Project #3: BTS(Brain Teased Scheduler)

Heejae Kim
(adpp00@snu.ac.kr)

Systems Software & Architecture Lab.

Seoul National University

2023.10.17

Reminder: Late Submission Policy

- **You can use up to *3 slip days* during this semester**

- You should explicitly declare the number of slip days you want to use on the QnA board right after each submission

- Once slip days have been used, they cannot be canceled later

- 25% of the credit will be deducted for every single day delay (if you are not using slip days for this project)

XV6 Process States

- XV6 process states (in proc.h)
  - enum procstate
    {UNUSED, USED, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };

- UNUSED: not used
- USED: initialized for new process
- SLEEPING: wait for I/O, wait() or sleep()
- RUNNABLE: ready to run
- RUNNING: now running
- ZOMBIE: exited and waiting for parent to call wait()

XV6 Scheduler

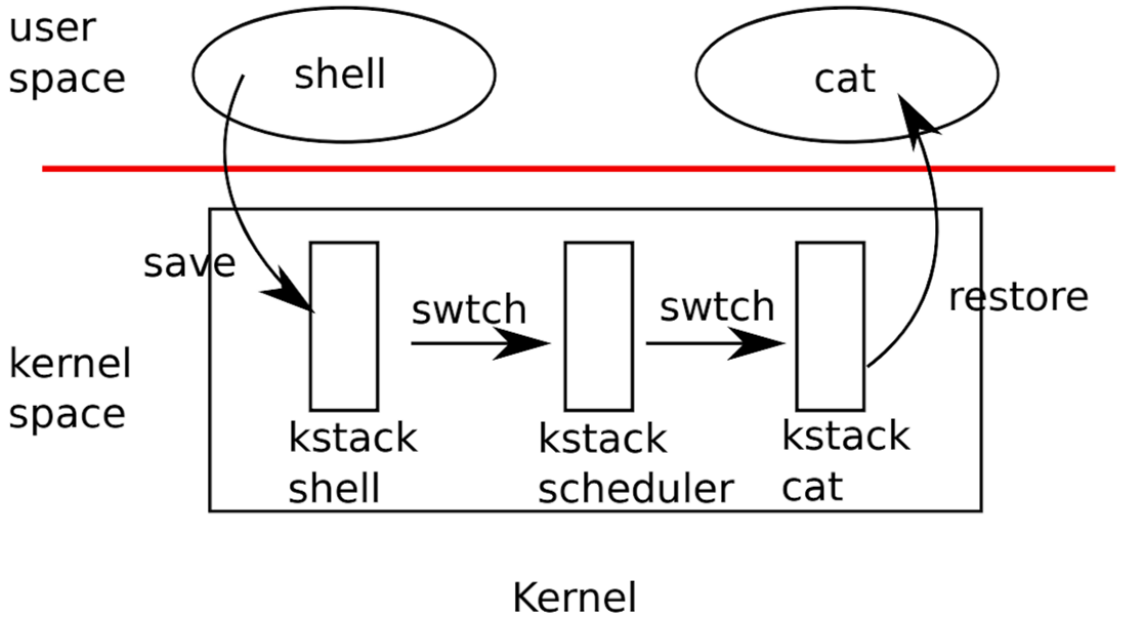
- Xv6 multiplexes by switching each CPU from one process to another

- The xv6 scheduler implements a simple scheduling policy
  - Runs each process in turn
  - This is called Round Robin

- Each CPU calls scheduler()
- Scheduler never returns.

XV6 Scheduler

- Steps involved in switching from one user process to another
  - 1. User-kernel transition to the old's process's kernel thread
  - 2. Context switch to the current CPU's scheduler thread
  - 3. Context switch to a new process's kernel thread
  - 4. Trap return to the user-level process



XV6 Code : scheduler()

- In kernel/proc.c
  - void scheduler(void)
- Scheduler loops, doing
  - 1. Choose a RUNNABLE process p to run
  - 2. Mark process p's state to RUNNING
  - 3. Set the per-CPU current process
  - 4. Context switch (start running process p)
  - 5. If process is done running, go to 1.
- Scheduler never returns

