

# OS project1 wiki

컴퓨터소프트웨어학부

2021018068 정환

## 1. Design

proc.c의 scheduler()함수를 바꾸는 것이 주요 과제였다. fcfs를 위한 queue와 mlfq를 위한 queue를 만들어 CPU scheduling을 수행하는 상황이 오면 schedmode에 따른 queue에서 프로세스를 꺼내 CPU를 할당한다. CPU scheduling은 scheduler() 함수를 통해 수행된다. 프로세스의 상태가 RUNNABLE이 되는 때에는 schedmode에 따른 queue에 enqueue를 수행한다. 구현의 세부 사항은 다음에서 설명한다.

## 2. Implementation

### a) proc.h

먼저 proc.h에 다음과 같이 추가했다.

```
83 | enum schedulermode { FCFS, MLFQ };
95 |     int level;                      // Process level (for MLFQ)
96 |     int priority;                   // Process priority (for MLFQ)
97 |     int timequantum;                // Time quantum (for MLFQ)
98 |     int creationtime;               // ticks value|
```

  

```
114 | struct queue {
115 |     struct proc *proc[NPROC];
116 |     int front;
117 |     int rear;
118 |     int size;
119 | };
```

이걸 기반으로 proc.c에 새로운 코드를 추가했다.

b) proc.c

```
29 enum schedulermode schedmode;
30 struct spinlock schedmode_lock;
31
32 struct queue fcfs_queue;
33 struct queue mlfq_queue[6]; // 0 : L0, 1 : L1, 2 ~ 5 : L2(priority 3 ~ 0)
34 struct spinlock fcfs_queue_lock;
35 struct spinlock mlfq_queue_lock;
36
37
38 int tickcount = 0;
39 struct spinlock tickcount_lock;
```

schedmode는 현재 scheduling 모드가 FCFS인지 MLFQ인지를 나타낸다. 다음으로 RUNNABLE상태인 프로세스들을 넣을 fcfs\_queue와 mlfq\_queue를 선언했다. mlfq\_queue에서 인덱스 3, 4, 5인 queue는 priority가 2, 1, 0인 level 2 queue이다.

그리고 priority boost의 조건을 위한 tickcount를 선언했다. 이 변수들의 race condition을 막기 위해 해당하는 spinlock들을 생성해주었다.

1) procinit()

```
322 schedmode = FCFS;
323 fcfs_init();
324 mlfq_init();
```

xv6가 부팅할 때 procinit()을 실행하는데 queue와 schedmode도 초기화해주기 위해 procinit()안에 위와 같이 넣어주었다.

```
41 void
42 fcfs_init(void)
43 {
44     initlock(&fcfs_queue_lock, "fcfs_queue_lock");
45     fcfs_queue.front = 0;
46     fcfs_queue.rear = -1;
47     fcfs_queue.size = 0;
48 }
50
51 void
52 mlfq_init(void)
53 {
54     initlock(&mlfq_queue_lock, "mlfq_queue_lock");
55     for(int i = 0; i < 6; i++) {
56         mlfq_queue[i].front = 0;
57         mlfq_queue[i].rear = -1;
58         mlfq_queue[i].size = 0;
59     }
60 }
```

각 init함수는 이와 같은 구조이다.

## 2) scheduler()

```
715  scheduler(void)
716  {
717      struct proc *p;
718      struct cpu *c = mycpu();
719
720      c->proc = 0;
721      for(;;){
722          // The most recent process to run may have had interrupts
723          // turned off; enable them to avoid a deadlock if all
724          // processes are waiting.
725          intr_on();
726
727          if(isEmptyQueue()){
728              asm volatile("wfi");
729              continue;
730          }
731
732          p = dequeue();
733          acquire(&p->lock);
734          // printf("[scheduler] Dequeued PID %d (level=%d, priority=%d)", p->pid, p->level, p->priority);
735          p->state = RUNNING;
736          c->proc = p;
737          swtch(&c->context, &p->context);
738
739          c->proc = 0;
740          release(&p->lock);
741      }
742  }
```

CPU scheduling이 필요할 때 사용되는 scheduler함수는 위와 같이 수정했다. ready 큐가 비어있지 않다면 프로세스 하나를 큐에서 빼고 상태를 RUNNING으로 만들어 context switch를 하여 실행한다.

