

SWE3004 Operating Systems, Fall 2024

Project 2. CPU Scheduling

TA)

Osama Khan

Gwanjong Park

Chanu Yu

Shinhyun Park

Project plan

- Total 6 projects

0) Booting xv6 operating system

+1) System call

2) CPU scheduling

- Linux CFS scheduler

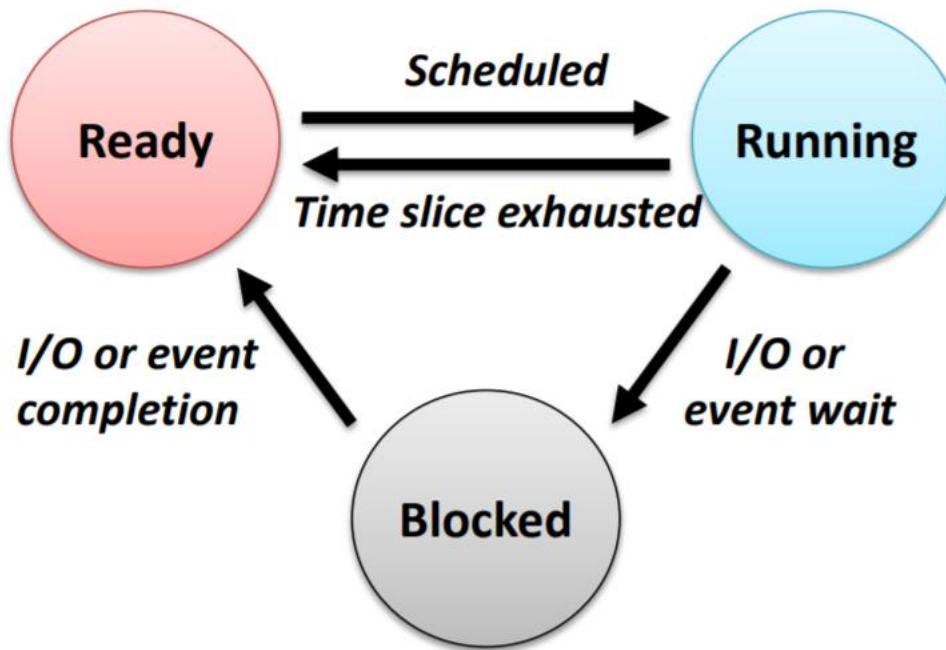
3) Virtual memory

4) Page replacement

5) File systems

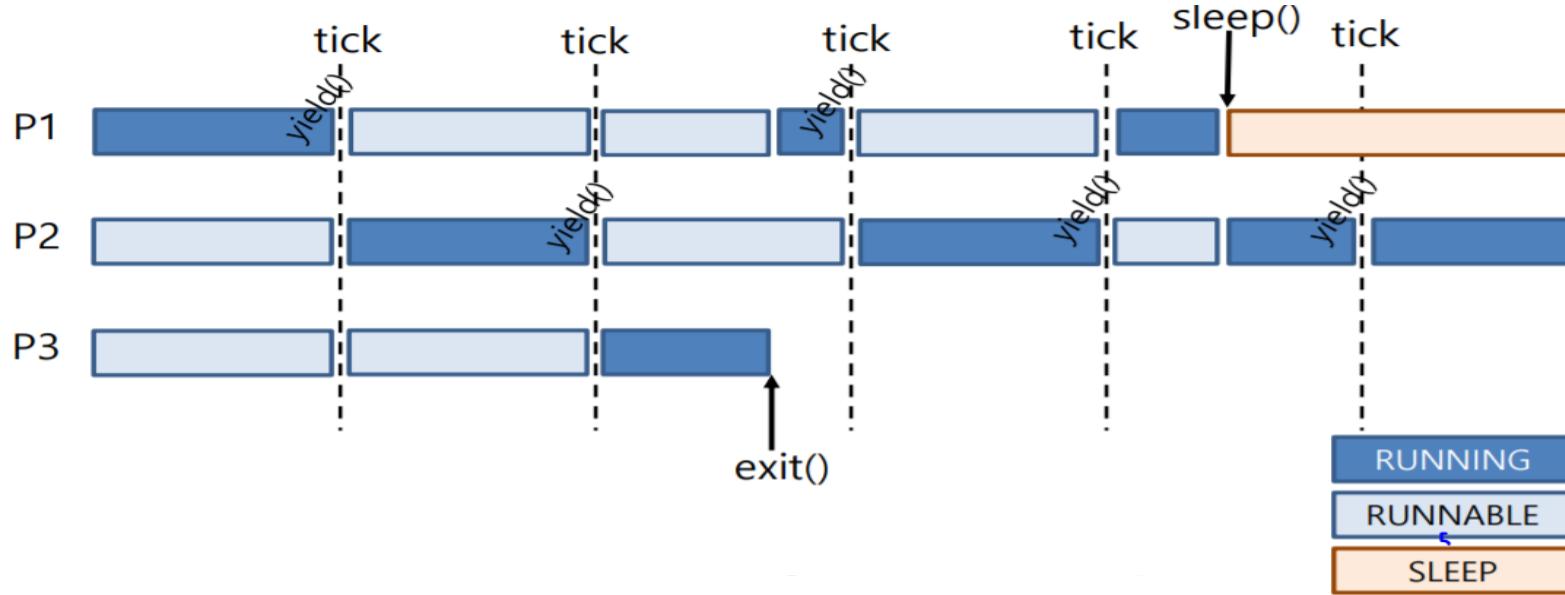
CPU scheduling

- Selects from 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

The diagram shows a horizontal rectangle representing CPU time. A bracket at the top is labeled "Weight of task". A bracket at the bottom is labeled "Total weight". A bracket on the right side is labeled "Total CPU time". Inside the rectangle, the formula $C_{\tau_i}(t_1, t_2) = \frac{W(\tau_i)}{S_\Phi} \times (t_2 - t_1)$ is written, with arrows pointing from each label to its corresponding part in the formula.

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

- 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

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

CFS parameters

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

- In this project, you need to implement the following

I. Implement CFS on xv6

- CFS must operate well so that runtime increases in accordance with priority
- vruntime and time slice must be properly calculated
- Upon wake up, the defined rule must be strictly followed

2. Modify ps system call to output appropriate value

- runtime/weight, runtime, vruntime, and total tick
- We base our scoring on the output printed by ps()
 - Even if CFS is well implemented, if ps fails to properly display the values, you may not receive a score

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**)

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

- vruntime calculation

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

Project 2. Implement CFS on xv6

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

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

(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. Modify ps System Call

- To check if CFS is implemented properly, ps () should be modified
- Sample 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/weight, runtime, vruntime, total tick
 - Do NOT use float/double types to present runtime and vruntime
 - Kernel avoid floating point operation as much as possible
- There's no need for the output to match the sample exactly
- Check whether the runtime corresponds with the priority and whether the vruntime of the processes is similar

Project 2. FAQ

- Project 2 should be done based on your project 1 code
 - Please refer to the trap.c file for anything related to timer interrupts
 - This project is not related to future projects
-
- You don't need to consider situations where runtime or vruntime is too large (exceeding the range of int)
 - The vruntime formula on page 8 is for conceptual explanation.
Please refer to page 11 for the actual implementation.
-
- You don't need to worry about anything related to exec()
 - Do not worry about runtime at the time of wakeup

Submission

- Please implement CFS on xv6 and modify ps()
- Use the ***submit*** & ***check-submission*** binary file in Ui 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

- 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: 10/30(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-revII.pdf>