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

  c->proc = 0;
  for(;;){
    // Avoid deadlock by ensuring that devices can interrupt.
    intr_on();

    for(p = proc; p < &proc[NPROC]; p++) {
      acquire(&p->lock);
      if(p->state == RUNNABLE) {
        // Switch to chosen process.  It is the process's job
        // to release its lock and then reacquire it
        // before jumping back to us.
        p->state = RUNNING;
        c->proc = p;
        swtch(&c->context, &p->context);

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

XV6 Code : sched()

- In kernel/proc.c
  - void sched(void)
- Called from exit(), sleep(), yield()
- Context switch (return to scheduler)

```
void
sched(void)
{
  int intena;
  struct proc *p = myproc();

  if(!holding(&p->lock))
    panic("sched p->lock");
  if(mycpu()->noff != 1)
    panic("sched locks");
  if(p->state == RUNNING)
    panic("sched running");
  if(intr_get())
    panic("sched interruptible");

  intena = mycpu()->intena;
  swtch(&p->context, &mycpu()->context);
  mycpu()->intena = intena;
}
```

XV6 Code : swtch()

- In kernel/swtch.S
  - void swtch(struct context *old, struct context *new)

- Save current registers in old, load from new

```
.globl swtch
swtch:
        sd ra, 0(a0)
        sd sp, 8(a0)
        sd s0, 16(a0)
        sd s1, 24(a0)
        sd s2, 32(a0)
        sd s3, 40(a0)
        sd s4, 48(a0)
        sd s5, 56(a0)
        sd s6, 64(a0)
        sd s7, 72(a0)
        sd s8, 80(a0)
        sd s9, 88(a0)
        sd s10, 96(a0)
        sd s11, 104(a0)

        ld ra, 0(a1)
        ld sp, 8(a1)
        ld s0, 16(a1)
        ld s1, 24(a1)
        ld s2, 32(a1)
        ld s3, 40(a1)
        ld s4, 48(a1)
        ld s5, 56(a1)
        ld s6, 64(a1)
        ld s7, 72(a1)
        ld s8, 80(a1)
        ld s9, 88(a1)
        ld s10, 96(a1)
        ld s11, 104(a1)

        ret
```

Project#3: BTS(Brain Teased Scheduler)

- In this project, you have to
  - 1. Implement the nice() system call (10 points)
  - 2. Implement the BTS(Brain Teased Scheduler) algorithm (80 points)
  - 3. Submit design documents (10 points)

- Due date is 11:59(PM), October 31st (Tuesday)

1. Implement the nice() system call

- int nice(int pid, int value)
  - Sets the current nice value of the process with pid to value
  - The range of nice value is from -3 to 3 (7 levels)
  - The nice value of the init process is set to zero.
  - When a process is created, its nice value is inherited from the parent by default.
  - If pid is positive, then the nice value of the process with the specified pid is changed.
  - If pid is zero, then the nice value of the calling process is changed.

1. Implement the nice() system call

- int nice(int pid, int value) returns
  - On success, zero is returned
  - Returns -1 on error. The possible error conditions are as follows.
    - pid is negative
    - There is no valid process that has pid
    - The value is outside the range of [-3,3]

- The system call number for nice() is already assigned to 23

2. Implement the BTS algorithm

- The timeslice is just one timer tick in BTS
  - The kernel scheduler is invoked on every timer tick.

- The kernel has a predefined prio_ratio[] table
  - This table is used to compute the virtual deadline
  - prio_ratio[] table's entry corresponds to the nice level

```
/* proc.c */
#define NICE_TO_PRIO(n)    ((n) + 3)

int prio_ratio[] = {1, 2, 3, 5, 7, 9, 11};
```

2. Implement the BTS algorithm

- When a process is put into the runqueue, modify its virtual deadline
  - Process P's Virtual deadline = current tick + prio_ratio[NICE_TO_PRIO(P's nice value)]

- The scheduler determines the next process to execute among the RUNNABLE processes based on the following priorities:
  - 1. Process with the minimum virtual deadline
  - 2. The last process (which runs immediately before)
  - 3. Process with the lower nice value
  - 4. Process with the lower pid

2. Implement the BTS algorithm

- The running process is NOT preempted until the end of its timeslice (even if a process with a lower virtual deadline is created or wakes up)
- When the current process is blocked, its virtual deadline is not changed
- When the process is awakened and resumes execution, the virtual deadline remains unadjusted
- When the nice value of a RUNNABLE process is changed, its virtual deadline is immediately changed
- If there are no RUNNABLE processes, the behavior of the scheduler is the same as that of the current round-robin scheduler

