COL331: Assignment 2

Harshit Mawandia 2020CS10348
Dhruv Tyagi 2020CS10341

April 5, 2023

1 Real Time Scheduling

In real-time systems, tasks must be completed within strict time limits, known as deadlines. To enable the execution of real-time jobs in xv6, we added the following real-time scheduling policies: the earliest deadline first (EDF) and rate monotonic (RM) scheduling algorithms. The default scheduler in xv6 is a $round-robin\ scheduler$ which treats each task with the same priority and gives a certain amount of time to each of them. The scheduler code for xv6 may be found in `proc.c` and its accompanying header file, `proc.h`.

1.1 Implementations

In `xv6` after every 10ms there is an hardware interrupt due to the TIMER as defined in the `traps.c` file. After every tick of the timer we call the scheduler to schedule the process with the highest priority at that point. We also check which processes have `p->killed=1` or `p->elapsed_time >= p->exec_time` and use the `exit()` system call on them to kill them.

We run the `scheduler` function to first check whether any process with EDF policy is `RUNNABLE`, if yes, then we find the process woth EDF policy with the earliest deadline. If no such process is found, we check if there exists a `RUNNABLE` process with RMS scheduling policy. If yes, then we find the one with the lowest weight. If not, we all other `RUNNABLE` processes with the round robin scheduling policy.

1.1.1 EDF

In the **Earliest Deadline First** scheduling policy, we go through all the processes and choose the `RUNNABLE` process `p` with the lowest `p->deadline + p->arrival_time`. We break ties with the lowest `p->pid`.

1.1.2 RM

In the **Rate Mean** Scheduling Policy, we choose the process which has the lowest weight amongst all processes with `sched_policy = 1`(Rate Mean Scheduling), and we break ties by choosing the process with a lower `pid`. To calculate weight w we use the given formula:

$$w = \frac{(30 - r) * 3 + 28}{29}$$

where r = `p->rate`.

After finding the process with the highest priority (according to either of the scheduling policies):

1. Assign the process to the CPU.
2. Switch to the user space
3. Set the state of the process from `RUNNABLE` to `RUNNING`
4. Increment the elapsed time counter.
5. Switch the context to the chosen process. Save the current registers on the stack, creating a struct context, and save its address in `*old`.

6. Switch stacks to new and pop previously-saved registers.
7. Switch back to the kernel context after the process is done running
8. Set the process assigned to the CPU back to 0
9. Reiterate by looking for the next executable process.

1.2 System Specification:

- We use only 1 core, by specifying this in the `makefile` `CPUS` variable.
- We assume all user-tasks to be pre-emptable, there is no precedence ordering and tasks do not share resources
- We are also assuming that all tasks would be either EDF or RM
- By default all processes follow round robin scheduling until user specifies there scheduling policy by using `sys_sched_policy` system call

All the system calls have been declared in the same manner as in assignment 1. To add the system calls to the OS, we add them in *usys.S* as `SYSCALL(<name of syscall>))` , also add it in *user.h* as `int <name of syscall>()` in *syscall.h* as `#define SYS_<name of syscall> <syscall number>` and also add it to the `syscalls[]` in *syscall.c*.

1.3 `sys_sched_policy`

In this system call, we set the scheduling policy (`process->sched_policy`) for a process as specified by the user to either RMS(`sched_policy=1`) or EDF(`sched_policy=0`). We also check whether the process is schedulable with the co existing process, and `return 0` on success or `-22` if the process is not schedulable.

The checks used to test schedulability for both EDF and RMS are given below:

1.3.1 EDF Schedulability Check

To check whether a process is **EDF schedulable**:

1. Check whether the new process `p` has codesched_policy=0
2. Iterate through all the process and check which processes have EDF policy
3. For the processes with EDF policy, calculate utilisation using formula:

$$U = \sum_{\text{policy=EDF}} \frac{\text{exec\_time}_i}{\text{deadline}_i}$$

4. if `U<1` the processs `p` is schedulable, return 0
5. We also initialise `p->elapsed_time = 0`
6. if not, set `p->killed = 1` and `p->schedulable=0`

For processes that are `RUNNING` or `RUNNABLE`

1.3.2 RMS Schedulability Check

To check whether a process is **RMS schedulable**

1. Check whether the new process `p` has codesched_policy=1
2. Iterate through all the process and check which processes have RMS policy

3. For the processes with RMS policy, calculate utilisation using formula:

$$U = \sum_{\text{policy=RMS}} \frac{\text{exec\_time}_i}{\text{period}_i}$$

As we are given rate (which is the inverse of the period) instead:

$$U = \sum_{\text{policy=RMS}} \text{exec\_time}_i * \text{rate}_i$$

Note that in the actual code, we also divide U calculated above by 100 to match the units.

4. if $U < n(2^{1/n} - 1)$ the processs `p` is schedulable, return 0
5. We also initialise `p->elapsed_time = 0`
6. if not, set `p->killed = 1` and `p->schedulable=0`

1.4 sys_exec_time

We implement a new system call `sys_exec_time` by which we set the `process->exec_time` for the `pid` as specified by the user, by iteration through the `ptable` and finding the process with the matching pid.

We use locks on the `ptable` as necessary before accessing its contents using `acquire(&ptable.lock)` and `release(&ptable.lock)` wherever necessary.

1.5 sys_deadline

We implement a new system call `sys_deadline` by which we set the `process->deadline` for the `pid` as specified by the user, by iteration through the `ptable` and finding the process with the matching pid.

We use locks on the `ptable` as necessary before accessing its contents using `acquire(&ptable.lock)` and `release(&ptable.lock)` wherever necessary.

1.6 sys_rate

We implement a new system call `sys_rate` by which we set the `process->rate` for the `pid` as specified by the user, by iteration through the `ptable` and finding the process with the matching pid.

We use locks on the `ptable` as necessary before accessing its contents using `acquire(&ptable.lock)` and `release(&ptable.lock)` wherever necessary.

2 Report

In this report, we have specified

- The implementation methodology for both EDF and RM scheduling policies.
- All the 4 system calls.
- Schedulability Checks for the EDF and RM scheduling policies when the `sys_sched_policy` is system call is executed.

In this report we have specified in detail the approach to each part of the assignment with steps/pseudo- code wherever necessary.

3 Submission Instructions

We have submitted the compressed copy of the modified xv6-public folder which contains all the code files and the report to the moodle.

The name of the zip folder is `assignment1_easy_2020CS10348_2020CS10341.tar.gz`