HW5. XV6 Priority Scheduler 구현하기

학번: 20170404

이름: 한종수

제출일: 2021. 4. 19

1.과제 개요

본 과제에서는 우선순위가 높은 순서대로 프로세스를 스케줄링 하는 Priority Scheduler 를 구현하고, test_sched1, test_sched2 코드를 통해 올바른 동작을 확인한다.

2.소스코드

(수정한 코드를 캡처하고 간단히 설명함)

1) scheduler

proc.c (/mnt/d/work/2021_Ope

```
320 //      via swtch back to the scheduler.
321 void
322 scheduler(void) {
323     struct proc *p;
324     struct proc *pp;
325     struct cpu *c = mycpu();
326     c->proc = 0;
327 
328     int MaxPriority = -1;   //HW#5 highest priortiy in table
329 
330     for (;;) {
331 
332         // Enable interrupts on this processor.
333         sti();
334 
335         // Loop over process table looking for process to run.
336         acquire(&ptable.lock);
337 
338         for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
339             //HW#5 run most highest priority process
340             for (pp = ptable.proc; pp < &ptable.proc[NPROC]; pp++) {
341                 if (pp->state != RUNNABLE)
342                     continue;
343                 if (pp->priority > MaxPriority) {
344                     MaxPriority = pp->priority;
345                 }
346             }
347 
348             if (p->state != RUNNABLE)
349                 continue;
350             if (p->priority < MaxPriority)
351                 continue;
352 
353             MaxPriority = -1;
354 
355             // Switch to chosen process.  It is the process's job
356             // to release ptable.lock and then reacquire it
357             // before jumping back to us.
358             c->proc = p;
359             switchuvm(p);
360             p->state = RUNNING;
361 
362             swtch(&(c->scheduler), p->context);
363             switchkvm();
364 
365             // Process is done running for now.
366             // It should have changed its p->state before coming back.
367             c->proc = 0;
368         }
369         release(&ptable.lock);
370 
371     }
372 }
373 
                                                  373,0-1       56%
```

scheduler context를 불러와서 scheduler가 실행될 때 338번 줄 루프에서 처음부터 ptable을 탐색하는 것이 아닌 context에 저장되어있던 process 부터 탐색한다. 루프 안쪽에서 가장 큰 priority 를 검색 해주고, context에 있던 process 기준으로 뒤의 process 중 maxpriority 이상 가진 것이 없으면, 다시 ptable 처음부터 검색을 시작하여 maxpriority를 가진 process를 실행한다.

324: pp는 340번 줄의 루프를 돌기 위해서 필요한 변수

339~346: 가장 큰 priority 값을 찾기 위한 루프

350~351: p->priority 가 MaxPriority보다 작으면 다음 프로세스를 조회한다.

2) fork

proc.c (/mnt/d/work/2021_Ope

```
179 int
180 fork(void) {
181     int i, pid;
182     struct proc *np;
183     struct proc *curproc = myproc();
184 
185     // Allocate process.
186     if ((np = allocproc()) == 0) {
187         return -1;
188     }
189 
190     // Copy process state from proc.
191     if ((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0) {
192         kfree(np->kstack);
193         np->kstack = 0;
194         np->state = UNUSED;
195         return -1;
196     }
197     np->sz = curproc->sz;
198     np->parent = curproc;
199     *np->tf = *curproc->tf;
200     np->priority = 5; //HW#5
201 
202     // Clear %eax so that fork returns 0 in the child.
203     np->tf->eax = 0;
204 
205     for (i = 0; i < NOFILE; i++)
206         if (curproc->ofile[i])
207             np->ofile[i] = filedup(curproc->ofile[i]);
208     np->cwd = idup(curproc->cwd);
209 
210     safestrcpy(np->name, curproc->name, sizeof(curproc->name));
211 
212     pid = np->pid;
213 
214     acquire(&ptable.lock);
215 
216     np->state = RUNNABLE;
217 
218     release(&ptable.lock);
219 
220     yield();
221 
222     return pid;
223 }
                                                      179,2        30%
```

200: Child process 의 priority를 중간 값인 5로 설정한다.

220: yield() 함수를 호출해 새로 스케쥴링한다.