3. Design document

- In this project, you need to submit a report explaining your implementation (in a single PDF file)
- These must be included in your report
  - Brief summary of modifications you have made
  - The details about your implementation of
    - The nice() system call
    - The BTS algorithm
  - The results of running schedtest3 with the analysis of your result
    - xv6.log file ($make qemu-log)
    - graph.png file ($make png)

Skeleton Code

- You should work on the pa3 branch as follows:

```
$ git clone https://github.com/snu-csl/xv6-riscv-snu
$ cd xv6-riscv-snu
$ git checkout pa3
```

- Then, you have to set your STUDENTID in the Makefile

- Also, you should install python numpy, pandas, and matplotlib packages.

```
$ sudo apt install python3-numpy python3-pandas python3-matplotlib
```

Skeleton Code

- In this project, getticks() system call is included in your skeleton code
- int getticks(int pid)
  - getticks() returns the number of ticks used by the process pid
  - If pid is positive, return the number of ticks used by process pid is returned
  - If pid is zero, return the number of ticks used by the calling process

- Two new fields, nice and ticks, are added in the proc structure
  - In kernel/proc.h, struct proc

Restrictions

- The number of CPUs is already set to 1 in the Makefile.
- Your implementation should pass the following test programs (These test programs are already available on xv6)
  - schedtest1
  - schedtest2
  - schedtest3
- Do not add any system calls other than nice() and getticks()
- You only need to modify those files in the ./kernel directory
  - Changes to other source code will be ignored during grading.
- Please remove all the debugging outputs before you submit

Tips

- You may read & modify
  - kernel/proc.c
  - kernel/proc.h
  - kernel/sysproc.c
  - and other files if necessary

- Read xv6 book
  - Chapter 7 to understand the scheduling subsystem of xv6

Submission

- Perform the make submit command to generate a compressed tar file
- Upload this tar file + report to the submission server
- The total number of submissions will be limited to 30
- Only the version marked FINAL will be considered
- It takes a long time to grading, so please wait for a few minutes

Example 1: Same nice value

- Two processes P0 and P1 have same nice value, -2
- Prio_ratio of P0 and P1 will be prio_ratio[-2 + 3] = 2
- P0 has a lower pid than P1

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	2											
Virtual Deadline P1	2											
Scheduled Process												

Example 1: Same nice value

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	2											
Virtual Deadline P1	2											
Scheduled Process	P0											

Example 1: Same nice value

- Update the virtual deadline of P0
- Virtual deadline of P0: current tick (1) + prio_ratio(2) = 3

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	2	3										
Virtual Deadline P1	2	2										
Scheduled Process	P0											

Example 1: Same nice value

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	2	3										
Virtual Deadline P1	2	2										
Scheduled Process	P0	P1										

Example 1: Same nice value

- Update the virtual deadline of P1
- Virtual deadline of P1: current tick (2) + prio_ratio(2) = 4

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	2	3	3									
Virtual Deadline P1	2	2	4									
Scheduled Process	P0	P1										

Example 1: Same nice value

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	2	3	3									
Virtual Deadline P1	2	2	4									
Scheduled Process	P0	P1	P0									

Example 1: Same nice value

- Update the virtual deadline of P0
- Virtual deadline of P0: current tick (3) + prio_ratio(2) = 5

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	2	3	3	5								
Virtual Deadline P1	2	2	4	4								
Scheduled Process	P0	P1	P0									

Example 1: Same nice value

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	2	3	3	5								
Virtual Deadline P1	2	2	4	4								
Scheduled Process	P0	P1	P0	P1								

Example 1: Same nice value

- Update the virtual deadline of P1
- Virtual deadline of P1: current tick (4) + prio_ratio(2) = 6

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	2	3	3	5	5							
Virtual Deadline P1	2	2	4	4	6							
Scheduled Process	P0	P1	P0	P1								

Example 1: Same nice value

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	2	3	3	5	5							
Virtual Deadline P1	2	2	4	4	6							
Scheduled Process	P0	P1	P0	P1	P0							

