Simulation System for Performance Analysis of Scheduling Algorithm with Uniprocessor

Khin San Wai, Moe Sanda Htun
Computer University, Sittway
khinsanwaistw@gmail.com; moesanhtum@gmail.com

Abstract

Multiprogramming is used to improve the efficiency of the micro-processor. That is when the currently running process is blocked due to waiting for an I/O event, another process may be scheduled to run. At this moment, the issue of scheduling is involved, deciding which process should be executed by the processor, because usually more than one process may be ready for being scheduled.

This paper presents performance analysis for different scheduling algorithms for multi-programming in uniprocessor environment Performance analysis is based on average turn-around time, average waiting time and average ratio or turn-around time to service time. Performance analysis shows that RR requires more waiting time and therefore , increase more turn-around time and turn- around to service time ratio. On the other hand,Short Process Next (SPN),Shortest Remaining Time (SRT) and Highest Response Ratio Next (HRRN) algorithms perform the best.

Keywords: Uniprocessor Scheduling, Scheduling algorithms, multiprogramming, performance analysis.

1. Introduction

The design of real-time systems is usually approached by decomposing the system into a set of tasks that are scheduled on a set of processing resources. For uniprocessor scheduling it has been shown that the system's schedulability is maximized when the priorities are chosen in the inverse order of the tasks' deadlines (deadline monotonic scheduling policy). The scheduling problem under precedence constraints arises from the software implementation of real-time applications with deterministic communications: for efficiency reasons the real-time application is decomposed into interacting parallel activities that are scheduled on a set of computing resources. Since these parallel activities represent a global computation they may be subjected to precedence constraints to ensure that the correct global function is computed. Intuitively, to proportionally adjust task periods is useful for this purpose, the problem of preemptive scheduling of n sporadically arriving tasks on a uniprocessor. Static-priority scheduling solves this problem by assigning a priority to each task and every job is given the priority of the task that released the job. At run-time, a dispatcher selects, among jobs that have not yet finished execution, the job with the highest priority and this selected job executes on the processor.

The structure of this paper is as follows: section 2 is the related work of the uniprocessor scheduling section 3 is the multiprogramming,section is 4 is the system impletmentation and section 5 is conclusion.

2. Related Work

The design of real-time systems is usually approached by decomposing the system into a set of tasks that are scheduled on a set of processing resources. Real-time requirements impose constraints on the necessary computational resources. For these reasons, the scientific community has focused on establishing the conditions by which the task set is schedulable on a set of processing resources by some scheduling policy under given real-time requirements.

For uniprocessor fixed-priority (FP) scheduling it has been shown [Liu and Layland 1973; Leung and Whitehead 1982] that, under certain conditions, the system's schedulability is maximized when the priorities are chosen in the inverse order of the tasks' deadlines (deadline monotonic scheduling policy). Liu and Layland [1973] proved a similar condition for dynamic-priority scheduling on a uniprocessor, where, under certain conditions, schedulability is maximized when higher priorities are given to the tasks with earlier absolute deadlines. This scheduling policy is known as the earliest-deadline-first (EDF) scheduling policy.

Scheduling affects the performance of a system by determining (indirectly) how long processes have to wait until they may be executed. Because there will typically be more than one process in any of the process states at any one time, the operating system must maintain queues of waiting processes.

3. Multiprogramming

Multiprogramming is used to improve the efficiency of the micro-processor. That is when the currently running process is blocked due to waiting for an I/O event, another process may be scheduled to run. At this moment, the issue of scheduling is involved, deciding which process should be executed by the processor, because usually more than one process may be ready for being scheduled.

In uniprocessor, scheduling is the activity of deciding which thread of control gets to run on the CPU. In a single processor multi-programming system, multiple processes are contained within memory (or its swapping space). Processes alternate between executing (Running), being able to be executed (Ready), waiting for some event to occur (Blocked), and swapped-out (Suspend).A significant goal is to keep the processor busy, by "feeding" it processes to execute, and always having at least one process able to execute.

