SWE3004 Operating Systems, Fall 2025

Project 2. CPU Scheduling

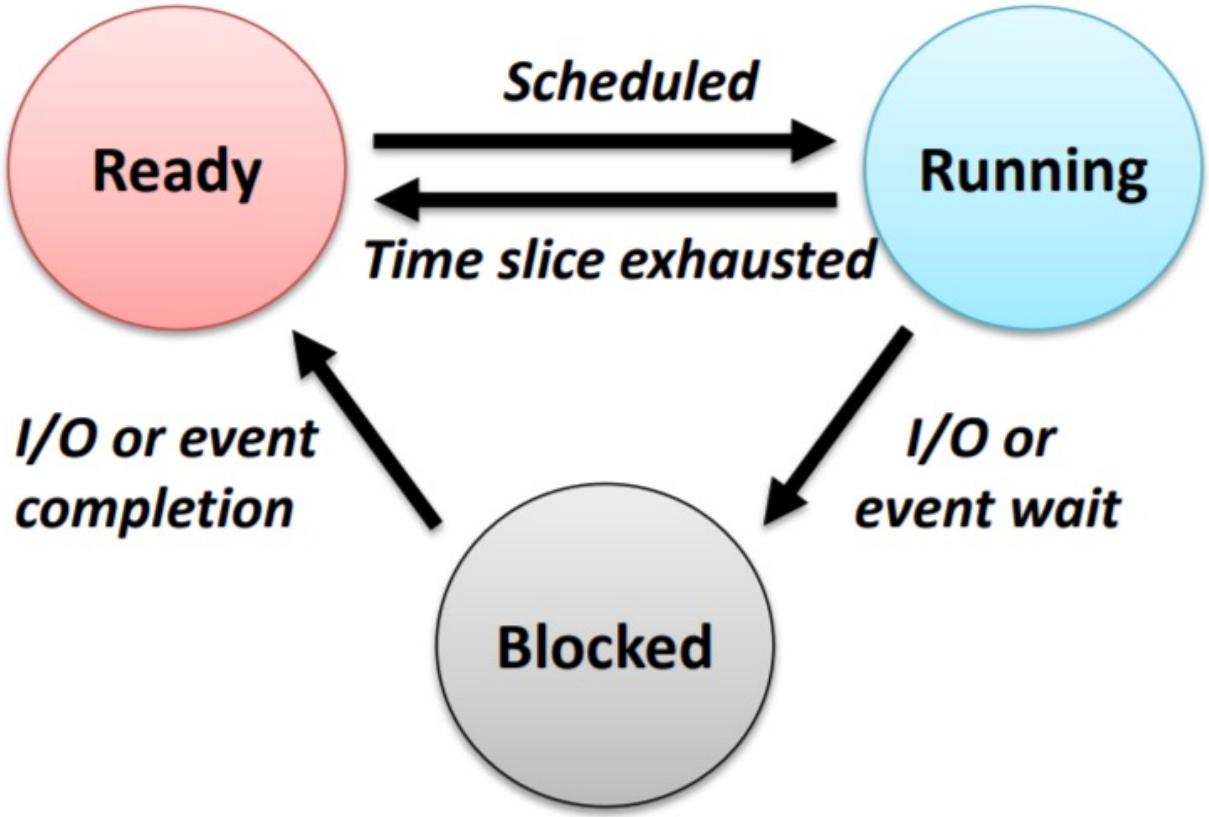
TA)
Gwanjong Park
Yunseong Shin

Project plan

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

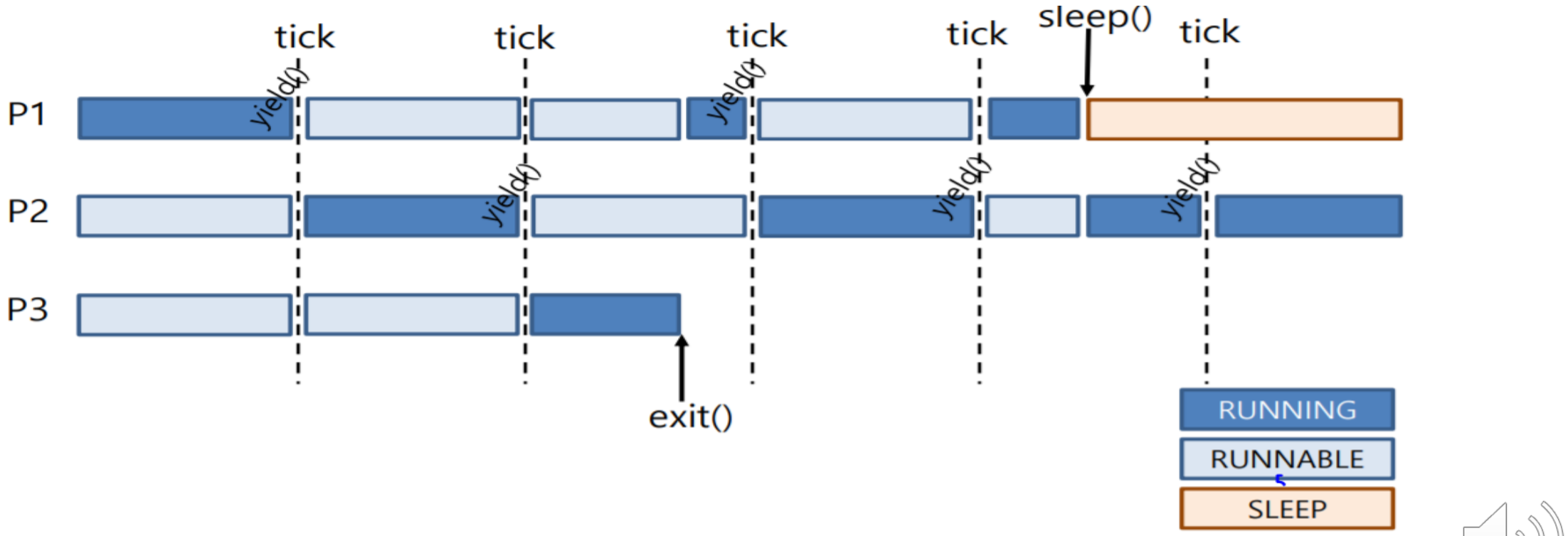
CPU scheduling

- Selects from the processes in memory that are ready to execute, and a--llocates 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



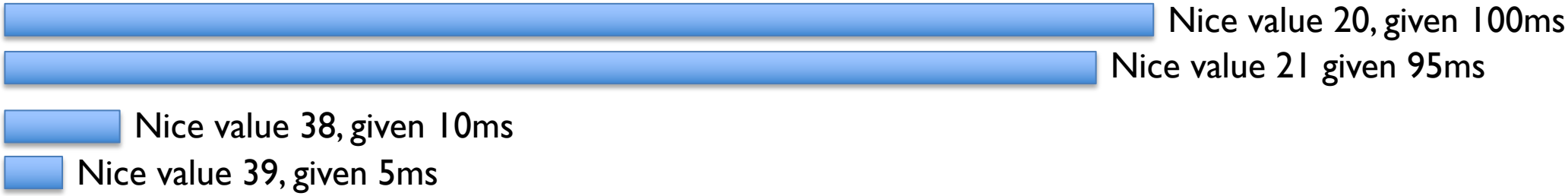
Strawman scheduler

- 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



  - The differences are same to 5ms, but it's not proportional
  - To solve this problem, CFS(Completely Fair Scheduler) had been used since Linux kernel 2.6.23. to 6.5.
  - To solve CFS's latency problem, EEVDF(Earliest Eligible Virtual Deadline First Scheduler) has been used since Linux kernel 6.6.

EEVDF (Earliest Eligible Virtual Deadline First)

- Linux default scheduler (Linux kernel 6.6~)
- Basic concept
  - Ensure both fairness and latency requirements.
  - Among Eligible Processes, Pick one with earliest virtual deadline.

