Assignment 5
Enhancing xv6 OS
Computer Systems Engineering-1
Monsoon 2019

Deadline: 6th November, 11:55 PM. There will be NO deadline extension.

Xv6 is a simplified operating system developed at MIT. You will be tweaking the Xv6 operating system as a part of this assignment.

Task 1

Waitx Sys Call

As a part of this task, you're expected to extend the current proc structure and add new fields ctime, etime and rtime for creation time, end-time and total time respectively of a process. When a new process gets created the kernel code should update the process creation time. The run-time should get updated after every clock tick for the process. To extract this information from the kernel add a new system call which extends wait. The new call will be:

int waitx(int* wtime, int* rtime)

The two arguments are pointers to integers to which waitx will assign the total number of clock ticks during which process was waiting and total number of clock ticks when the process was running. The return values for waitx should be same as that of wait system-call. Create a test program which utilizes the waitx system call by creating a ‘time’ like command for the same.
Note: This can be used to implement your scheduler functions.

Getpinfo Sys Call

You'll need to create a new system call for this part: **int getpinfo(struct proc_stat *)**. This routine returns some basic information about each process: its process ID, total run time, how many times it has been chosen to run, which queue it is currently on 0, 1, 2, 3 or 4 (check Task 2 part c) and ticks received in each queue. To do this, you will need to create the **proc_stat** struct :-

```
struct proc_stat {
    int pid;   // PID of each process
    float runtime;  // Use suitable unit of time
    int num_run; // number of time the process is executed
    int current_queue; // current assigned queue
    int ticks[5]; // number of ticks each process has received at each of the 5  priority queue
};
```

Task 2

The default scheduler of xv6 is a round-robin based scheduler. In this task, you'll implement 3 other scheduling policies and incorporate them in Xv6.

(a) First come - First Served (FCFS)

Implement a non preemptive policy that selects the process with the lowest creation time. The process runs until it no longer needs CPU time.

(b) Priority Based Scheduler

A priority-based scheduler selects the process with the highest priority for execution. In case two or more processes have the same priority, we choose them in a round-robin fashion. The priority of a process can be in the range [0,100], the smaller value will represent higher priority. Set the default priority of a process as 60. To change the default priority add a new system call **set_priority** which can change the priority of a process.

int set_priority(int)

The system-call returns the old-priority value of the process. In case the priority of the process increases (the value is lower than before), then rescheduling should be done.

(c) Multi-level Feedback queue scheduling

MLFQ scheduler allows processes to move between different priority queues based on their behavior and CPU bursts. If a process uses too much CPU time, it is pushed to a lower priority queue, leaving I/O bound and interactive processes for higher priority queues. Also, to prevent starvation, it implements aging.

Keeping all these benefits in mind, implement a simplified multi-level feedback queue scheduler.

Scheduler Details :-

1. Create five priority queues, with the highest priority being number as 0 and bottom queue with the lowest priority as 4.
2. Assign a suitable value for 1 tick of CPU timer.

3. The time-slice for priority 0 should be 1 timer tick. The times-slice for priority 1 is 2 timer ticks; for priority 2, it is 4 timer ticks; for priority 3, it is 8 timer ticks; for priority 4, it is 16 timer ticks.

Procedure :-

1. On the initiation of a process, push it to the end of the highest priority queue.
2. The highest priority queue should be run always, if not empty.
3. If the process completes, it leaves the system.
4. If the process used the complete time slice assigned for its current priority queue, it is pre-empted and inserted at the end of the next lower level queue.
5. If a process voluntarily relinquishes control of the CPU, it leaves the queuing network, and when the process becomes ready again after the I/O, it is inserted at the tail of the same queue, from which it is relinquished earlier (Explain in the report how could this be exploited by a process ?).
6. A round-robin scheduler should be used for processes at the lowest priority queue.
7. To prevent starvation, implement the aging phenomenon using the structure defined in Task - 1 :-
    a. If the wait time of a process in lower priority queues exceeds a given limit(assign a suitable limit to prevent starvation), their priority is increased and they are pushed to the next higher priority queue.
    b. The wait time is defined as the difference between the current time and the time at which the process was last executed.

Modify the Makefile to support SCHEDULER - a macro for compilation of the specified scheduling algorithm. Use the flags for compilation :-

- First Come First Serve = FCFS
- Priority Based = PBS
- Multilevel Feedback Queue = MLFQ

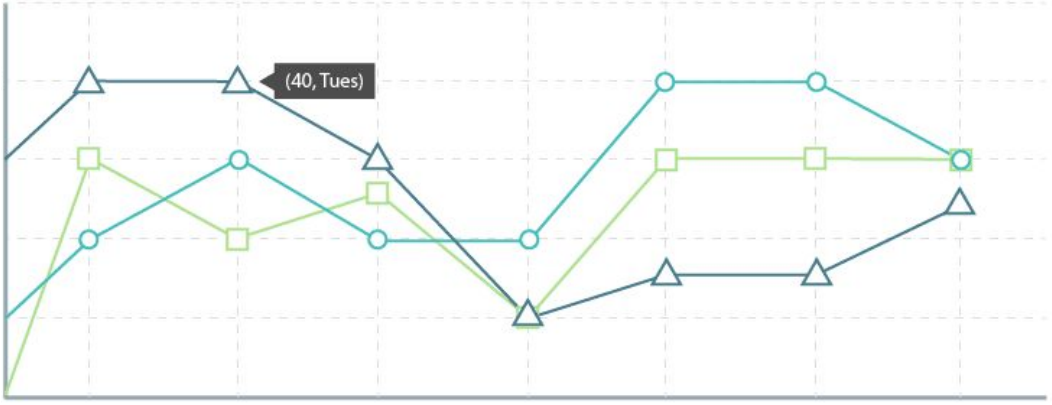
Example :- make qemu SCHEDULER=MLFQ

Write a sample benchmark program (make sure it runs for at least 20 secs on your laptop) which can be used to compare the performances of the scheduling algorithms and demonstrate using that program during the evals. Also, submit a report covering performance comparison between the default and 3 implemented policies.

Bonus :-

Plot timeline graphs for processes running with MLFQ Scheduler from the output received from the getpinfo() sys call. Use the benchmark/workload from Task 2 to vary how long each process uses the CPU before relinquishing voluntarily (Hint: use sleep()).

The graph should be a timeline/scatter plot between queue_id(y-axis) and time elapsed(x-axis) from start with color-coded processes. Add to the report the observations recorded for different types of processes. Example :-



Note :- Plotting of the graph can be done in the language of your choice.

Guidelines

1. Submission format: RollNo_Assignment5.tar.gz. Wrong submission formats will be penalized.
2. Submission by email to TAs will not be accepted.
3. Any copy cases found will lead to serious consequences.
4. Make sure you write a readme which briefly describes the implementation.
5. Whenever you add new files do not forget to add them to the Makefile so that they get included in the build.
6. The xv6 OS base code can be downloaded from https://github.com/mit-pdos/xv6-public