Report

Jainit Bafna

2021114003

FCFS Scheduler

- For this scheduler, we iterate over the proc table, and select the process with the minimum `ctime`.
- This value of `ctime` is set equal to number of ticks at the time of process creation, and thus, this is the time of creation of the process.
- I acquire locks for every process that I check.
- Also, I modified the file `kernel/trap.c` to disable processes being yeilded due to clock interrupts. Changes were made in the `kerneltrap` and the `usertrap` function.

MLFQ Implementation

Implement a simplified preemptive MLFQ scheduler that 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 in the higher priority queues.
- To prevent starvation, implement aging.

Details:

1. Create four priority queues, giving the highest priority to queue number 0 and lowest priority to queue number 3
2. The time-slice are as follows:
    1. For priority 0: 1 timer tick
    2. For priority 1: 3 timer ticks
    3. For priority 2: 9 timer ticks
    4. For priority 3: 15 timer ticks

    NOTE: Here tick refers to the clock interrupt timer. (see kernel/trap.c)

How ?

- For this scheduler, I initialized 4 arrays(queues) numbered 0 to 3, with 0 having the highest priority.
- During the initialization of XV-6, I set all the queues to empty.
- After that, every time during fork and userinit, I insert the process into the **0th queue**. For convenience, I created functions to insert a process into the queue and to delete the ith process from the queue.

- Further, at the time of inserting into the 0th queue through fork, I check if the current process being executed(through `myproc()`) is in a lower priority queue(1-3), then i yield() and the currently running process gets preempted
- Now, at the start of the scheduler, I check for ageing in the queues.
  - **For ageing here, as well as for the wtime printed in MLFQ's procdump, I consider the queue wait time as the number of ticks for which the process in that queue was in the RUNNABLE state. This value is what is used for implementing ageing.**
  - The maximum age that is allowed is defined in the AGE_MAX parameter.
  - If the wait time of the process has gone beyond the AGE_MAX value, then it gets inserted into the higher priority queue(except if it is in 0th queue).
- After this, I find the first process in a non-empty highest prioirity queue, and if it is runnable, select it for running. Now, we set its state to running and give it CPU to run.
- In order to implement time slicing, I made modifications in `kernel/trap.c` and modified such that if the current queue run time exceeds the ticks limit, then the process gets yielded.
  - Also, in this case, I set a flag `q_leap` to 1.
- Now, after the process has finished running, I check for the `q_leap` flag, in which case I increase its queue number. Now, if the process is still runnable, then I insert it back into the queue according to its current queue_number.
- Further, for processes that themselves relinquish control(for I/O, etc.), they go into the sleep state, and they are not **pushed back into the queue**. However, when the I/O stops, they reach the `wakeup` function, where they are added back into the queue they were previously in before going to sleep.
- Further, after killing a process, I check if the process was in the RUNNABLE state, and if so, I remove it from the queue it was in.

The following table shows the average response time and average waiting time for the scheduler test cases.

Algorithm	Number of CPUs	Average Response Time (rtime)	Average Waiting Time (wtime)
MLFQ	1	14	152
RR	3	18	118
RR	1	14	158
FCFS	1	14	129
FCFS	3	22	108

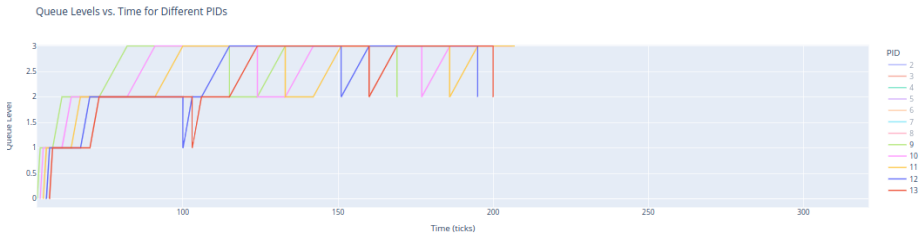

Figure 1: Graph with aging 30