  - Eligible process: process that is allowed to run
  - Lag value: difference between idle runtime and actual runtime
  - Time slice: task's minimum time to run before preemption
  - Virtual runtime: task's tracked runtime adjusted by its weight
  - Virtual deadline: the earliest time a task should finish its runtime

EEVDF (Earliest Eligible Virtual Deadline First)

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 val ues
- Therefore, a new concept "weight"
- Although there is formula, hard-code pre-defined array like Linux

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

Time-slice

- Time-slice: Task's minimum time to be executed before it is preempted
- Allocated to the process in proportion to number of CPU

$$time\_slice = scheduling\ base\ slice * (1 + \log_2(\text{ncpus}))$$

- Time period to satisfy Minimal preemption granularity for CPU-bound tasks
- Scheduling latency (0.75ms by default, can be customized in kernel)

EEVDF (Earliest Eligible Virtual Deadline First)

- Among Eligible Processes, Pick one with earliest virtual deadline
- What is "Eligible"?
  - Eligible Process: Process that is owed CPU time by the scheduler and is ready to run. Determined by lag value.
  - Eligible Time: The earliest time a process is allowed to start running
- What is "Lag Value"?
  - The difference between the time that process should have gotten and how much it actually got.
    - E.g, 3 processes A, B, C in 300ms should have gotten 100ms each, assuming same priority.
    - If process A ran for 300ms, A, B and C earned 100ms each as a fair share, and process A used 300ms total, owing CPU time to the scheduler
    - Resulting lag values of three processes A= -200ms, B = 100ms, C = 100ms.
  - If a process has positive (greater or equal to zero) lag value, it is considered "Eligible". If a process has negative lag value, it is considered "non-eligible".
- What is "Virtual deadline"?
  - The earliest time a process should ideally finish execution
  - Virtual deadline is calculated based on its weight, time slice and virtual runtime

EEVDF parameters

- 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

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

- vdeadline (virtual deadline)
  - The earliest time by which a process should have received its due CPU time.
  - Computed by adding the time remaining in its time slice to the time it became eligible.
  - Linux kernel simplifies it by using vruntime as eligible time.

$$virtual\ deadline = eligible\ time + (weighted)\ allocated\ time$$

EEVDF scheduling

1. Each task sets its virtual deadline when it becomes eligible to run.
2. The task with the earliest virtual deadline is scheduled.
3. While the task is running, virtual runtime and other scheduling parameters are updated.
4. After a task consumes its allocated execution time, its virtual deadline and eligibility are updated, and scheduler go back to 2.

Project 2. EEVDF

- In this project, you need to implement the following

1. Implement EEVDF on xv6
   - EEVDF must operate well so that runtime increases in accordance with priority
   - Vruntime, vdeadline and eligibility 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, vdeadline, eligibility and total tick

- We base our scoring on the output printed by ps()
  - Even if EEVDF is well implemented, if ps fails to properly display the values, you may not receive a score

Project 2. Implement EEVDF on xv6

- Implement EEVDF on xv6
  - Select process with earliest virtual deadline from runnable processes
  - Update runtime/vruntime/time slice for each timer interrupt
  - If task runs more than given time slice, update vdeadline and enforce a yield of the CPU
  - Default nice and latency 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 setting (our scheduling base slice is 5 ticks)
    - we do not use calculation formula for time slice setting.
    - Every process has same time slice. (5 ticks)

$$time\_slice = 5\ ticks$$

  - Timer interrupt setting
    - In xv6-riscv, timer interrupt period is defined in trap.c, clockintr() function.
    - In clockintr() function, next timer interrupt is called after 1000000 cycles, which is about a tenth of a second (0.1s).
    - You should adjust the timer interrupt period to be about a thousandth of a second. (100000 cycl

Project 2. Implement EEVDF on xv6

- Implement EEVDF on xv6
  - Virtual runtime calculation
    - Virtual runtime, actual runtime, and remaining time slice are updated as the process is running

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

  - Virtual deadline calculation
    - While process is running, virtual deadline is not updated.
    - Instead, it is updated when the process uses up the allocated time slice.
    - It should also be recalculated if its weight has been changed (calling setnice).

$$virtual\ deadline = vruntime + \text{base time slice} \times \frac{weight\ of\ nice\ 20\ (1024)}{weight\ of\ current\ process}$$

Project 2. Implement EEVDF on xv6

- Implement EEVDF on xv6
  - Lag value calculation

$$Lag_i = S - s_i = w_i * (V - v_i) = \frac{\sum((v_i - v_0) * w_i)}{\sum w_i} + v_0 - v_i$$

  - S: Ideal service time (system-wide virtual runtime)
  - S_i: Actual service time of task
  - V_i: task i's virtual runtime
  - W_i task i's weight
  - V: Global virtual time (weighted average of all vruntimes of the runqueue – running or runnable)
    - $V = v_0 + \frac{\sum(v_i - v_0) * w_i}{\sum w_i}$, where v_0 is minimum vruntime of the runqueue.
    - It is not calculated as $\frac{\sum V_i * w_i}{\sum w_i}$ since that makes the result value to easily overflow.