3.1 Uniprocessor Scheduling

Multiprogramming is used to improve the efficiency of the micro-processor. That is when the currently running process is blocked due to waiting for an I/O event, another process may be scheduled to run. At this moment, the issue of scheduling is involved, deciding which process should be executed by the processor, because usually more than one process may be ready for being scheduled.

The scheduling algorithm that could be selected depends on the requirements. Different algorithms yield different results. It is assumed that there are ten jobs and each takes a day to finish. The parameter sweep applications, created using a combination of task and data parallel models, contain a large number of independent jobs operating different data sets. A range of scenarios and parameters to be explored are applied to the program input values to generate different data sets.

The key to keeping the processor busy is the activity of process scheduling, of which we can categorize three main types:

Long-term scheduling is performed to decide if a new process is to be created and be added to the pool of processes. Long-term scheduling controls the degree of multiprogramming. The more processes that are created, the smaller is the percentage of time that each process can be executed. Thus the long term scheduler may limit the degree of multiprogramming to provide satisfactory service to the current set of processes. Whenever a process terminates, or the fraction of time that the processor is idle exceeds a certain threshold, the long-term scheduler may be invoked. What process to be created is another issue that long-term scheduling needs to deal with. The decision may be made on a first-come-first-served basis or it can be a tool to manage system performance. For example, if the information is available, the scheduler may attempt to keep a mix of processor-bound and I/O-bound processes. A processor-bound process is one that mainly performs computational work and occasionally uses I/O devices, while a I/O-bound process is one that uses I/O devices more than the microprocessor.

Medium-term scheduling is a part of the swapping function. It decides if a process should be loaded into the main memory either completely or partially so as to be available for execution. The swapping mechanism has been discussed in previous chapters.

Short-term scheduling is the most common use of the term scheduling, i.e. deciding which ready process to execute next. The short-term scheduler, also known as the dispatcher, is invoked whenever an event occurs that may lead to the suspension of the current process or that may provide an opportunity to preempt a currently running process in favor of another.

3.1.1 Preemption and Non-Preemption

Another issue relating to scheduling is whether a running process could be preempted or not. There are two categories:

- Non preemptive: In this case, a running process continues to execute until (a) it terminates or (b) blocks itself to wait for I/O or to request some operating system service.
- Preemptive: The currently running process may be interrupted and moved to the Ready state by the operating system. The preemption may possibly be made due to the arrival of a new process, or the occurrence of an interrupt that places a blocked process in the READY state.

Selects from among the processes in memory that are ready to execute, and allocates the processor to one of them. Uniprocessor scheduling decisions may take place when a process:

- Switches from running to waiting state.
- Switches from running to ready state.
- Switches from waiting to ready.

- Terminates.

3.1.2 Objective of Scheduling Algorithm

The primary goal of process scheduling is to determine the execution order of processes whilst attempting to maximize some objective functions measuring:

- processor efficiency,
- job turn-around time, and
- perceived response time.

The scheduling activities can be considered as state transitions in the process diagrams.

4. Proposed System

In this system, a survey of the best-known real-time scheduling techniques executing on multiple processors is conducted. These techniques solve the problem of scheduling sets of periodic and preemptable tasks. It is compared for the partitioning scheme as well as for the global scheme. Extensive simulation experiments are conducted to analyze and compare the performance of the algorithms. Figure 4.1 is the system architecture of the system.

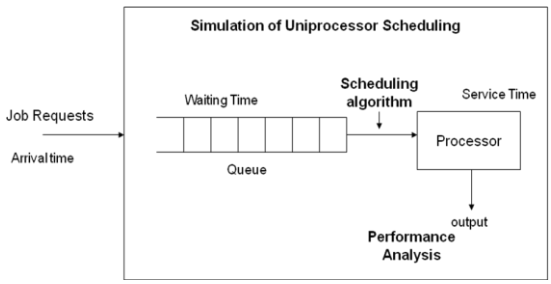

Figure 1: Proposed System overview

4.1. Types of Scheduling Algorithm

When all processes are of the same priority, or processes at a given priority level must be decided amongst, the selection function determines which process to execute next.

The selection function typically makes its decision based on resource requirements, or on the past execution profile of all processes.

