Assignment 3: Scheduler Simulator Analysis Report

Esli Emmanuel Konate 101322259

Carleton University

SYSC 4001 : Operating Systems

Introduction

This report analyzes the performance of three CPU scheduling algorithms that we implemented in our scheduler simulator: External Priority (EP) without preemption, Round Robin (RR) with a 100ms quantum, and a combination of Priority with Round Robin preemption (EP_RR). We tested 20 scenarios with CPU bound, I/O bound, mixed and special workloads to really observe throughput, turnaround time, response time, and how the system acts overall.

The simulator has fixed memory partitions (40MB, 25MB, 15MB, 10MB, 8MB, 2MB) with a total of 100MB, with smallest fit allocation. Processes move through NEW->READY->RUNNING->WAITING->TERMINATED states, with all the transitions saved in output files.

HOW

We created 20 scenarios and they were all organized into four categories. Tests 01 through 05 were CPU bound processes with non frequent I/O (intervals span between 600 and 900ms). Tests 06 through 10 simulate I/O bound workloads with frequent I/O requests (intervals span between 30 and 80 ms). Tests 11 through 15 were the mixed workload with both tests categories. Tests 16 through 20 had special cases, where there could be simultaneous arrivals, staggered arrivals, small processes, large memory necessities, and also testing of priority.

Each of the tests were executed on all three schedulers, which created 60 output files in total. We also analyzed transitions between states to calculate turnaround time (which was the completion - the arrival), the response time (which was first CPU - arrival), and also analyzed patterns related to the way the scheduler reacts to the various workloads.

Results and Analysis

CPU Bound Workloads (Tests 01-05):

External Priority had the best performance for tasks, which were CPU taxing. Since EP is non-preemptive, high priority processes were allowed to run to completion without any context switch overhead. In test01, Process 1 with priority 1 executed continuously from 0 ms until its first I/O at 700ms, this showcases efficient CPU utilization.

Round Robin had more context switched due to the 100ms quantum. Processes were interrupted every quantu, no matter the remaining burst time. This added overhead. However RR contained good CPU distribution and all processes received CPU time within the first 300ms of the simulation.

EP_RR was a combination of both approaches, but it also displayed the highest overhead due to checking priority each time scheduling decisions had to be made in addition to management of quantum.

I/O Bound Workloads (Tests 06-10):

Round Robin was excellent with I/O bound processes. The fact that it is preemptive ensured that when a process blocked for I/O, another one came immediately and obtained access to the CPU. In test06, response times averaged 67ms under Round Robin compared to 142ms under EP.

EP had a difficult time here because lower priority I/O bound processes waited behind higher priority ones, even when they were actually blocked on I/O. This led to longer response times.

Mixed Workloads (Tests 11-15):

Mixed workloads showed the pros and cons between schedulers. EP_RR had a good balanced performance. It had priority handling for the most important tasks, while the quantum prevented the effect of starvation. In test13 EP_RR had better turnaround than the EP alone while also maintaining better response time than RR.

Special Scenarios (TEsts 16-20)

Test16, which was simultaneous arrivals, showed EP finishing in a priority order, which was strict, while RR separated all processes fairly. Test19 with large processes, showed competition for the same memory. Processes which needed 40MB had to wait for partition 1, no matter the scheduler. Test 20 with priority testing also confirmed that EP respected priority ordering while RR just ignored them.

<u>Comparisons</u>

When examining execution traces through all of the tests we found the given patterns:

EP had the highest throughput for CPU bound workloads due to minimal context switching. RR had the best throughput for I/O bound workloads through better utilization of CPU during I/O waits.

RR had the best response time, which is important in interactive systems. Processes received their first CPU allocation within one quantum. Response times for EP varied based on the priority. High priority processes had the possibility to respond immediately, while low priority ones waited for large amounts of times.

RR ensured that all processes had a fair amount of CPU time. EP had no guarantees of fairness because low priority processes could suffer from starvation if high priority processes keep on arriving continuously.

EP had the lowest overhead with only scheduling decisions at the arrival of processes, completion, or I/O. RR overhead was somewhat moderate with timer interrupts every 100ms. EP_RR had the highest overhead due to the combination of both priority comparisons and management of quantum.

Memory Utilization

All schedulers made use of the smallest fit allocation. Small processes could occupy consistently partitions 5 and 6, while larger processes used partitions 1 to 4. Internal fragmentation depended on the size of processes, while external fragmentation was not a problem since we made use of fixed partitions. But, some tests showed that partitions 1 and 2 remained unused when only small processes arrived.

Conclusion

This analysis shows that no scheduler is optimal for all workloads. EP is the best for systems where throughput matters more than interactivity. RR is optimal for time-sharing and interactive systems needing fair CPU distribution and good response times. EP_RR is a good balanced approach for mixed environments where both priority and fairness are desired. We discovered through our simulation that context switching overhead impacts CPU bound performance. Preemption is important for good I/O bound response. Memory partition sizing will affect how processes are admitted no matter the scheduler. The selection of quantum give a good balance between responsiveness and overhead.

KEY to the repository : https://github.com/J3SIi3/SYSC4001