Project 2. Implement EEVDF on xv6

- Implement EEVDF on xv6
  - Eligibility calculation
  - What it means by being eligible?
    - Lag value is equal or greater than zero.

$$lag_i \geq 0 \rightarrow V - v_i \geq 0 \rightarrow \frac{\sum((v_i - v_0) * w_i)}{\sum w_i} + v_0 - v_i \geq 0$$

    - However, since kernel does not support floating point operations, using lag value is inaccurate.
    - So in this project, we will not use or make "lag value" of each process.
    - Instead, we will use following formular, which represents the eligibility of lag value for eligibility calculation.

$$\sum((v_i - v_0) * w_i) \geq (v_i - v_0) * \sum w_i$$

    - It is encouraged to keep track of left term, $\sum w_i$ and v_0 as variables, as the linux kernel does.

Project 2. Implement EEVDF on xv6

- How about newly forked process?
  - A process inherits the parent process's vruntime and nice value
  - It does not inherit actual runtime and remain time slice.
    Instead, every new process has actual runtime 0, and default time slice.
  - Vdeadline and eligibility should be recalculated based upon those parameters.
- How about woken process?
  - When a process is woken up, its virtual runtime and nice value remain the same before sleeping.
  - It gets default time slice, even if it has leftover time slice from before.
  - Vdeadline and eligibility should be recalculated based upon those parameters.
- How about sleeping processes?
  - Linux ensure that the negative lag value adds up to zero before it's removed from the run queue.
  - But in xv6, every parameters will remain saved, even if is a non-eiligible process.

Project 2. Implement EEVDF on xv6

- DO NOT call sched() during a wake-up of a process
  - Ensure that the time slice of the current process expires
    - Woken-up process might have the minimum vdeadline.
    - 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
- Modify Makefile so that qemu runs with a single CPU
  - In given Makefile, qemu runs with 3 CPUs, max 8 CPUs by qemu options.
  - Modify line 168 CPUS to 1.

```
ifndef CPUS
CPUS := 1
endif
```

    - This is for avoiding multi-core scheduling for your easier implementation.

Project 2. Modify *ps* System Call

- To check if EEVDF is implemented properly, `ps()` should be modified.
- Sample output (`mytest.c`)

```
$ mytest
=== TEST START ===
name    pid     state           priority        runtime/weight  runtime         vruntime        vdeadline       is_eligible     tick 2286000
init    1       sleep           20              0               0               0               5000            true
sh      2       sleep           20              0               1000            1000            5000            true
mytest  4       run             0               18              1670000         19370           19370           true
mytest  5       runble          10              18              172000          19404           19404           false
$
```

  - Print out the following information about the processes
  - Use millitick unit (multiply the tick by 1000)
    - runtime/weight, runtime, vruntime, vdeadline, eligibility, total tick
      - Do NOT use float/double types to present runtime, vruntime, and vdeadline
      - Xv6-riscv kernel does not allow floating point operations
  - 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, and eligibilities make sense for processes that are running/runnab

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 parameters are too large (exceeding the range of int)
- Page 7 to 10 is for conceptual explanation. Please refer to page 13~17 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

- How did eligibility discriminant formular derive from the lag value?
- $\frac{\sum((v_i - v_0) * w_i)}{\sum w_i} + v_0 - v_i \geq 0 \rightarrow \frac{\sum((v_i - v_0) * w_i)}{\sum w_i} \geq v_i - v_0 \rightarrow \sum((v_i - v_0) * w_i) \geq (v_0 - v_i) * \sum w_i$
- You should only consider processes that are runnable or running when calculating eligibility.

Submission

- Please implement EEVDF on xv6 and modify ps()

- Use the ***submit & check-submission*** binary file in Ye Server
  - make clean
  - $ ~swe3004/bin/submit pa2 xv6-riscv
  - 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: 10/19(Sun.), 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