Example 1: Same nice value

- And so on…

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	2	3	3	5	5	7	7	9	9	11	11	13
Virtual Deadline P1	2	2	4	4	6	6	8	8	10	10	12	12
Scheduled Process	P0	P1	P0	P1	P0	P1	P0	P1	P0	P1	P0	P1

Example 2. Three Processes

- Process P0, P1, P2 have the nice values of -3, -2, and 0 respectively
- Prio_ratio of P0, P1, P2 will be 1, 2, 5

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1											
Virtual Deadline P1	2											
Virtual Deadline P2	5											
Scheduled Process												

Example 2. Three Processes

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1											
Virtual Deadline P1	2											
Virtual Deadline P2	5											
Scheduled Process	P0											

Example 2. Three Processes

- Update the virtual deadline of P0
- Virtual deadline of P0: current tick (1) + prio_ratio(1) = 2

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2										
Virtual Deadline P1	2	2										
Virtual Deadline P2	5	5										
Scheduled Process	P0											

Example 2. Three Processes

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2										
Virtual Deadline P1	2	2										
Virtual Deadline P2	5	5										
Scheduled Process	P0	P0										

Example 2. Three Processes

- Update the virtual deadline of P0
- Virtual deadline of P0: current tick (2) + prio_ratio(1) = 3

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3									
Virtual Deadline P1	2	2	2									
Virtual Deadline P2	5	5	5									
Scheduled Process	P0	P0										

Example 2. Three Processes

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3									
Virtual Deadline P1	2	2	2									
Virtual Deadline P2	5	5	5									
Scheduled Process	P0	P0	P1									

Example 2. Three Processes

- Update the virtual deadline of P0
- Virtual deadline of P0: current tick (3) + prio_ratio(2) =5

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3								
Virtual Deadline P1	2	2	2	5								
Virtual Deadline P2	5	5	5	5								
Scheduled Process	P0	P0	P1									

Example 2. Three Processes

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3								
Virtual Deadline P1	2	2	2	5								
Virtual Deadline P2	5	5	5	5								
Scheduled Process	P0	P0	P1	P0								

Example 2. Three Processes

- Update the virtual deadline of P0
- Virtual deadline of P0: current tick (4) + prio_ratio(1) = 5

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5							
Virtual Deadline P1	2	2	2	5	5							
Virtual Deadline P2	5	5	5	5	5							
Scheduled Process	P0	P0	P1	P0								

Example 2. Three Processes

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5							
Virtual Deadline P1	2	2	2	5	5							
Virtual Deadline P2	5	5	5	5	5							
Scheduled Process	P0	P0	P1	P0	P0							

Example 2. Three Processes

- Update the virtual deadline of P0
- Virtual deadline of P0: current tick (5) + prio_ratio(1) = 6

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6						
Virtual Deadline P1	2	2	2	5	5	5						
Virtual Deadline P2	5	5	5	5	5	5						
Scheduled Process	P0	P0	P1	P0	P0							

Example 2. Three Processes

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6						
Virtual Deadline P1	2	2	2	5	5	5						
Virtual Deadline P2	5	5	5	5	5	5						
Scheduled Process	P0	P0	P1	P0	P0	P1						

Example 2. Three Processes

- Update the virtual deadline of P1
- Virtual deadline of P1: current tick (6) + prio_ratio(2) = 8

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6	6					
Virtual Deadline P1	2	2	2	5	5	5	8					
Virtual Deadline P2	5	5	5	5	5	5	5					
Scheduled Process	P0	P0	P1	P0	P0	P1						

Example 2. Three Processes

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6	6					
Virtual Deadline P1	2	2	2	5	5	5	8					
Virtual Deadline P2	5	5	5	5	5	5	5					
Scheduled Process	P0	P0	P1	P0	P0	P1	P2					

Example 2. Three Processes

- Update the virtual deadline of P2
- Virtual deadline of P2: current tick (7) + prio_ratio(5) = 11

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6	6	6				
Virtual Deadline P1	2	2	2	5	5	5	8	8				
Virtual Deadline P2	5	5	5	5	5	5	5	12				
Scheduled Process	P0	P0	P1	P0	P0	P1	P2					

Example 2. Three Processes

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6	6	6				
Virtual Deadline P1	2	2	2	5	5	5	8	8				
Virtual Deadline P2	5	5	5	5	5	5	5	12				
Scheduled Process	P0	P0	P1	P0	P0	P1	P2	P0				