## 3) allocproc()

```
389 |     p->creationtime = ticks;
```

새로운 프로세스를 생성할 때 creationtime을 ticks로 설정해 먼저 생성된 프로세스를 구별할 수 있게 하였다.

## 4) userinit(), fork(), wakeup(), kill()

```
879 |         enqueue(p);
```

프로세스의 상태를 RUNNABLE로 만드는 함수들 안에 enqueue(p)로 ready queue에 넣어주었다.

## 5) enqueue(proc)

```
144 void
145 enqueue(struct proc *p)
146 {
147     acquire(&schedmode_lock);
148     enum schedulermode mode = schedmode;
149     release(&schedmode_lock);
150
151     if(mode == FCFS) {
152         p->level = -1;
153         p->priority = -1;
154         p->timequantum = -1;
155         fcfs_enqueue(p);
156     } else if(mode == MLFQ) {
157         p->level = 0;
158         p->priority = 3;
159         p->timequantum = 2 * p->level + 1;
160         mlfq_enqueue(p);
161     } else {
162         panic("Unknown scheduling mode");
163     }
164 }
```

현재 schedmode에 따라 프로세스의 level, priority, timequantum을 초기화해주고 해당 queue에 넣는다. enqueue를 실행하는 모든 곳에 p의 lock이 이미 잡혀있다는 것을 주의했다.

## 6) fcfs\_enqueue(proc)

```
61 void
62 fcfs_enqueue(struct proc *p)
63 {
64     acquire(&fcfs_queue_lock);
65     if(fcfs_queue.size == NPROC) {
66         release(&fcfs_queue_lock);
67         panic("FCFS queue is full");
68     }
69     fcfs_queue.rear = (fcfs_queue.rear + 1) % NPROC;
70     fcfs_queue.proc[fcfs_queue.rear] = p;
71     fcfs_queue.size++;
72     release(&fcfs_queue_lock);
73
74     //printf("Enqueued PID %d (level=%d, priority=%d,
75 }
```

fcfs\_queue에 락을 걸어 동시 접근을 막고 fcfs\_queue에 인자로 받은 프로세스를 추가했다.

### 7) mlfq\_enqueue(proc)

```
79 void
80 mlfq_enqueue(struct proc *p)
81 {
82     int queueidx = p->level;
83
84     if(p->priority < 3) {
85         queueidx = 5 - p->priority; // idx will be 3 ~ 5
86     }
87
88     acquire(&mlfq_queue_lock);
89
90     if(mlfq_queue[queueidx].size == NPROC) {
91         release(&mlfq_queue_lock);
92         panic("MLFQ queue is full");
93     }
94
95     mlfq_queue[queueidx].rear = (mlfq_queue[queueidx].rear + 1) % NPROC;
96     mlfq_queue[queueidx].proc[mlfq_queue[queueidx].rear] = p;
97     mlfq_queue[queueidx].size++;
98
99     release(&mlfq_queue_lock);
100
101 //printf("Equeued PID %d (level=%d, priority=%d, state=%d)\n", p->pid,
102 }
103 }
```

priority와 level에 따른 mlfq\_queue의 인덱스를 계산하고, 해당하는 queue에 프로세스를 넣었다.

### 8) dequeue()

```
166 struct proc*
167 dequeue(void)
168 {
169     acquire(&schedmode_lock);
170     enum schedulermode mode = schedmode;
171     release(&schedmode_lock);
172
173     if(mode == FCFS) {
174         return fcfs_dequeue();
175     } else if(mode == MLFQ) {
176         return mlfq_dequeue();
177     } else {
178         panic("Unknown scheduling mode");
179     }
180 }
```

dequeue도 schedmode에 따라 구분하였다.

### 9) fcfs\_dequeue()

```
105 struct proc*
106 fcfs_dequeue(void)
107 {
108     acquire(&fcfs_queue_lock);
109
110     if(fcfs_queue.size == 0) {
111         release(&fcfs_queue_lock);
112         panic("FCFS queue is empty");
113     }
114
115     struct proc *p = fcfs_queue.proc[fcfs_queue.front];
116     fcfs_queue.front = (fcfs_queue.front + 1) % NPROC;
117     fcfs_queue.size--;
118     release(&fcfs_queue_lock);
119
120     return p;
121 }
```

특별한 내용 없이 제일 앞에 있는 프로세스를 꺼낸다.

