Scheduler Report

201911146정소희

실행 결과

MLFQ

```
$ bench1
# of task: 10, Completion time of each task: 0 (tick)
Time from fork() to wait(): 3
(Child) Total response time: 12 (avg: 1), Total turnaround time: 12 (avg: 1)
```

```
$ bench2
# of task: 10, Completion time of each task: 4 (tick)
Time from fork() to wait(): 48
(Child) Total response time: 53 (avg: 5), Total turnaround time: 138 (avg: 13)
```

```
$ bench3
# of task: 10, Completion time of each task: 16 (tick)
Time from fork() to wait(): 168
(Child) Total response time: 92 (avg: 9), Total turnaround time: 1025 (avg: 102)
```

```
$ bench4
# of task: 10, Completion time of each task: [0,5,10,15,20,0,..] (tick)
Time from fork() to wait(): 108
(Child) Total response time: 68 (avg: 6), Total turnaround time: 450 (avg: 45)
```

```
$ bench5
# of task: 50, Completion time of each task: one [200] and other [0,3,6,9,12,0,..]
turnaround time of long task: 547
Time from fork() to wait(): 551
```

```
$ bench6
# of task: 2, Completion time of each task: [32] and [sleep(1)*1000] (tick)
child [0] begin: 2090
child [1] begin: 2091
child [0] end: 2127
child [1] end: 3099
Time from fork() to wait(): 1011
(0) response time: 1, turaround time: 38
(1) response time: 1, turaround time: 1009
(Child) Total response time: 2 (avg: 1), Total turnaround time: 1047 (avg: 523)
```

Round Robin

```
$ bench1
# of task: 10, Completion time of each task: 0 (tick)
Time from fork() to wait(): 8
(Child) Total response time: 17 (avg: 1), Total turnaround time: 17 (avg: 1)
```

```
$ bench2
# of task: 10, Completion time of each task: 4 (tick)
Time from fork() to wait(): 46
(Child) Total response time: 74 (avg: 7), Total turnaround time: 272 (avg: 27)
```

```
$ bench3
# of task: 10, Completion time of each task: 16 (tick)
Time from fork() to wait(): 166
(Child) Total response time: 91 (avg: 9), Total turnaround time: 1462 (avg: 146)
```

```
$ bench4
# of task: 10, Completion time of each task: [0,5,10,15,20,0,..] (tick)
Time from fork() to wait(): 108
(Child) Total response time: 68 (avg: 6), Total turnaround time: 549 (avg: 54)
```

```
$ bench5
# of task: 50, Completion time of each task: one [200] and other [0,3,6,9,12,0,..]
turnaround time of long task: 537
Time from fork() to wait(): 538
```

```
$ bench6
# of task: 2, Completion time of each task: [32] and [sleep(1)*1000] (tick)
childchild [1] begin: 2810
 [0] begin: 2809
child [0] end: 2846
child [1] end: 3814
Time from fork() to wait(): 1006
(0) response time: 1, turaround time: 38
(1) response time: 1, turaround time: 1005
(Child) Total response time: 2 (avg: 1), Total turnaround time: 1043 (avg: 521)
```

Mlfq는 짧은 job이 여러 개 들어오는 환경에서는 sjf와 같이 동작하고, 긴 job이 들어올땐 round robin과 유사합니다.

고로 bench1~4에서는 sjf처럼 동작하여 우수한 turnaround time을 보이고, bench5에서는 mlfq에서는

starvation 상황이 발생하므로, turnaround time for long task는 오히려 roundrobin보다 깁니다. Bench6에선 sleep을 하는 동안 높은 priority를 유지하므로 child1이 좀더 늦게 끝나기를 원했으나, sleep을 하는 동안 round robin처럼 mlfq도 priority에 상관없이 포기된 동안 rr로 child1이 동작하므로 거의 비슷한 움직임을 보입니다.

