SSE3044 Introduction to Operating Systems

Prof. Jinkyu Jeong

Project 2. CPU scheduling

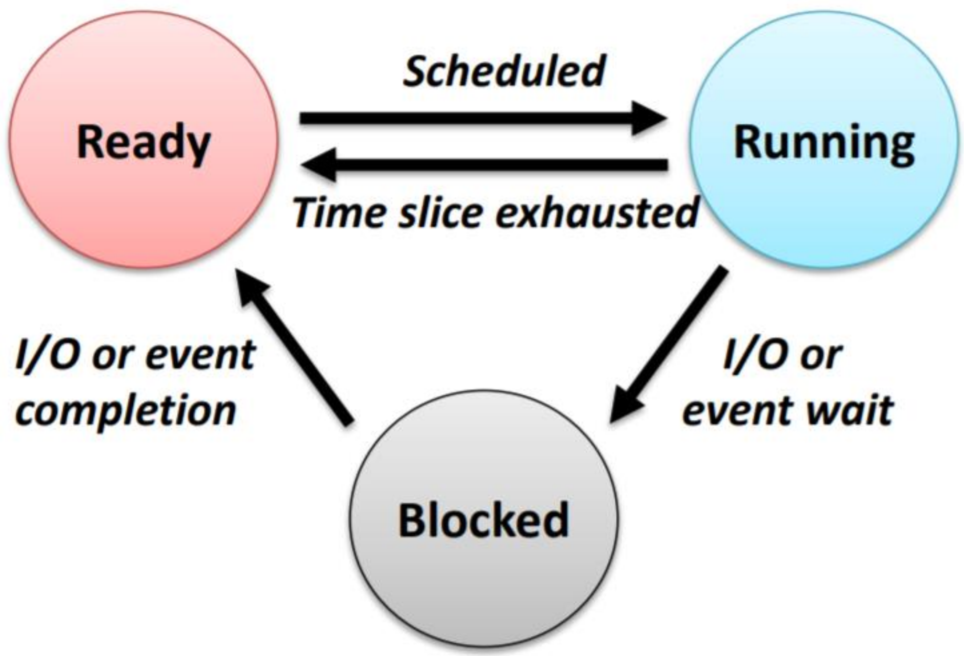
TA)
송원석(wonsuk.song@csi.skku.edu)
진승우(seungwoo.jin@csi.skku.edu)

Project plan

- Total 6 projects
  0) Booting xv6 operating system
  1) System call
  2) CPU scheduling
     - Linux CFS scheduler
  3) Virtual memory
  4) Page replacement
  5) File systems