#### 10) mlfq\_dequeue()

```
123 struct proc*
124 mlfq_dequeue(void)
125 {
126     for(int i = 0; i < 6; i++) {
127         acquire(&mlfq_queue_lock);
128
129         if(mlfq_queue[i].size > 0) {
130             struct proc *p = mlfq_queue[i].proc[mlfq_queue[i].front];
131             mlfq_queue[i].front = (mlfq_queue[i].front + 1) % NPROC;
132             mlfq_queue[i].size--;
133
134             release(&mlfq_queue_lock);
135
136             return p;
137         }
138         release(&mlfq_queue_lock);
139     }
140
141     panic("MLFQ queue is empty");
142 }
```

level 0에서부터 level2까지 탐색하여 level이 낮은 queue의 데이터를 꺼낸다.

#### 11) isEmptyQueue()

schedmode에 따라 fcfs\_queue와 mlfq\_queue가 비어있는지를 확인했다.

## 12) receiveTimerIntr()

기존은 타이머 인터럽트가 오면 yield()만을 수행했다. 이제는 FCFS일 때 오면 아무 일도 발생하지 않고, MLFQ일 때 오면 적절한 과정을 수행해주어야 한다. 그래서 trap.c에 타이머 인터럽트가 왔을 때 yield()를 해주는 부분에 receiveTimerIntr()라는 함수를 호출해주었다.

```
245 void
246 receiveTimerIntr(void)
247 {
248     acquire(&schedmode_lock);
249     enum schedulermode mode = schedmode;
250     release(&schedmode_lock);
251
252     if(mode == MLFQ) {
253         acquire(&tickcount_lock);
254         tickcount++;
255         if(tickcount == 50){
256             tickcount = 0;
257             release(&tickcount_lock);
258             priorityboost();
259             return;
260         }
261         release(&tickcount_lock);
262
263         struct proc *p = myproc();
264
265         acquire(&p->lock);
266         p->timequantum--;
267
268         if(p->timequantum == 0) {
269             if(p->level < 2) {
270                 p->level++;
271             } else {
272                 p->priority = (p->priority == 0) ? 0 : p->priority - 1;
273             }
274             p->timequantum = 2 * p->level + 1;
275
276             release(&p->lock);
277
278             yield();
279         } else {
280             release(&p->lock);
281         }
282     }
283 }
```

schedmode가 mlfq일 때만 작업을 수행한다. 먼저 mlfq에서 시간이 얼마나 지났는지를 확인하기 위한 tickcount를 증가한다. 만약 tickcount가 50이 되면 priorityboost과정을 수행한다. 아니라면 현재 실행중인 프로세스의 timequantum을 1감소하고 timequantum을 다 썼다면 level, priority, timequantum을 설정해준 뒤 yield()를 호출하여 cpu를 양보한다.

### 13) priorityboost()

```
224 void priorityboost(){
225     struct proc *p;
226
227     for(p = proc; p < &proc[NPROC]; p++) {
228         acquire(&p->lock);
229         if(!(p->state == UNUSED && p->state == ZOMBIE)) {
230             p->level = 0;
231             p->priority = 3;
232             p->timequantum = 2 * p->level + 1;
233         }
234         release(&p->lock);
235     }
236
237     for(int i = 1; i < 6; i++){
238         while(mlfq_queue[i].size != 0){
239             enqueue(mlfq_queue[i].proc[mlfq_queue[i].front]);
240             mlfq_queue[i].front = (mlfq_queue[i].front + 1) % NPROC;
241             mlfq_queue[i].size--;
242         }
243     }
244 }
245 }
```

모든 상태의 프로세스들이 있는 proc배열을 돌며 UNUSED와 ZOMBIE상태가 아닌 프로세스들의 level, priority, timequantum을 규칙에 맞게 바꿔준다. 그 후 level0에 있는 프로세스를 제외한 모든 level의 큐에 있는 프로세스를 level0으로 enqueue한다. 현재 실행중인 프로세스는 상태만 바꿔준 뒤 그대로 실행하게 둔다. 따라서 실행중인 프로세스는 바꿔준 timequantum만큼 실행한 뒤 level1로 이동하게 된다.