In addition, the decision mode decides the moments in time at which the decision function is invoked:

- non-pre-emptive: once a process is executing, it will continue to execute until
  - it terminates, or
  - it makes an I/O request which would block the process, or
  - it makes an operating system call
- pre-emptive: the same three conditions as above apply, and in addition the process may be pre-empted by the operating system when
  - a new process arrives (perhaps at a higher priority), or
  - an interrupt or signal occurs, or
  - a (frequent) clock interrupt occurs

4.1.1 First Come First Serve

First-come-first-served (FCFS) is the simplest scheduling policy, also known as FIFO. With this policy, when a process becomes ready, it joins the ready queue and when the currently running process finishes, the process at the head of the ready queue, which has waited there for the longest time, is selected for execution.

4.1.2 Round Robin

A straightforward way to reduce the suffering of short processes is to use a time-sharing scheme, called round robin, with which the operating system assigns short time slices to each process and if the slices allocated for a process are not enough for it to complete, then the process has to wait until its time slice comes again. With the help of a clock, whenever a clock interrupt occurs, the operating system will check if the time slice for the current process ends.

4.1.3 Shortest Process Next

Another policy avoiding long processes in favor in FCFS is a nonpreemptive one, called shortest process next (SPN), in which the process with the shortest expected execution time is selected next.

4.1.4 Shortest Remaining Time

The shortest remaining time (SRT) policy is a preemptive version of SPN. Whenever a new process is brought in, the scheduler will run to perform the selection function to see if the new process has shorter remaining time than the currently running one. If yes, the current process is preempted and the new one is scheduled.

Another policy avoiding long processes in favor in FCFS is a nonpreemptive one, called shortest process next (SPN), in which the process with the shortest expected execution time is selected next.

4.1.5 Highest Response Ratio Next

Normalized turnaround time may be used to measure the performance of scheduling algorithms.

The highest response ratio next (HRRN) policy is proposed to minimize the average value of the normalized turnaround time over all processes. For each process in the process pool, we first compute the following ratio:

R=(W+S)S

where w is the time since the process was created and s is the expected service time. Then whenever the current process is blocked or completes, the process with the greatest value will be scheduled to run.

4.1.6 Feedback

Another approach to favor shorter processes is to penalize processes that have been running longer, thus avoiding predicting the expected processing time.

4.2 System Implementation

This system is implemented using Java Programming, Jdk 1.5. It is implemented as simulation program, where number of processes is provided as the user input, number of jobs is created based on that number and service time is generated randomly by the system. The simulation system is run on the desktop computer with Processor Intel(R) Pentium(R) Dual CPU 2.20GHz, Memory 2 GB of RAM. Arrival times are in serial format with 2 unit difference for each job. In this simulation system, following scheduling algorithms are implemented:

- First Come First Served (FCFS)
- Round Robin(RR)
- Shortest Process Next(SPN)
- Shortest Remaining Time(SRT)
- Highest Response Ratio Next(HRRN)
- Feedback(FB)

Simulated arrival times and service times are shown in Table 1. Arrival times are simulated with 2 unit different between each other. Service times are generated randomly between 1 and 10.

Table 1: Arrival Time and Service Time of Simulation process

Process	Arrival Time	Service Time
A0	0	6
B0	7	7
C0	4	9
D0	6	8
E0	8	3
F0	10	9

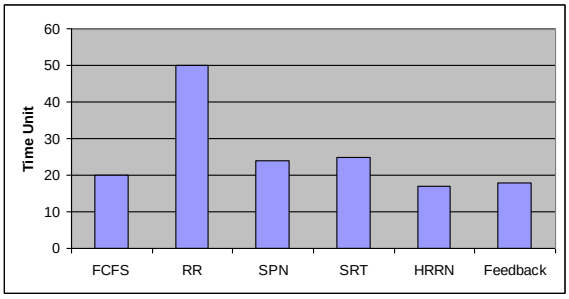

Figure 2 (a) Performance Analysis for Turn Around Time

Performance analysis for above running processes is shown detailed Figure 4.2(a). There are three performance analysis graphs in this system, average turn-around time graph, average waiting graph and average turn-around to service ratio graph. Scheduling algorithm with minimum values is supposed to be best algorithm. As in Figure 4.2(b) average turn-around times for all five algorithms except Round-Robin are low and almost the same. Therefore in this case, Round robin is the worst algorithm to generate the longest turn-around time.