3) trap

trap.c (/mnt/d/work/2021_Ope

```
63              kbdintr();
64              lapiceoi();
65              break;
66          case T_IRQ0 + IRQ_COM1:
67              uartintr();
68              lapiceoi();
69              break;
70          case T_IRQ0 + 7:
71          case T_IRQ0 + IRQ_SPURIOUS:
72              cprintf("cpu%d: spurious interrupt at %x:%x\n",
73                      cpuid(), tf->cs, tf->eip);
74              lapiceoi();
75              break;
76 
77              //PAGEBREAK: 13
78          default:
79              if (myproc() == 0 || (tf->cs & 3) == 0) {
80                  // In kernel, it must be our mistake.
81                  cprintf("unexpected trap %d from cpu %d eip %x (cr2=0x%x)\n",
82                          tf->trapno, cpuid(), tf->eip, rcr2());
83                  panic("trap");
84              }
85              // In user space, assume process misbehaved.
86              cprintf("pid %d %s: trap %d err %d on cpu %d "
87                      "eip 0x%x addr 0x%x--kill proc\n",
88                      myproc()->pid, myproc()->name, tf->trapno,
89                      tf->err, cpuid(), tf->eip, rcr2());
90              myproc()->killed = 1;
91      }
92 
93      // Force process exit if it has been killed and is in user space.
94      // (If it is still executing in the kernel, let it keep running
95      // until it gets to the regular system call return.)
96      if (myproc() && myproc()->killed && (tf->cs & 3) == DPL_USER)
97          exit();
98 
99      // Force process to give up CPU on clock tick.
100     // If interrupts were on while locks held, would need to check nlock.
101     //if(myproc() && myproc()->state == RUNNING &&
102     //   tf->trapno == T_IRQ0+IRQ_TIMER)
103     //  yield();
104 
105     // Check if the process has been killed since we yielded
106     if (myproc() && myproc()->killed && (tf->cs & 3) == DPL_USER)
107         exit();
                                                      107,9        98%
```

101~103: timer interrupt를 주석처리한다.

4) getnice setnice

proc.c (/mnt/d/work/2021_Ope

```c
543
544 int setnice(int pid, int nice) {
545     if (nice < 0 || nice > 10)
546         return -1;
547     if(pid < 1)
548         return -1;
549
550     struct proc *p;
551     acquire(&ptable.lock);
552     int priority = 0;
553     priority = 10 - nice;
554
555     /*************************/
556     /*       do program      */
557     /*************************/
558     for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
559         if (p->pid == pid) {
560             p->priority = priority;
561             release(&ptable.lock);
562             return 0;
563         }
564     }
565
566     myproc()->state = RUNNABLE;
567     release(&ptable.lock);
568
569     yield();    // Call Scheduler due to priority change
570
571     return -1; //non-existing pid
572 }
573
574 int getnice(int pid) {
575
576     if(pid < 1)
577         return -1;
578
579     struct proc *p;
580     acquire(&ptable.lock);
581
582     /*************************/
583     /*       do program      */
584     /*************************/
585     for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
586         if (p->pid == pid) {
587             release(&ptable.lock);
588             return (10 - p->priority);
589         }
590     }
591
592     release(&ptable.lock);
593
594     return -1; //non-existing pid
595
596 }
597
                                                 597,0-1       95%
```

552~553: nice value가 높으면 priority value 는 반대로 낮아진다. user로 부터 들어온 nice 값을 priortiy값으로 변경해 priority 변수에 저장한다.

569: priority 가 변경되었으므로 새로 스케줄링한다.

588: priority 값을 nice value로 변경하여 리턴한다.

3.결과

(테스트 실행결과를 캡처)

jongsoo@DESKTOP: ~/C

```
format=raw -smp 1 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 i
nodestart 32 bmap start 58
init: starting sh
$ test_sched1
#### State 1 ####
#### State 2 ####
PID 4 is finished
#### State 3 ####
PID 3 is finished
```

```
$ test_sched2
#### State 2 ####
PID 6 is finished
#### State 1 ####
#### State 3 ####
PID 5 is finished
$
```

test_sched1   test_sched2