Example 2. Three Processes

- Update the virtual deadline of P0
- Virtual deadline of P0: current tick (8) + prio_ratio(1) = 9

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6	6	6	9			
Virtual Deadline P1	2	2	2	5	5	5	8	8	8			
Virtual Deadline P2	5	5	5	5	5	5	5	12	12			
Scheduled Process	P0	P0	P1	P0	P0	P1	P2	P0				

Example 2. Three Processes

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6	6	6	9			
Virtual Deadline P1	2	2	2	5	5	5	8	8	8			
Virtual Deadline P2	5	5	5	5	5	5	5	12	12			
Scheduled Process	P0	P0	P1	P0	P0	P1	P2	P0	P1			

Example 2. Three Processes

- Update the virtual deadline of P1
- Virtual deadline of P1: current tick (9) + prio_ratio(2) = 11

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6	6	6	9	9		
Virtual Deadline P1	2	2	2	5	5	5	8	8	8	11		
Virtual Deadline P2	5	5	5	5	5	5	5	12	12	12		
Scheduled Process	P0	P0	P1	P0	P0	P1	P2	P0	P1			

Example 2. Three Processes

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6	6	6	9	9		
Virtual Deadline P1	2	2	2	5	5	5	8	8	8	11		
Virtual Deadline P2	5	5	5	5	5	5	5	12	12	12		
Scheduled Process	P0	P0	P1	P0	P0	P1	P2	P0	P1	P0		

Example 2. Three Processes

- Update the virtual deadline of P0
- Virtual deadline of P0: current tick (10) + prio_ratio(1) = 11

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6	6	6	9	9	11	
Virtual Deadline P1	2	2	2	5	5	5	8	8	8	11	11	
Virtual Deadline P2	5	5	5	5	5	5	5	12	12	12	12	
Scheduled Process	P0	P0	P1	P0	P0	P1	P2	P0	P1	P0		

Example 2. Three Processes

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6	6	6	9	9	11	
Virtual Deadline P1	2	2	2	5	5	5	8	8	8	11	11	
Virtual Deadline P2	5	5	5	5	5	5	5	12	12	12	12	
Scheduled Process	P0	P0	P1	P0	P0	P1	P2	P0	P1	P0	P0	

Example 2. Three Processes

- Update the virtual deadline of P0
- Virtual deadline of P0: current tick (11) + prio_ratio(1) = 12

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6	6	6	9	9	11	12
Virtual Deadline P1	2	2	2	5	5	5	8	8	8	11	11	11
Virtual Deadline P2	5	5	5	5	5	5	5	12	12	12	12	12
Scheduled Process	P0	P0	P1	P0	P0	P1	P2	P0	P1	P0	P0	

Example 2. Three Processes

- 1. Process with the minimum virtual deadline
- 2. The last process (which runs immediately before)
- 3. Process with the lower nice value
- 4. Process with the lower pid

tick	0	1	2	3	4	5	6	7	8	9	10	11
Virtual Deadline P0	1	2	3	3	5	6	6	6	9	9	11	12
Virtual Deadline P1	2	2	2	5	5	5	8	8	8	11	11	11
Virtual Deadline P2	5	5	5	5	5	5	5	12	12	12	12	12
Scheduled Process	P0	P0	P1	P0	P0	P1	P2	P0	P1	P0	P0	P1

Example 3. schedtest3

- In user/schedtest3.c

- Creates three CPU-intensive child processes
- Measures the number of ticks used by those processes every 60 ticks
- Their nice values changing every 300 ticks

```c
void loginit()
{
  p1 = getticks(pid1);
  p2 = getticks(pid2);
  p3 = getticks(pid3);
  printf("%d, %d, %d, %d\n", sec, p1, p2, p3);
}

void log()
{
  int i;

  for (i = 0; i < R; i++)
  {
    sleep(P * 10);
    sec += P;
    t1 = getticks(pid1);
    t2 = getticks(pid2);
    t3 = getticks(pid3);
    printf("%d, %d, %d, %d\n", sec, t1 - p1, t2 - p2, t3 - p3);
    p1 = t1;
    p2 = t2;
    p3 = t3;
  }
}

void
main(int argc, char *argv[])
{

  if ((pid1 = fork()) == 0)
    while (1);

  if ((pid2 = fork()) == 0)
    while (1);

  if ((pid3 = fork()) == 0)
    while (1);

  loginit();

  // Phase 1: 0/0/0
  log();

  // Phase 2: -3/0/3
  nice(pid1, -3);
  nice(pid2, 0);
  nice(pid3, 3);
  log();

  // Phase 3: -2/0/2
  nice(pid1, -2);
  nice(pid2, 0);
  nice(pid3, 2);
  log();
```