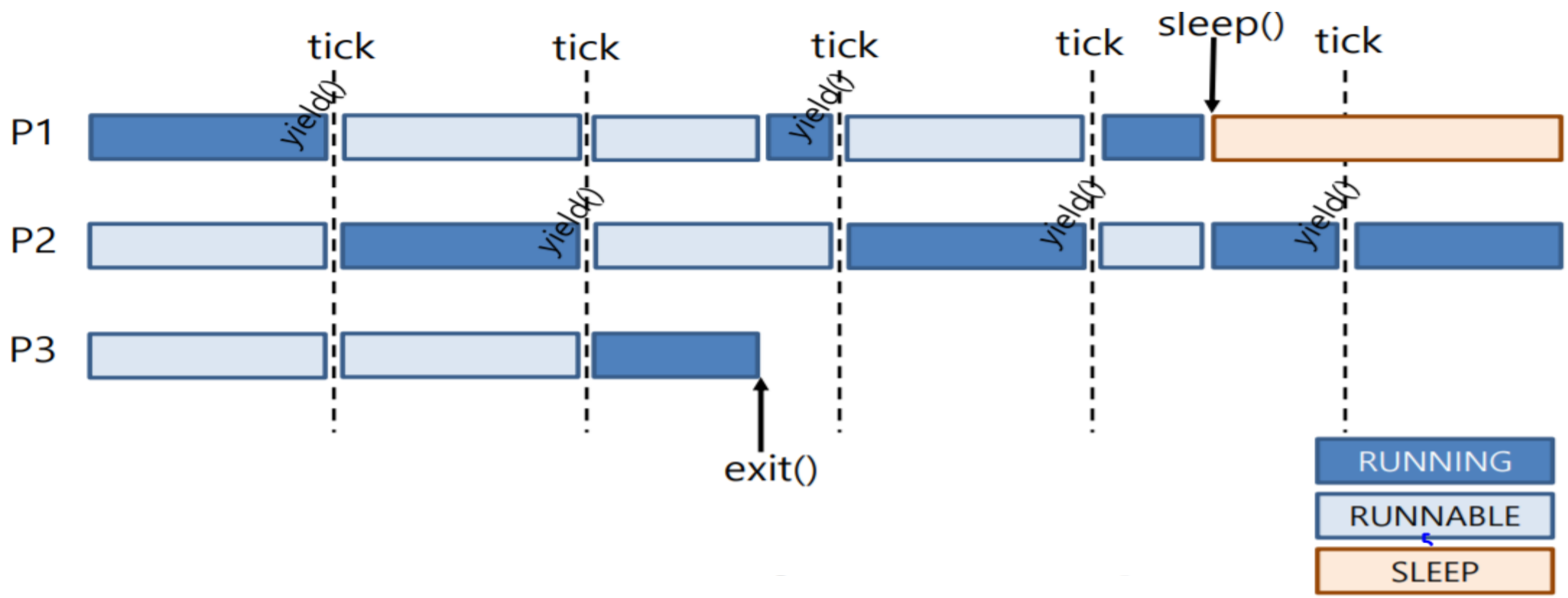
CPU scheduling

- Selects from among the processes in memory that are ready to execute, and allocates CPU to one of them



How current scheduler works in xv6?

- Every timer IRQ enforces an yield of a CPU
- Process to be scheduled to RUNNING will be chosen in round-robin manner



Strawman scheduler

- Organize all processes as a simple list
- In schedule():
  – Pick first one on list to run next
  – Put suspended task at the end of the list

- Problems?
  – Allows only round-robin scheduling
  – Can't prioritize tasks

Fair scheduling

- And, how should time slices be distributed according to priority?
  - The difference of time slice by the nice value is not fair
    - E.g, processes with nice value 0 and 1 are given 100ms and 95ms
    - Processes with nice value 18 and 19 are given 10ms and 5ms
  - The differences are same to 5ms, but it's not proportional
  - To solve this problem, From the Linux kernel 2.6.23, CFS(Completely Fair Scheduler) was used

CFS (Completely Fair Scheduling)

- Linux default scheduler
- Basic concept
  - The CPU is allocated to the process in proportion to its weight
  - CPU time of any task satisfies in any given time between t_1 and t_2

Weight of task

$$\boxed{C_{\tau_i}(t1, t2) = \frac{W(\tau_i)}{S_\Phi} \times (t_2 - t_1)}$$

CPU time

Total weight

Total CPU time

  - Nice to weight
    - Difference in nice by 1 provides 10% more (or less) CPU time
    - However, the larger the absolute value of nice, the smaller the ratio between the two values
    - Therefore, a new concept "weight"
    - Although there is formula, use pre-defined as array in Linux

$$Weight = 1024(weight\ of\ nice\ 0) \times (1.24)^{-NICE}$$

CFS parameters

- Time slice
  - Task's minimum time to be executed before it is preempted
  - Allocated to the process in proportion to its weight

$$time\ slice = scheduling\_latency * \frac{weight\ of\ task}{total\ weight\ of\ runqueue}$$

  - Scheduling latency (6ms by default)
    - Minimum time period to satisfy proportional CPU time distribution

- vruntime (virtual runtime)
  - accounts for how long a process has run proportional to its weight
  - It's easy to compare how fairly the CPU is allocated
  - By comparing this value, allowing you to select the next process to be performed

$$vruntime = actual\ runtime \times \frac{nice\ 0's\ weight}{weight\ of\ task}$$

CFS scheduling

1. A task with minimum virtual runtime is scheduled

2. Scheduled task gets time slice proportional to it's weight/total weight

3. During task running, virtual runtime is updated

4. After task runs more than time slice, go back to 1

Project 2. Implement CFS on xv6

- Implement CFS on xv6
  - Select process with minimum virtual runtime among runnable processes
  - Update runtime, vruntime every timer interrupt
  - If task runs more than time slice, enforces an yield of the CPU
  - Default nice value is 0, range from -5 to 5, and weight of nice 0 is 1024

