Introduction to Operating Systems (SWE3004)

Project #2 – CPU Scheduling

Beomseok Nam (남범석)

Xv6 Process

- Process states (procstate in proc.h)

```
52 enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
53
54 // Per-process state
55 struct proc {
56   uint sz;                     // Size of process memory (bytes)
57   pde_t* pgdir;                // Page table
58   char *kstack;                // Bottom of kernel stack for this process
59   enum procstate state;        // Process state
60   int pid;                     // Process ID
```

- – UNUSED: Not used
- – EMBRYO: Newly allocated (not ready for running yet)
- – SLEEPING: Waiting for I/O, child process, or time
- – RUNNABLE: Ready to run
- – RUNNING: Running on CPU
- – ZOMBIE: Exited

Xv6 Process Scheduler

- main() in main.c

```
17 int
18 main(void)
19 {
39   userinit();      // first user process
40   // Finish setting up this processor in mpmain.
41   mpmain();
42 }
```

- mpmain() in main.c

```
54 // Common CPU setup code.
55 static void
56 mpmain(void)
57 {
58   cprintf("cpu%d: starting\n", cpu->id);
59   idtinit();       // load idt register
60   xchg(&cpu->started, 1); // tell startothers() we're up
61   scheduler();     // start running processes
62 }
```

Xv6 Process Scheduler

- scheduler() in proc.c
  - Round-robin fashion

```
265 void
266 scheduler(void)
267 {
268   struct proc *p;
269
270   for(;;){
271     // Enable interrupts on this processor.
272     sti();
273
274     // Loop over process table looking for process to run.
275     acquire(&ptable.lock);
276     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
277       if(p->state != RUNNABLE)
278         continue;
279
280       // Switch to chosen process.  It is the process's job
281       // to release ptable.lock and then reacquire it
282       // before jumping back to us.
283       proc = p;
284       switchuvm(p);
285       p->state = RUNNING;
286       swtch(&cpu->scheduler, proc->context);
287       switchkvm();
288
289       // Process is done running for now.
290       // It should have changed its p->state before coming back.
291       proc = 0;
292     }
293     release(&ptable.lock);
294
295   }
296 }
```

Xv6 Process Scheduler

- swtch() in swtch.S

```
 8 .globl swtch
 9 swtch:
10   movl 4(%esp), %eax
11   movl 8(%esp), %edx
12
13   # Save old callee-save registers
14   pushl %ebp
15   pushl %ebx
16   pushl %esi
17   pushl %edi
18
19   # Switch stacks
20   movl %esp, (%eax)
21   movl %edx, %esp
22
23   # Load new callee-save registers
24   popl %edi
25   popl %esi
26   popl %ebx
27   popl %ebp
28   ret
```

Xv6 Entering Scheduler

- sched() in proc.c

```
298 // Enter scheduler.  Must hold only ptable.lock
299 // and have changed proc->state.
300 void
301 sched(void)
302 {
303   int intena;
304
305   if(!holding(&ptable.lock))
306     panic("sched ptable.lock");
307   if(cpu->ncli != 1)
308     panic("sched locks");
309   if(proc->state == RUNNING)
310     panic("sched running");
311   if(readeflags()&FL_IF)
312     panic("sched interruptible");
313   intena = cpu->intena;
314   swtch(&proc->context, cpu->scheduler);
315   cpu->intena = intena;
316 }
```

Xv6 Entering Scheduler

- When?
  1. Exiting process (exit() in proc.c)

```
208   // Jump into the scheduler, never to return.
209   proc->state = ZOMBIE;
210   sched();
```

  2. Sleeping process (sleep() in proc.c)

```
371   // Go to sleep.
372   proc->chan = chan;
373   proc->state = SLEEPING;
374   sched();
```

Xv6 Entering Scheduler

- When?
  3. Yielding CPU due to timer interrupt
     - trap() in trap.c

```
103   // Force process to give up CPU on clock tick.
104   // If interrupts were on while locks held, would need to check nlock.
105   if(proc && proc->state == RUNNING && tf->trapno == T_IRQ0+IRQ_TIMER)
106     yield();
```

     - yield() in proc.c

```
318 // Give up the CPU for one scheduling round.
319 void
320 yield(void)
321 {
322   acquire(&ptable.lock);  //DOC: yieldlock
323   proc->state = RUNNABLE;
324   sched();
325   release(&ptable.lock);
326 }
```

Project Assignment #2

- Implement MLFQ scheduler on xv6
  - The lower nice value, the higher priority
  - The highest priority process is selected for next running
  - Tiebreak: round-robin fashion
  - More clarifications on the requirements will be announced later.

- Entering scheduler when
  1. Exiting process
  2. Sleeping process
  3. Yielding CPU
  4. Changing priority

Template Code

- git clone https://github.com/jinsoox/xv6-skku.git -b pa2

- Modifications
  - halt system call
    - Halt xv6 program
  - make tarball
    - Compress your source codes into one .tar.gz file for submission
    - You should enter your ID & project no. on Makefile
  - CPUS=1
  - Ignore to yield CPU on clock tick
  - yield system call
    - Yield CPU