Example 3. schedtest3

- Tick 0: schedtest forks 3 times, calls loginit(), and calls sleep(60)
  - nice value of P1, P2, P3 is 0,0,0, respectively

tick	0	0	0	0	0	0	1	2	3	4	5
schedtest	fork()	fork()	fork()	loginit()	sleep(60)	sleeping…					
VD P1	5	5	5	5	5						
VD P2		5	5	5	5						
VD P3			5	5	5						
Scheduled Process	sched test	sched test	sched test	sched test	sched test						

Example 3. schedtest3

- After schedtest calls sleep(60), P1, P2, P3 will be scheduled.

Tick	0	0	0	0	0	0	1	2	3	4	5
schedtest	fork()	fork()	fork()	loginit ()	sleep (60)	sleeping…					
VD P1	5	5	5	5	5	5	6	6	6	9	9
VD P2		5	5	5	5	5	5	7	7	7	10
VD P3			5	5	5	5	5	5	8	8	8
Scheduled Process	sched test	sched test	sched test	sched test	sched test	P1	P2	P3	P1	P2	P3

Example 3. schedtest3

- Tick 60: schedtest wakes up, prints log, and sleep(60) again…
  - nice value of P1, P2, P3 is 0,0,0, respectively

tick	60	60	60								
schedtest	print log	sleep (60)	sleeping…								
VD P1	63	63									
VD P2	64	64									
VD P3	65	65									
Scheduled Process	sched test	sched test									

Example 3. schedtest3

- And then…

tick	60	60	60	61	62	63	64	65	66	67	68
schedtest	print log	sleep (60)	sleeping…								
VD P1	63	63	63	66	66	66	69	69	69	72	72
VD P2	64	64	64	64	67	67	67	70	70	70	73
VD P3	65	65	65	65	65	68	68	68	71	71	71
Scheduled Process	sched test	sched test	P1	P2	P3	P1	P2	P3	P1	P2	P3

Example 3. schedtest3

- Tick 300: Schedtest wakeup, prints log, change nice value and sleep…
  - nice value of P1, P2, P3 changes to -3,0,3, respectively

tick	299	300	300	300	300	300	300				
schedtest	sleeping…	print log	nice (pid1)	nice (pid2)	nice (pid3)	sleep (60)	sleeping…				
VD P1	303	303	301	301	301	301					
VD P2	304	304	304	305	305	305					
VD P3	302	305	305	305	311	311					
Scheduled Process	P3	sched test	sched test	sched test	sched test	sched test					

Example 3. schedtest3

- And then…

tick	299	300	300	300	300	300	300	301	302	303	304
schedtest	sleeping…	print log	nice (pid1)	nice (pid2)	nice (pid3)	sleep (60)	sleeping...				
VD P1	303	303	301	301	301	301	301	302	303	304	305
VD P2	304	304	304	305	305	305	305	305	305	305	305
VD P3	302	305	305	305	311	311	311	311	311	311	311
Scheduled Process	P3	sched test	sched test	sched test	sched test	sched test	P1	P1	P1	P1	P1

Example 3. schedtest3

- If you successfully implement BTS,
  you should be able to get a result like the following:

```
xv6 kernel is booting

init: starting sh
$ schedtest3
0, 0, 0, 0
6, 20, 20, 20
12, 20, 20, 20
18, 20, 20, 20
24, 20, 20, 20
30, 20, 20, 20
36, 46, 10, 4
42, 45, 10, 5
48, 45, 10, 5
54, 45, 10, 5
60, 45, 10, 5
66, 43, 11, 6
```

```
72, 42, 12, 6
78, 42, 12, 6
84, 42, 12, 6
90, 42, 12, 6
96, 37, 14, 9
102, 37, 14, 9
108, 38, 13, 9
114, 36, 14, 10
120, 37, 14, 9
126, 20, 20, 20
132, 20, 20, 20
138, 20, 20, 20
144, 20, 20, 20
150, 20, 20, 20
$ QEMU: Terminated
```

