NC State University

Department of Electrical and Computer Engineering

ECE 565: Fall 2023

Project 2

by

RAGHUL SRINIVASAN
UnityID: rsriniv7
(200483357)

Questions

Q1. What is the ready list? List all the system calls that operate on it.

The processes that are ready in the PR_READY state (ready to be scheduled by the scheduler) are present in the ready list queue in descending order of priority.
Ready list is operated on in the following system calls

- Sysinit()
- Ready()
- Resched() -> used to compare priorities of the current running process with the processes in a ready list and the process with the highest priority is scheduled

Q2. What is the default process scheduling policy used in Xinu? Can this policy lead to process starvation? Explain your answer.

XINU follows round-robin priority-based scheduling. This policy can lead to starvation when high-priority processes keep scheduling (PR_CURR state) and don't allow lower-priority processes to be executed.

Q3. When a context switch happens, how does the resched function decide if the currently executing process should be put back into the ready list?

Resched() compares the priority of the currently running process (in PR_CURR state) with the priority of the process at the top of the ready list queue. If the priority of the process at the top of the ready list queue has a higher or equal priority to that of the running process, a context switch happens and the process that was running is put back into the ready list.

Q4. Analyze Xinu code and list all circumstances that can trigger a scheduling event.

Scheduling event can be triggered when

- In system/clkhandler.c, resched() is called whenever QUANTUM amount of time is passed
- In system/kil.c, resched() is called when process is killed.
- In system/ready.c, resched is called when the process enters PR_READY state.
- In system/receive.c, resched() is called when a process waits for a message.
- In system/recevtime.c, resched() is called when a process waits for a message for a defined amount of time.
- In system/resched.c, resched() is called when the defer attempt is successful.
- In system/sleep.c, resched() is called when the process enters sleep (PR_SLEEP) state.
- In system/suspend.c, resched() is called when the process enters PR_SUSPEND state.
- In system/wait.c, resched() is called when a process is waiting for semaphone.
- In system/yield.c, resched() is called when a process voluntarily yields the CPU to other processes.

Part 2(Lottery Scheduling)

The below files have been modified for the implementation of Lottery Scheduling.

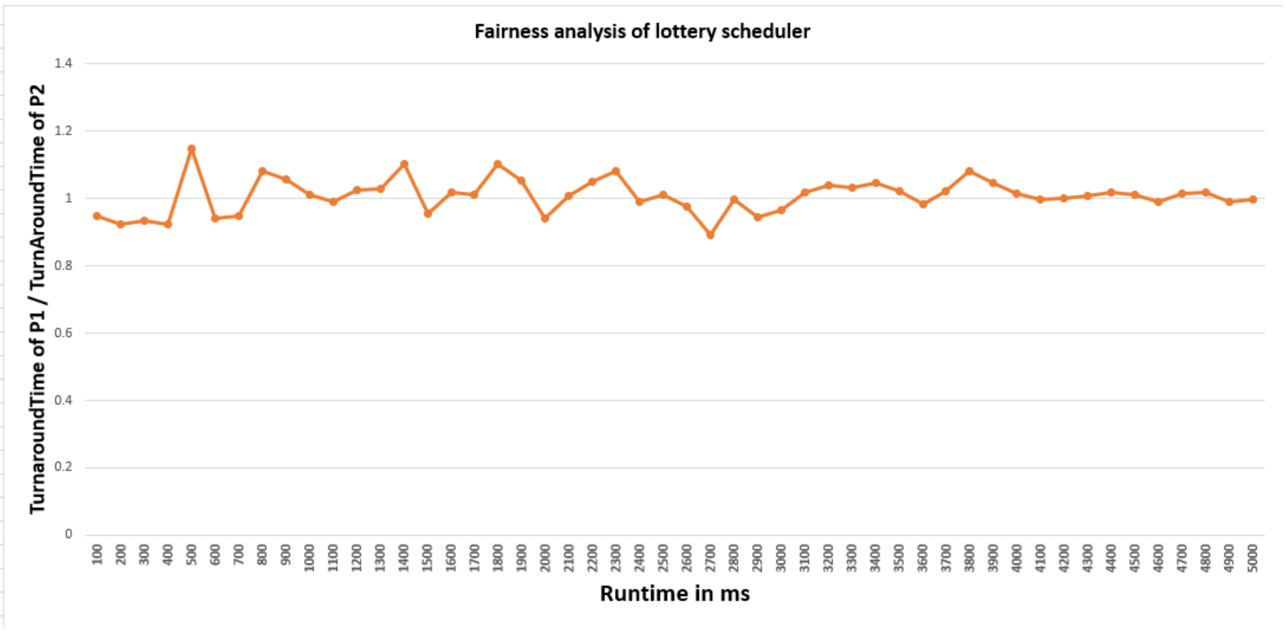
- In include/clock.h, added ctr100 for millisecond counting.
- In include/process.h – added parameters to process table such as process_creation_time, runtime, turnaroundtime, num_ctxsw, user_process, tickets.
- In include/queue.h, modified NQENT (+2) to consider lottery list.
- In system/clkhandler.c, increment runtime, ctr1000 of the running process every millisecond.
- In system/clkinit.c, ctr1000 = 0.
- In system/create.c, create_user_process() is added as per the project spec for user processes.
- In system/insert.c, added logic to add the process to lotterylist in the order as specified in spec.
- In system/initialize.c, null process is initialized for new parameters as well.
- In system/kill.c, calculate turnaround time for process that is killed.
- In system/ready.c, User processers are inserted into lottery list.
- In system/resched.c, lottery scheduling is implemented.