### 14) yield()

```
768 void
769 yield(void)
770 {
771     struct proc *p = myproc();
772
773     acquire(&schedmode_lock);
774     int mode = schedmode;
775     release(&schedmode_lock);
776
777     acquire(&p->lock);
778     p->state = RUNNABLE;
779     if(mode == FCFS){
780         //printf("yield process %d, level : %d\n", p->pid, p->level);
781         fcfs_enqueue(p);
782     } else if(schedmode == MLFQ){
783         mlfq_enqueue(p);
784     }
785     sched();
786     release(&p->lock);
787 }
788 }
```

yield함수는 실행중인 프로세스의 상태를 RUNNABLE로 바꾼 뒤 schedmode에 따라 queue에 넣는다. 그 후 sched()를 호출해 scheduler를 작동시킨다.

### 15) setpriority(pid, priority)

```
987 int setpriority(int pid, int priority)
988 {
989     struct proc *p = getprocess(pid);
990
991     if(p == 0) {
992         return -1;
993     }
994
995     if(priority < 0 || priority > 3) {
996         return -2;
997     }
998
999     acquire(&p->lock);
1000    p->priority = priority;
1001    // printf("pid : %d, priority : %d\n", pid, p->priority);
1002    release(&p->lock);
1003
1004    return 0;
1005 }
```

pid로 받은 프로세스의 priority를 인자로 받은 priority로 바꾼다. 이때 조건에 따라 값을 return 한다.

### 16) getprocess(pid)

```
969 struct proc*
970 getprocess(int pid)
971 {
972     struct proc *p;
973
974     for(p = proc; p < &proc[NPROC]; p++) {
975         acquire(&p->lock);
976         if(p->pid == pid) {
977             release(&p->lock);
978             return p;
979         }
980         release(&p->lock);
981     }
982
983     return 0;
984 }
```

pid를 인자로 받아서 해당하는 프로세스가 있다면 proc 포인터를 return하고 없으면 0을 return 한다.

### 17) getlev()

```
1008 |     getlev(void)
1009 | {
1010 |     if(schedmode == FCFS) {
1011 |         return 99;
1012 |     } else {
1013 |         struct proc *p = myproc();
1014 |
1015 |         acquire(&p->lock);
1016 |         int level = p->level;
1017 |         release(&p->lock);
1018 |
1019 |         return level;
1020 |     }
1021 | }
```

현재 schedmode가 FCFS라면 99를 return하고, MLFQ라면 해당 level을 return한다.

### 18) mlfqmode()

```
1023 |     int mlfqmode(void)
1024 | {
1025 |     acquire(&schedmode_lock);
1026 |
1027 |     if(schedmode == MLFQ) {
1028 |         printf("Scheduling mode is already MLFQ\n");
1029 |         release(&schedmode_lock);
1030 |         return -1;
1031 |     }
1032 |
1033 |     schedmode = MLFQ;
1034 |     release(&schedmode_lock);
1035 |
1036 |     acquire(&tickcount_lock);
1037 |     tickcount = 0;
1038 |     release(&tickcount_lock);
1039 |
1040 |     struct proc *runningp = myproc();
1041 |     acquire(&runningp->lock);
1042 |     runningp->level = 0;
1043 |     runningp->priority = 3;
1044 |     runningp->timequantum = 2 * runningp->level + 1;
1045 |     release(&runningp->lock);
1046 |
1047 |     while(!isEmptyfcfsQueue()) {
1048 |         struct proc *p = fcfs_dequeue();
1049 |         acquire(&p->lock);
1050 |         p->level = 0;
1051 |         p->priority = 3;
1052 |         p->timequantum = 2 * p->level + 1;
1053 |         mlfq_enqueue(p);
1054 |         release(&p->lock);
1055 |     }
1056 |
1057 |     return 0;
1058 | }
```

현재 schedmode가 FCFS일 때 MLFQ로 바꾸는 작업을 수행한다. tickcount를 0으로 초기화하고, state가 UNUSED와 ZOMBIE가 아닌 프로세스들의 level, priority, timequantum을 설정한다. 그 후 fcfs\_queue에 있는 모든 프로세스를 dequeue해서 mlfq\_queue로 enqueue한다.