  - Nice(-5~5) to weight(Although there is formula, use pre-defined as array like Linux)

$$Weight = 1024(weight\ of\ nice\ 0) \times (1.24)^{-NICE}$$

```
/*  -5 */    3121,    2501,    1991,    1586,    1277,
/*   0 */    1024,     820,     655,     526,     423,
/*   5 */     335,
```

  - Time slice calculation (our scheduling latency is 10ticks)

$$time\,slice = 10tick \times \frac{weight\,of\,current\,process}{total\,weight\,of\,runnable\,processes}$$

  - vruntime calculation

$$vruntime += \Delta\ runtime * \frac{weight\ of\ nice0(1024)}{weight\ of\ current\ process}$$

Project 2. Implement CFS on xv6

- How about forked process?
  - A process inherits the parent process's vruntime

- How about woken process?
  - When a process is woken up, it's virtual runtime gets (minimum vruntime of processes in the ready queue – vruntime(1tick))

$$vruntime(1tick) = 1tick * \frac{weight\ of\ nice0(1024)}{weight\ of\ current\ process}$$

- Do NOT call sched() during a wake-up of a process
  - Ensuring that the time slice of the current process expires
    - Woken-up process will have the minimum vruntime (due to the formula above)
    - But, we do NOT want to schedule the woken-up process before the time slice of current process expires.
  - This is by default in xv6

Project 2. Implement CFS on xv6

- To check that CFS is implemented properly, ps() should be modified
- Expected output (testcfs.c)

```
$ testcfs
=== TEST START ===
name            pid     state           priority        runtime/weight  runtime         vruntime        tick 3513000
init            1       SLEEPING        5               2               1000            1000
sh              2       SLEEPING        5               0               0               0
testcfs         3       RUNNABLE        5               1764            591000          589000
testcfs         4       RUNNING         0               1757            1800000         588600
```

  - Print out the following information about the processes
  - Use millitick unit (multiply the tick by 1000)
    - runtime, vruntime, total tick
      - Do NOT use float/double types to present runtime and vruntime
      - Kernel avoid floating point operation as much as possible

  - Indents of name section should be aligned even if process has long name (up to 10 letters) or very large value… (runtime, vruntime)

Project 2. Implementation details

- Project 2 should be done based on your project 1 code

- Consider the case of integer overflow vruntime
  - Even if over the scope of integer, shall operate without problems
  - And it has to be printed normally
  - Do not worry about runtime, total tick

- FAQ
  - Q : My time slice is 6.5. However, what if timer interrupt occurs every 1 tick? (context switch can occur only with 1 tick)
  - A : Tasks will run over it's time slice (7 ticks) & add vruntime

- Project 2 is very complex to implement, so please ask a lot of questions!

Submission

- Please implement CFS on xv6
- Compress your source code and upload on i-Campus
- How to compress your project
  - If you insert the user program, Modify the 'EXTRA' in Makefile
  - make dist
  - make tar
  - then, tar.gz file will be generated automatically
  - Rename to studentID-project2.tar.gz and upload in email

- Submit a report together, the file format of the report is limited to pdf
  - There is no limit to the format of the contents
  - But, include your description of your code and execution screen

Submission

- File format
  - StudentID-project2.tar.gz
  - StudentID-project2.pdf

- PLEASE DO NOT COPY
  - YOU WILL GET F GRADE IF YOU COPIED

- Due date: 4/21(Tue.), 23:59:59 PM
  - -25% per day for delayed submission

Questions

- If you have questions, please ask in Piazza

- You can also visit Semiconductor Building #400509
  - Please e-mail TA before visiting

- Reading xv6 commentary will help you a lot
  - http://csl.skku.edu/uploads/SSE3044S20/book-rev11.pdf