Implementation

- The old process can be either a User process or a system process. If the context switching is about to happen, User processes are put back in the lottery list queue whereas system processes are put back in the ready list.
- If there are no other system processes in ready list (except for NULL process), one of the user processes from the lottery list is picked and put into running mode based on lottery scheduling.
- If there is any system process in the ready list (except for NULL process), it is scheduled over the user process.
- Lottery() returns the pid of the user process to be scheduled.
- Lottery() adds the total tickets of all user processes and using rand() function, picks the desired user process to be executed.
- NULLPROC is executed only when there are no user processes to be executed.

Fairness analysis of lottery scheduling using the main function main.fairnessanalysis

- 2 processes A and B are spawned with same limited runtimes
- The above scenario is repeated for 50 times with increasing runtimes in multiples of 100ms.
- Graph of TurnaroundTimeofP1/TurnAroundTimeofP2 vs runtime is plotted below



From the above graph, we can see that as the process's run time is increased, turnaroundtimeofP1/TurnaroundTimeofP2 is close to 1. This implies that the difference in turnaround time of the processes reduces.

Part 3(Multi Level Feedback queue Scheduling)

Below files have been modified for the implementation of MLFQ Scheduling.

- In include/clock.h, added ctr100 for millisecond counting, priority_boost_counter for priority boost timer tracking.
- In include/process.h – added parameters to process table such as process_creation_time, runtime, turnaroundtime, num_ctxsw, user_process, timeallotment, upgrade, downgrade.
- In include/queue.h, modified NQENT (+2) to consider lottery list.
- In include/resched.h, definitions added for UPRIORITY_QUEUE, TIME_ALLOTMENT, PRIORITY_BOOST_PERIOD.
- In system/clkhandler.c, increment runtime, ctr1000, timeallotment, priority_boost_counter of the running process every millisecond.
- In system/clkinit.c, ctr1000 = 0.
- In system/create.c, create_user_process() is added as per the project spec for user processes.
- In system/insert.c, added logic to add the process to mulfqlist[] depending on the process priority.
- In system/initialize.c, null process is initialized for new parameters as well.
- In system/kill.c, calculate turnaround time for process that is killed.
- In system/ready.c, User processers are inserted into mlfqlist[].
- In system/resched.c, MLFQ scheduling is implemented.

Implementation

- The old process can be either a User process or a system process. If the context switching is about to happen, User processes are put back in the mlfqlist[] queue whereas system processes are put back in the ready list.
- If there are no other system processes in ready list (except for NULL process), one of the user processes from the mlfqlist[] is picked and put into running mode based on the mlfq scheduling.

- If there is any system process in the ready list (except for NULL process), it is scheduled over the user process.
- Mlfq() returns the pid() of the user process which is to be scheduled for context scheduling.
- Whenever a new user process comes, it enters mlfqlist[0] queue.
- Once a process uses up the TIME_ALLOTMENT in mlfqlist[0], it is downgraded to mlfqlist[1] and it goes on until mlfqlist[UPRIORITY_QUEUE -1] (downgrade - - for current process). Timeallotment is reset to 0.
- When PRIORITY_BOOST_PERIOD = priority_boost_counter, all the user processes in mlfqlist[] are moved to mlfq[0] (priority boost) (upgrade ++ for all processes). Timeallotment is reset to 0.
- Timeslice for all the mlfq lists are the same.
- Time_allotment for mlfq lists are increased by the factor of x2 from mlfq[0] to mlfqlist[UPRIORITY_QUEUE -1]