### 19) fcfsmode()

```
1Click to add a description (void)
1065 {
1066     acquire(&schedmode_lock);
1067
1068     if(schedmode == FCFS) {
1069         printf("Scheduling mode is already FCFS\n");
1070         release(&schedmode_lock);
1071         return -1;
1072     }
1073
1074     schedmode = FCFS;
1075     release(&schedmode_lock);
1076
1077     struct proc *dequeuedProc[NPROC];
1078     int procCnt = 0;
1079
1080     while(!isEmptyMfqQueue()) {
1081         dequeuedProc[procCnt++] = mlfq_dequeue();
1082     }
1083
1084     struct proc *p;
1085
1086     for(p = proc; p < &proc[NPROC]; p++) {
1087         acquire(&p->lock);
1088         if(!(p->state == UNUSED && p->state == ZOMBIE)) {
1089             p->level = -1;
1090             p->priority = -1;
1091             p->timequantum = -1;
1092         }
1093         release(&p->lock);
1094     }
1095
1096     int isEnqueued[NPROC] = {0};
```

먼저 mlfq\_queue에 있는 모든 프로세스를 dequeue해서 dequeuedProc배열에 넣는다. 그 후 state가 UNUSED와 ZOMBIE가 아닌 프로세스들의 level, priority, timequantum을 -1로 설정한다.

```

1098    while(1) {
1099        struct proc *earlyP;
1100        int findEarlyP = 0;
1101        int earlyPCreationTime = 0;
1102        int earlyPIIdx = -1;
1103
1104        for(int i = 0; i < procCnt; i++) {
1105            p = dequeuedProc[i];
1106
1107            if(isEnqueued[i] == 1) {
1108                continue;
1109            }
1110
1111            acquire(&p->lock);
1112            int creationTime = p->creationtime;
1113            release(&p->lock);
1114
1115            if(findEarlyP == 0) {
1116                earlyP = p;
1117                earlyPIIdx = i;
1118                earlyPCreationTime = creationTime;
1119                findEarlyP = 1;
1120            } else {
1121                if(creationTime < earlyPCreationTime) {
1122                    earlyP = p;
1123                    earlyPIIdx = i;
1124                    earlyPCreationTime = creationTime;
1125                }
1126            }
1127        }
1128
1129        if(findEarlyP) {
1130            isEnqueued[earlyPIIdx] = 1;
1131            acquire(&earlyP->lock);
1132            fcfs_enqueue(earlyP);
1133            release(&earlyP->lock);
1134        } else {
1135            break;
1136        }
1137    }
1138
1139    return 0;
1140}

```

그 다음으로 dequeuedProc안에 있는 프로세스 중 creationtime이 가장 작은 걸 순서대로 fcfs\_queue에 넣는다.

### c) sysproc.c

```
95  uint64
96  sys_yield(void)
97  {
98      yield();
99      return 0;
100 }
101
102 uint64
103 sys_setpriority(void)
104 {
105     int pid;
106     int priority;
107     argint(0, &pid);
108     argint(1, &priority);
109
110     return setpriority(pid, priority);
111 }
112
113 uint64
114 sys_getlev(void)
115 {
116     return getlev();
117 }
118
119 uint64
120 sys_mlfqmode(void)
121 {
122     return mlfqmode();
123 }
124
125 uint64
126 sys_fcfsmode(void)
127 {
128     return fcfsmode();
129 }
```

system call로 호출하기 위한 함수를 위와 같이 만들었다. 만드는 과정은 assignment1과 같으므로 설명을 생략한다.

