SWE3004 Operating Systems, Spring 2023

Project 2. CPU Scheduling

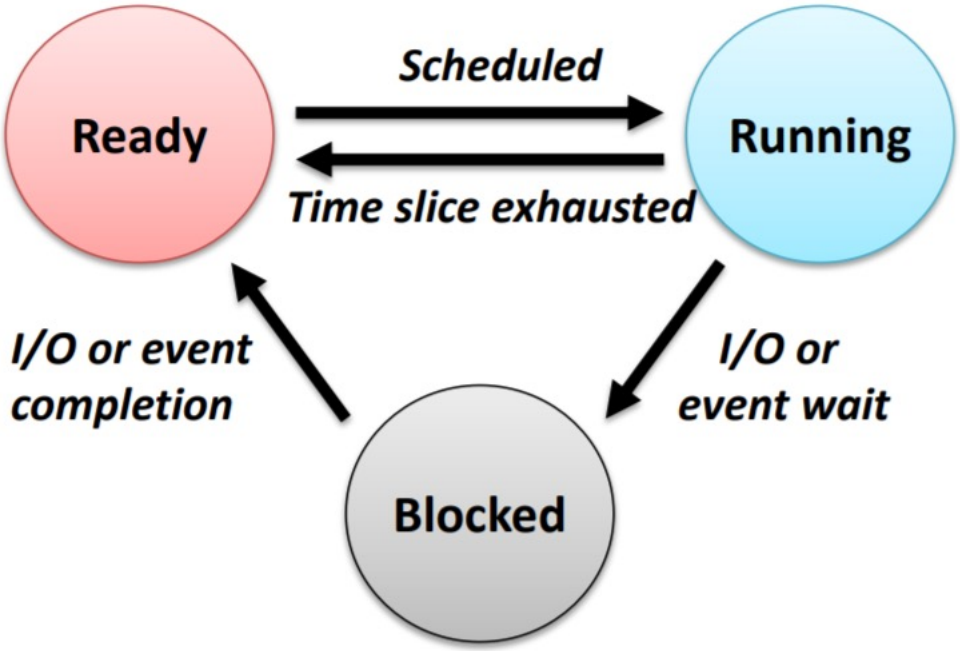
TA)
Hyeonmyeong Lee
Gwanjong Park
Younghoon Jun
Shinhyun Park
Jinwoo Jung

Project plan

- Total 6 projects
  - ~~0) Booting xv6 operating system (10pt)~~
  - ~~1) System call (15pt)~~
  - 2) CPU scheduling (25pt)
    - Linux CFS scheduler
  - 3) Virtual memory (25pt)
  - 4) Page replacement (25pt)
  - 5) File systems (0pt, optional)

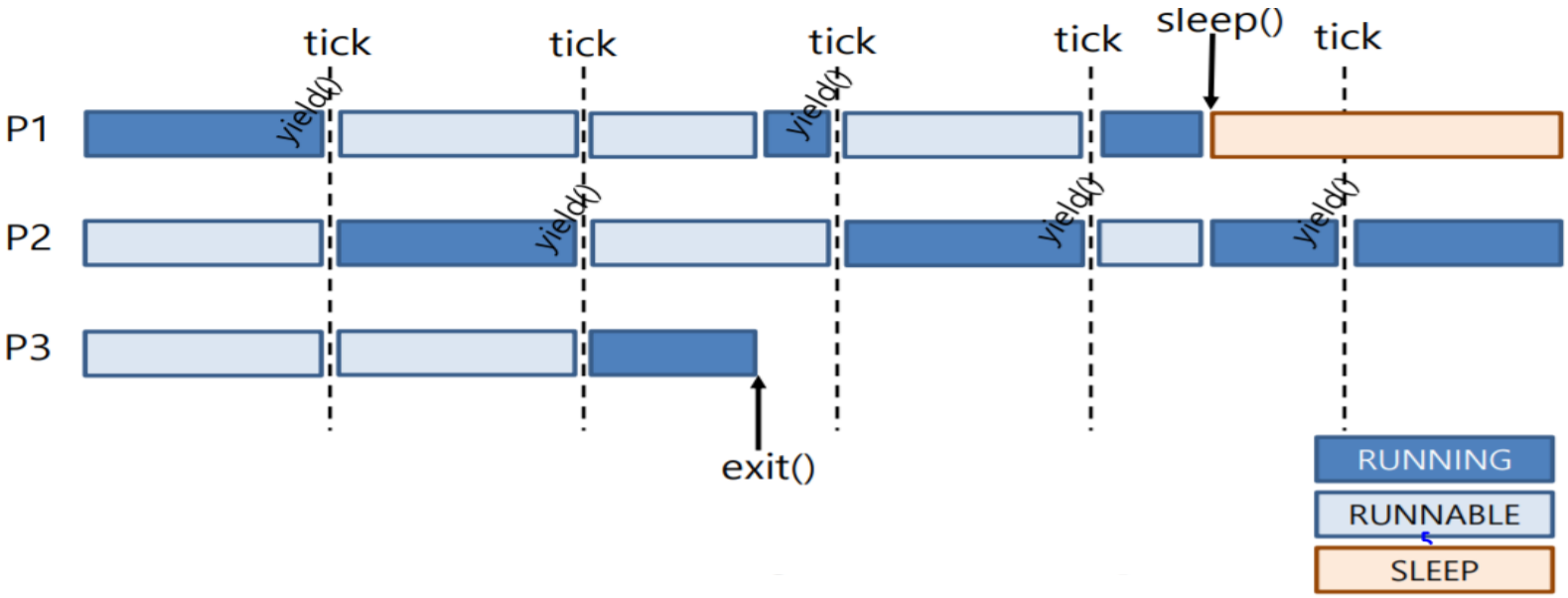
CPU scheduling

- Selects from 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 a yield of a CPU
- Process to be scheduled to be RUNNING state will be chosen in round-robin manner