무엇을 바꾸었는지

Proc.h의 Struct proc에 run_ticks를 추가해 줍니다(run_ticks는 priority를 내리는데 사용되지 않습니다)

```
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 run_ticks;
  int dead;
};
```

proc.c 에서는 다음과 같은 queue를 준비해줍니다.

이 queue level은 priority서 쓰이게 되고, runed queue는 같은 priority에서 rr을 위해 사용됩니다.

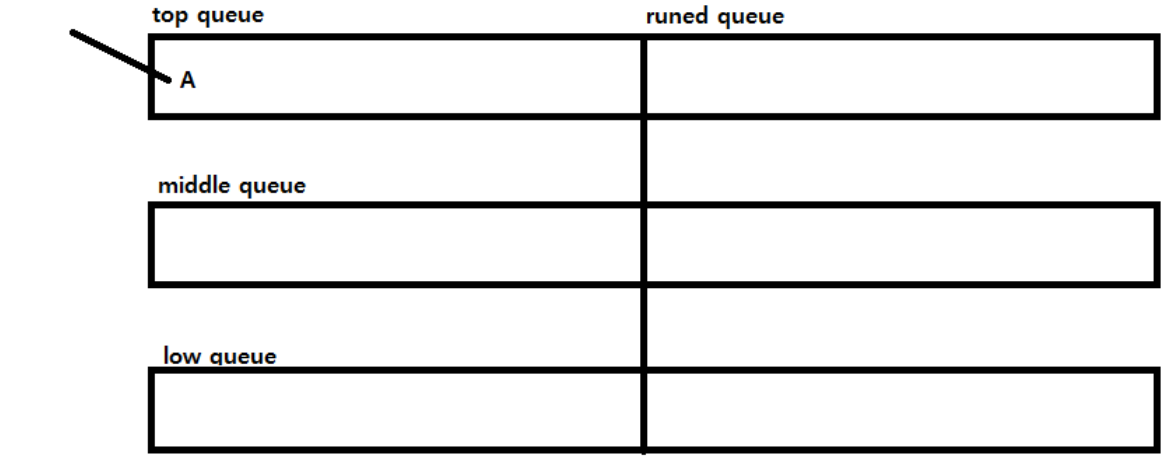
```
13
14  struct {
15    struct spinlock lock;
16    struct proc proc[NPROC];
17  } ptable;
18
19  static struct proc *initproc;
20
21  //write here
22  struct pstat pst;
23
24  struct proc* top_queue[1000];
25  struct proc* middle_queue[1000];
26  struct proc* bottom_queue[1000];
27  int top_num = 0;
28  int middle_num = 0;
29  int bottom_num = 0;
30
31  struct proc* top_queue_runed[1000];
32  struct proc* middle_queue_runed[1000];
33  struct proc* bottom_queue_runed[1000];
34  int top_num_run = 0;
35  int middle_num_run = 0;
36  int bottom_num_run = 0;
37  //write here
38
39  int nextpid = 1;
40  extern void forkret(void);
41  extern void trapret(void);
42
43  static void wakeup1(void *chan);
```

그리고 Allocproc에서 다음과 같이 값을 초기화해줍니다.