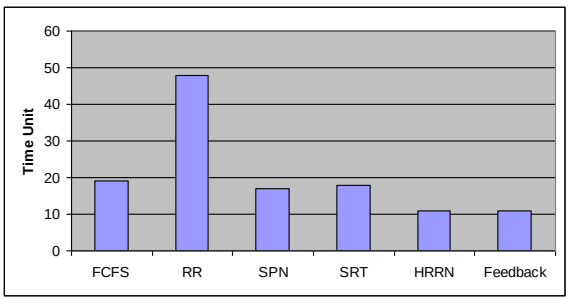

Figure 2 (b) Performance Analysis for Waiting Time

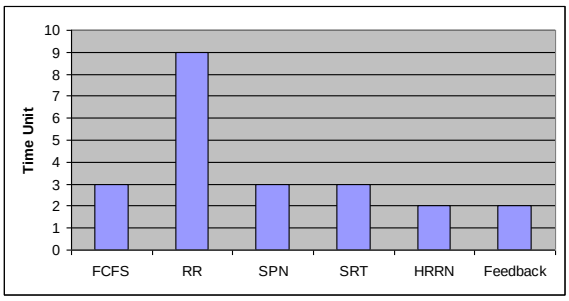

Figure 2 (c): Performance Analysis for Tr/Ts

In case of waiting time and ratio of waiting and service time, Round robin performs worst. In the middle graph of Figure 4.2(c) performance analysis for average waiting time graph, waiting time in FCFS algorithm gets a little bit higher and all other

looks remain the same. Table 2 presents the detailed statistics of FCFS scheduling algorithm.

Table 2 (a): Scheduling measurements for FSFS algorithm

Job	Waiting Time	Finished Time	Turn-around	Tr / Ts
A0	0	6	6	1
B0	4	13	11	1.571
C0	12	25	21	2.33
D0	19	33	27	3.375
E0	5	16	8	2.67
F0	23	42	32	3.556

5. Conclusion

Many considerations go into choosing a scheduling algorithm for use in a real-time system. One of these considerations should be how far removed the algorithm is from optimal behavior. Experimental results show that most of the time, best scheduling algorithm is based on the number of processes and their service times. But most of the time RR is the worst and sometimes, FCFS performs better than RR but worse than other algorithms. Remaining four algorithms best performs most of the time.

REFERENCES

[1] Andersson, B. “Static Priority Scheduling in Multiprocessors”, PhD Thesis, Department of Comp.Eng., Chalmers University, 2003.

[2] Leung, J. Y.-T. and Whitehead, J. “On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks, Performance Evaluation, number 2, pp. 237-250, 1982

[3] Mangeruca, L., Baleani, M., Ferrari, A., and Sangiovanni-Vincentelli, A. 2007. |Uniprocessor scheduling under precedence constraints for embedded systems design". ACM Trans. Embedd. Comput. Syst. 7, 1, Article 6 (December 2007), 30 pages. DOI = 10.1145/1324969.1324975 http://doi.acm.org/10.1145/1324969.1324975

[4] Niu, Jinzhong; "Uniprocessor SchedulingCSc33200: Operating Systems, CS-CCNY, Fall 2003

[5] Omar U. Pereira Zapata, Pedro Mej´ıa Alvarez, "EDF and RM Multiprocessor Scheduling Algorithms: Survey and Performance Evaluation", Av. I.P.N. 2508, Zacatenco, M´exico, D.F. 07000, 2004

[6] Baruah, S. “Optimal Utilization Bounds for the Fixed-Priority Scheduling of Periodic Tasks Systems on Identical Multiprocessors”, IEEE Transactions on Computers, pp. 781-784. 2004.

[7] Lecture 7: "Uniprocessor scheduling", Operating System 2230, Computer Science & Software Engineering