### 3. Results

```
$ test
FCFS & MLFQ test start

[Test 1] FCFS Queue Execution Order
Process 4 executed 100000 times
Process 5 executed 100000 times
Process 6 executed 100000 times
Process 7 executed 100000 times
[Test 1] FCFS Test Finished

Scheduling mode is already FCFS
nothing has been changed
successfully changed to MLFQ mode!

[Test 2] MLFQ Scheduling
Process 8 (MLFQ L0-L2 hit count):
L0: 10497
L1: 24369
L2: 65134
Process 9 (MLFQ L0-L2 hit count):
L0: 10876
L1: 23893
L2: 65231
Process 10 (MLFQ L0-L2 hit count):
L0: 9816
L1: 25193
L2: 64991
Process 11 (MLFQ L0-L2 hit count):
L0: 10332
L1: 23964
L2: 65704
[Test 2] MLFQ Test Finished

FCFS & MLFQ test completed!
$
```

test.c의 실행 결과이다. FCFS모드에서는 context switching이 거의 일어나지 않아 빠르지만 MLFQ모드에서는 context switching이 자주 일어나 수행속도가 느렸다.

```
[Test 1] FCFS Queue Execution Order
Equeued PID 4 (level=-1, priority=-1, state=3)
Equeued PID 5 (level=-1, priority=-1, state=3)
Equeued PID 6 (level=-1, priority=-1, state=3)
Equeued PID 7 (level=-1, priority=-1, state=3)
[scheduler] Dequeued PID 4 (level=-1, priority=-1)
Process 4 executed 100000 times
Equeued PID 3 (level=-1, priority=-1, state=3)
[scheduler] Dequeued PID 5 (level=-1, priority=-1)
Process 5 executed 100000 times
[scheduler] Dequeued PID 6 (level=-1, priority=-1)
Process 6 executed 100000 times
[scheduler] Dequeued PID 7 (level=-1, priority=-1)
Process 7 executed 100000 times
[scheduler] Dequeued PID 3 (level=-1, priority=-1)
[Test 1] FCFS Test Finished
```

enqueue와 dequeue될 때마다 print를 이용해 출력해보았을 때도 규칙에 맞는 과정으

로 CPU scheduling을 수행했다. queue에 들어온 순서대로 dequeue해서 실행하였다.

```
[scheduler] Dequeued PID 8 (level=2, priority=2)
Equeued PID 8 (level=2, priority=1, state=3)
[scheduler] Dequeued PID 9 (level=2, priority=2)
Equeued PID 9 (level=2, priority=1, state=3)
[scheduler] Dequeued PID 10 (level=2, priority=2)
Equeued PID 11 (level=0, priority=3, state=3)
Equeued PID 8 (level=0, priority=3, state=3)
Equeued PID 9 (level=0, priority=3, state=3)
Equeued PID 10 (level=1, priority=3, state=3)
[scheduler] Dequeued PID 11 (level=0, priority=3)
Equeued PID 11 (level=1, priority=3, state=3)
```

MLFQ 모드로 바꾼 후인 Test2의 수행 과정에 대한 print결과이다. time quantum을 다 썼을 때마다 level 2, priority 2에서 level 2, priority 1로 바꿔서 queue에 넣는 것을 볼 수 있다. 또한 pid 10인 프로세스가 실행되고 있을 때 priority boosting이 일어나서 level 0이 아닌 queue에 있던 11, 8, 9 프로세스를 level 0으로 넣는 것을 볼 수 있다. 현재 실행중인 10 프로세스도 level 0으로 상태를 변경하고 priority와 time quantum을 바꿔주었다. 바꾼 time quantum을 다 쓴 후 level 1인 큐에 들어가는 것을 볼 수 있다. 그 후에도 정상적으로 작동한다.

이 외에도 MLFQ에서 FCFS 모드로 변경하여 수행하였을 때도 정상작동하였다.

#### 4. Troubleshooting

proc.c에서 선언한 변수들과 프로세스의 lock을 고려하여 코드를 작성하는 것이 관건이였다. 전체적으로 lock을 잡고 푸는 사이의 과정을 최대한 짧게 만드는 것을 고려하였다. 또한 mlfq\_queue에서 level 2인 큐에서 priority에 따라 꺼내는 순서를 다르게 만드는 것을 queue의 규칙을 지키면서 어떻게 해야하나 고민했었다. 이 문제를 각 priority에 대한 queue를 또 만드는 것으로 해결하였다.