Strawman scheduler

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

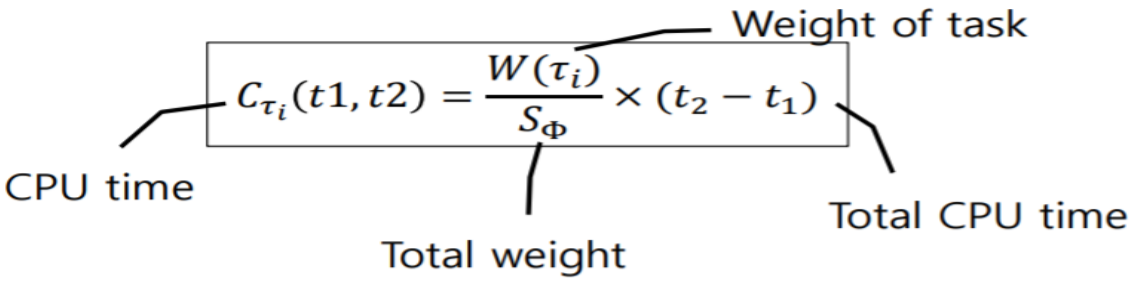
- Problems?
  - Allows only round-robin scheduling
  - Can't prioritize tasks

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 20 and 21 are given 100ms and 95ms
    - Processes with nice value 38 and 39 are given 10ms and 5ms
  - The differences are same to 5ms, but it's not proportional
  - To solve this problem, CFS(Completely Fair Scheduler) has been used since Linux kernel 2.6.23.

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



  - 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, hard-code pre-defined array like Linux

$$weight = 1024(weight\ of\ nice\ 20) \times (1.25)^{-(nice-20)}$$

CFS parameters

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

$$time\_slice = scheduling\_latency \times \frac{weight\ of\ task}{total\ weight\ of\ runqueue}$$

  - 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, you can select the next process to be scheduled

$$vruntime = (actual\ runtime) \times \frac{weight\ of\ nice\ 20\ (1024)}{weight\ of\ task}$$

CFS scheduling

1. A task with minimum virtual runtime is scheduled

2. Scheduled task gets time slice proportional to its {weight / total weight}

3. While the task is 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 from runnable processes
  - Update runtime/vruntime for each timer interrupt
  - If task runs more than time slice, enforce a yield of the CPU
  - Default nice value is 20, ranging from 0 to 39, and weight of nice 20 is 1024
  - Nice(0~39) to weight(Although there is formula, hard-code pre-defined array like Linux)

$$weight = \frac{1024}{(1.25)^{nice-20}}$$

```
/*  0 */    88761,    71755,    56483,    46273,    36291,
/*  5 */    29154,    23254,    18705,    14949,    11916,
/* 10 */     9548,     7620,     6100,     4904,     3906,
/* 15 */     3121,     2501,     1991,     1586,     1277,
/* 20 */     1024,      820,      655,      526,      423,
/* 25 */      335,      272,      215,      172,      137,
/* 30 */      110,       87,       70,       56,       45,
/* 35 */       36,       29,       23,       18,       15,
```

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

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

  - vruntime calculation

$$vruntime \mathrel{+}= \Delta runtime \times \frac{weight\ of\ nice\ 20\ (1024)}{weight\ of\ current\ process}$$

Project 2. Implement CFS on xv6

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

$$vruntime(1tick) = 1tick \times \frac{weight\ of\ nice\ 20\ (1024)}{weight\ of\ current\ process}$$

(If there is no process in the RUNNABLE state when a process wakes up, you can set the vruntime of the process to be woken up to "0")

- DO NOT call sched() during a wake-up of a process
  - Ensure that the time slice of the current process expires
    - Woken-up process will have the minimum vruntime (by 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 if CFS is implemented properly, `ps()` should be modified
- Expected output (`mytest.c`)

```
$ mytest
=== TEST START ===
name      pid       state      priority       runtime/weight   runtime        vruntime           tick 5488000
init      1         SLEEPING   5              0                4000           2000
sh        2         SLEEPING   5              0                1000           0
mytest    4         RUNNABLE   5              28               832000         38770
mytest    5         RUNNING    0              37               3370000        38755
```

  - 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 must 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

- Use the ***submit & check-submission*** binary file in Ji Server
  - make clean
  - $ ~swe3004/bin/submit pa2 xv6-public
  - you can submit several times, and the submission history can be checked through check-submission
    - Only the last submission will be graded

Submission

- Report
  - Submit to iCampus
  - Modified code and explanation
  - Free length
  - pa2_2023123456.pdf

Submission

- PLEASE DO NOT COPY
  - We will run inspection program on all the submissions
  - Any unannounced penalty can be given to **both students**
    - 0 points / negative points / F grade …

- Due date: 4/12(Wed.), 23:59:59 PM
  - **-25%** per day for delayed submission

Questions

- If you have questions, please ask on i-campus discussion section
  - Please use the discussion board
  - Discussion board preferred over messages

- You can also visit Corporate Collaboration Center #85533
  - Please iCampus message TA before visiting

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