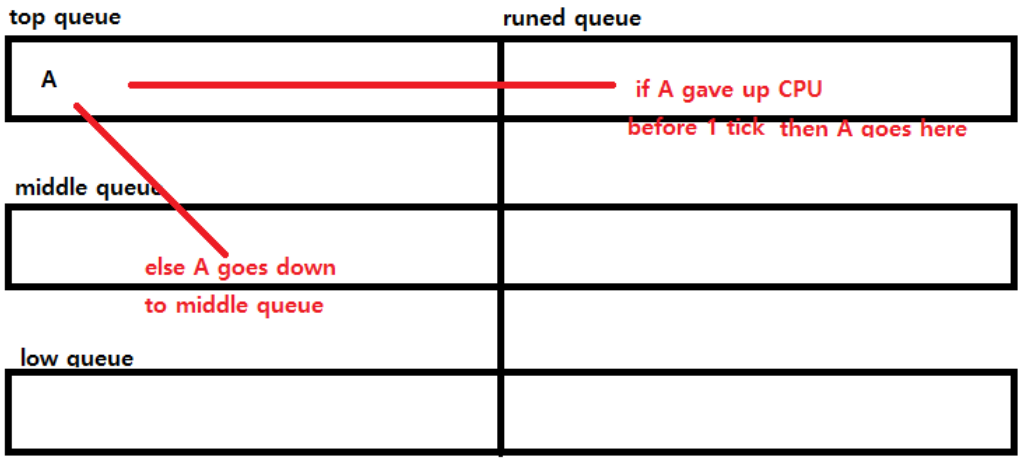
```
  p->state = EMBRYO;
  p->pid = nextpid++;
  p->run_ticks = 0;
  //write here
#ifdef DEBUG
  cprintf("\n\n%d'th proces made name \n\n",p->pid);
#endif
  p->dead = 0;
  top_queue[top_num] = p;
  pst.inuse[p->pid] = 1;
  pst.pid[p->pid] = p->pid;
  pst.priority[p->pid] = 0;
  pst.state[p->pid] = p->state;
  pst.ticks[p->pid][0] = 0;
  pst.ticks[p->pid][1] = 0;
  pst.ticks[p->pid][2] = 0;
  pst.wait_ticks[p->pid][0] = 0;
  pst.wait_ticks[p->pid][1] = 0;
  pst.wait_ticks[p->pid][2] = 0;
  top_num++;
  //write here
```

Priority 구현

1.



2.



실제 코드에서는 다음과 같이 implement했습니다.

```
if(pst.ticks[p->pid][0]==1){
    //cprintf("priority 0->1 downed\n");
    pst.priority[p->pid]++;
    middle_queue[middle_num] = p;
    middle_num++;
    for(int j=i;j<top_num-1;j++){
      top_queue[j] = top_queue[j+1];
    }
    top_num--;
  }
  else{
    //cprintf("priority 0 runed \n");
    pst.ticks[p->pid][0] = 0;
    p->run_ticks = 0;
    top_queue_runed[top_num_run] = p;
    top_num_run++;
    for(int j=i;j<top_num-1;j++){
      top_queue[j] = top_queue[j+1];
    }
    p->run_ticks = 0;
    top_num--;
```