Example 3. schedtest3

- In order to generate a graph, you should run xv6 using the make qemu-log command that saves output into the file named xv6.log
- And then run the make png command to generate the graph.png file

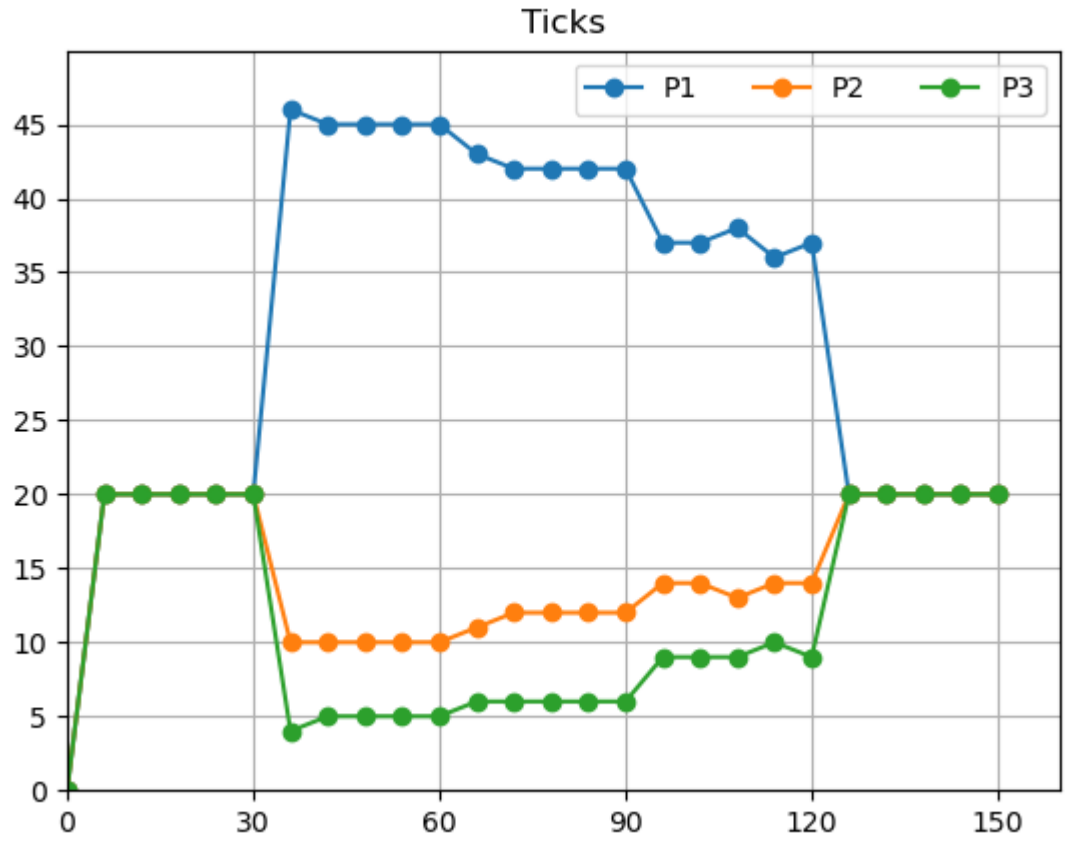
```
~/pa3/xv6-riscv-snu ❯ make qemu-log                                                    root@Heejae-Desktop 01:52:14 PM
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 1 -nographic -global virtio-mmio.force-legacy=false -drive f
ile=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 | tee xv6.log

xv6 kernel is booting

init: starting sh
$ schedtest3
0, 0, 0, 0
6, 20, 20, 20
12, 20, 20, 20
18, 20, 20, 20
24, 20, 20, 20
30, 20, 20, 20
36, 46, 10, 4
```

...

```
144, 20, 20, 20
150, 20, 20, 20
$ QEMU: Terminated
*** The output of xv6 is logged in the 'xv6.log' file.
~/pa3/xv6-riscv-snu ❯ make png                                          8m 17s root@Heejae-Desktop 02:00:36 PM
./graph.py xv6.log graph.png
graph saved in the 'graph.png' file
```

Example 3. schedtest3

- If everything goes fine, you will get the following graph:



Thank you!

- Any questions?