XV6 SCHEDULING

Operating Systems

University of Tehran-Faculty of Computer Engineering

Fall 98

LET'S SEE HOW XV6 DOES SCHEDULING

- Main.c → scheduler() / Proc.c → scheduler() → Round Robin Implementation

```c
static void
mpmain(void)
{
  if(cpu() != mpbcpu())
    lapicinit(cpu());
  ksegment();
  cprintf("cpu%d: mpmain\n", cpu());
  idtinit();
  xchg(&c->booted, 1);

  cprintf("cpu%d: scheduling\n", cpu());
  scheduler();
}
```

```c
void
scheduler(void)
{
  struct proc *p;

  for(;;){
    // Enable interrupts on this processor, in lieu of saving intena.
    sti();

    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE)
        continue;

      // Switch to chosen process.  It is the process's job
      // to release ptable.lock and then reacquire it
      // before jumping back to us.
      cp = p;
      usegment();
      p->state = RUNNING;
      swtch(&c->context, &p->context);

      // Process is done running for now.
      // It should have changed its p->state before coming back.
      cp = 0;
      usegment();
    }
    release(&ptable.lock);

  }
}
```

LET'S SEE HOW XV6 DOES SCHEDULING: SCHEDULER() FUNCTION

- swtch(&c->context, &p->context);
  - Makes process "p" run in next time quantum by substituting context pointers
  - &c->context: pointer to current CPU scheduler context
  - &p->context: pointer to next running process context

- What happens if a process is paused by timer interrupt or is blocked by I/O operation?
- How can we pick another process to run?

LET'S SEE HOW XV6 DOES SCHEDULING: CHOOSING ANOTHER PROCESS

- Assume we have timer interrupt
  - Timer generates interrupt → Cause syscall to call a trap(implemented in trap.c)
  - Yield function is executed(implemented in proc.c)

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

Trap.c

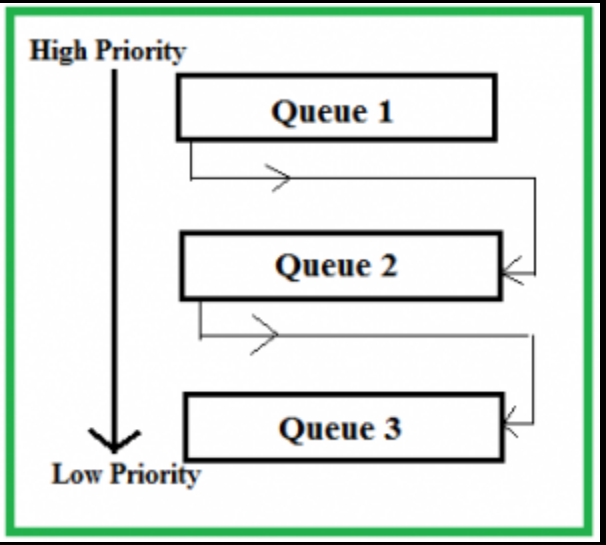
```
// Give up the CPU for one scheduling round.
void
yield(void)
{
  acquire(&ptable.lock);
  cp->state = RUNNABLE;
  sched();
  release(&ptable.lock);
}
```

Proc.c

This mentioned procedure is implementation of Round-Robin Scheduling in XV6!

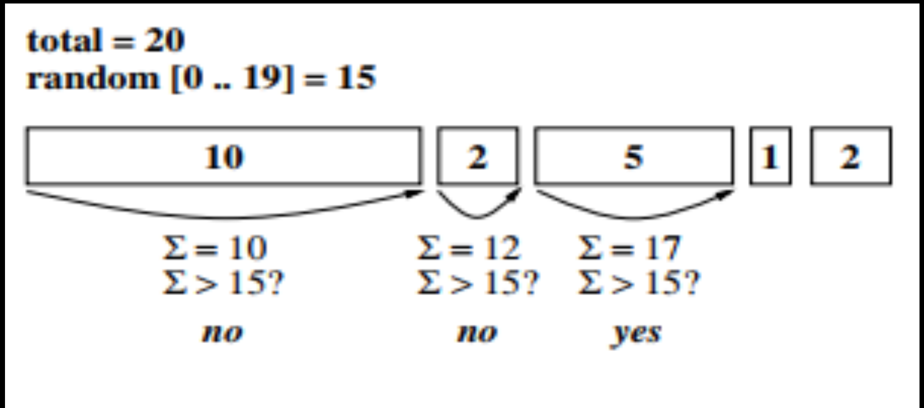
SCHEDULING MODIFICATIONS

- Round Robin → Multi Layer Scheduling
  - 1st level: Lottery Scheduling (First priority)
  - 2nd level: Highest Response Ratio Next (Second priority)
  - 3rd level: Shortest Remaining Priority First (Third priority)



LOTTERY SCHEDULING

- Generating lottery tickets for each process
- Generating random number in each interval and choose a process to run according to its lottery ticket numbers and generated random number
- Process with more lottery tickets is more probable to be chosen than process with less lottery tickets!



HRRN SCHEDULING

- You need to calculate a process waiting time and its executed cycles
- When a process executes, its executed cycle attributes increases 0.1 in magnitude, and the default value is set to 1
- The higher a process response ratio is, the higher the process chance is to be executed

$$HRRN = \frac{WaitingTime}{ExecutedCycleNumber}$$

$$WaitingTime = CurrentTime - ArrivalTime$$

SRPF SCHEDULING

- Every Process has an attribute named “Remaining Priority”
- Process with the lowest remaining priority is executed first
- When a process is executed, its remaining priority attribute will decrease 0.1 in magnitude
- Attention! The minimum magnitude of remaining priority is 0

COMPLEMENTARY SYSTEM CALLS

- 1. Change level of scheduling
- 2. Change remaining priority of processes in the last level
- 3. Assigning Lottery Tickets to 1st level processes
- 4. Listing all processes (helpful for your debugging)

Thank You for your attendance ☺️