1 tick을 다 소모하면 queue내 려주기

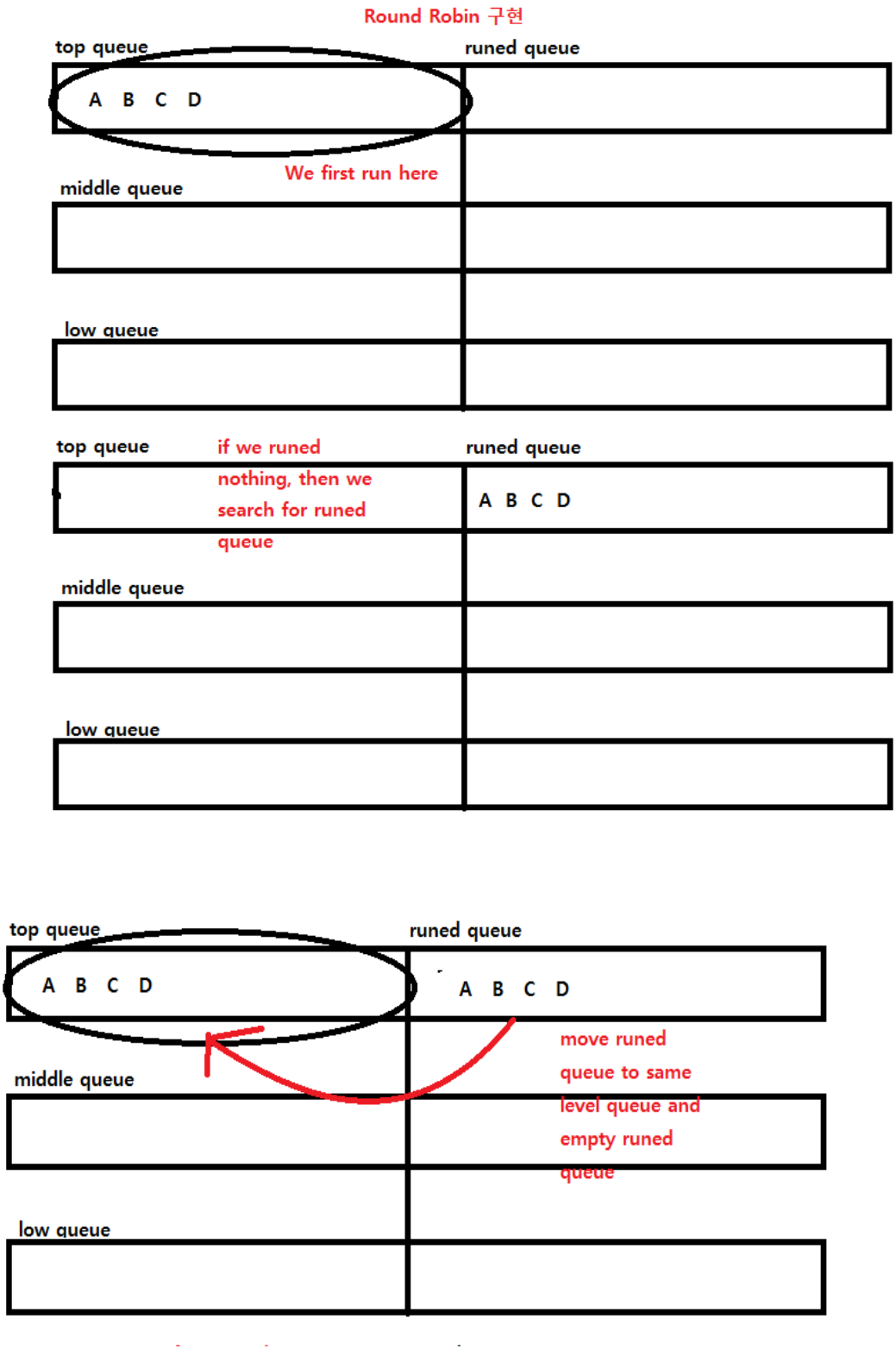
중간에 포기했으면 runed queue로 보내기

```
static void
wakeup1(void *chan)
{
  struct proc *p;

  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
    if(p->state == SLEEPING && p->chan == chan){
      int priority = pst.priority[p->pid];
      pst.ticks[p->pid][priority] = 0;
      p->state = RUNNABLE;
    }
}

// Wake up all processes sleeping on chan.
```

중간에 process가 포기했을 경우 경우 scheduler에서 wakeup을 걸어주게 되므로 이곳에서 tick을 0으로 만들어 줍니다.



실제 코드에서는 다음과 같이 implement 했습니다.

```
      middle_num--;
    }
    else if (p->run_ticks == 4){
      //cprintf("priority 1->1 runned\n");
      pst.ticks[p->pid][1] = 0;
      p->run_ticks = 0;
      middle_queue_runed[middle_num_run] = p;
      middle_num_run++;
      for(int j=i;j<middle_num-1;j++){
        middle_queue[j] = middle_queue[j+1];
      }
      middle_num--;
    }
    c->proc = 0;
    goto END;
  }
}
//cprintf("\ntick2-2\n");
if(middle_num_run>0){
  for(int i = 0;i<middle_num_run;i++){
    middle_queue[middle_num] = middle_queue_runed[i];
    middle_num++;
  }
  middle_num_run = 0;
  goto ENTRY;
}
```

4 tick을 돌았을 경우 runed queue로 돌아갑니다,
이 tick은 giveup과 상관없이 돌아간 tick수를 셉니다

돌지 않고 middle queue가 내려왔을 경우 돌았을 경우에는 End로 가서 다시 위로 올라감

올라가지 않고 이곳까지 도착했을경우 middle queue runed를 middle queue에 넣어주고 위로 올라감