경우를 위해 `dequeue` 를 해주었고 `Runnable`하거나 `Running` 상태인 경우에 L0에 놓게 하였다. `Running` 상태를 확인함으로써 process가 실행중에 `schedulerUnlock()`을 호출한 경우를 처리해 줄 수 있었다.