A Comparison of Common Processor Scheduling Algorithms

Mohammad B. Dadfar, Michael Brachtl, Sub Ramakrishnan

Department of Computer Science
Bowling Green State University
Bowling Green, Ohio 43403
Phone: (419) 372 2337 fax: 419 372 8061
email: datacomm@cs.bgsu.edu

Abstract

This paper describes some projects given in an operating systems course that deal with processor scheduling in a multiprogramming environment. We use a Java program to simulate a number of CPU scheduling algorithms including first-come first-served, shortest process next, round robin, shortest remaining time next, highest response ratio next, and feedback queue. We describe a single processor system and explain different performance criteria including response time, turnaround time, throughput, and processor utilization. Students can use the simulator to study the impact of additional soft constraints such as deadlines on the performance of a given algorithm.

1. Introduction

Study of operating systems concepts is an important subject area for most undergraduate computer science programs. A course on operating systems covers a range of topics including processes, CPU scheduling, concurrency, file and memory management. Where possible, hands-on programming projects can be used to enhance the learning process and to gain additional insight into the topic. This paper concerns study of CPU scheduling algorithms in an operating systems course. There are many textbooks available for teaching operating systems courses[1, 2, 3, 4]. In the past few years we have used different textbooks including the *Applied Operating System Concepts* by Silberschatz, Galvin, and Gagne that uses Java to demonstrate different concepts.

Processor scheduling can be used to create a multiprogramming environment even with a single processor. There are many possible scheduling algorithms each having different characteristics. In general efficient scheduling algorithms attempt to maximize processor utilization. We describe some projects given in an operating systems course that deal with processor scheduling. The key to multiprogramming is processor and I/O scheduling. We use a Java program to simulate a number of CPU scheduling algorithms. The program can be parameterized to show the behavior of the algorithms and for visually displaying the results of the simulation. Some of the algorithms discussed in this paper are first-come first-served, shortest process next, round robin, shortest remaining time next, highest response ratio next, and feedback queue. The algorithms described

in this paper have practical relevance and should help reinforce the theoretical concepts introduced in the operating systems course.

We describe a single processor system. In general in a multiprogramming environment our goal is to minimize average waiting time. This in turn will increase the processor utilization and the throughput of the system. We explain different performance criteria including response time, turnaround time, throughput, processor utilization and study the impact for many scheduling algorithms, including preemptive and nonpreemptive scheduling disciplines. By completing these projects students explore the processor scheduling algorithms. We show how the simulation can be extended to include many processors. In a multiprocessor system, we study an environment that contains several (simulated) processors. Students can use the simulator to study the impact of additional soft constraints such as deadlines on the performance of a given algorithm.

2. Processor Scheduling Algorithms and Performance Metrics

There are three common processor scheduling that are discussed in an operating systems course. The *long-term* scheduling is concerned with the decision of adding new processes to the pool of active processes for execution. The *medium-term* scheduling is a part of swapping function and deals with the decision of adding the processes in the secondary storage to the number of processes that are partially or fully in main storage. Once a process has been admitted to main memory the third level of scheduling referred to as *short-term* scheduling is used. Based on a particular scheduling algorithm a decision is made as to which ready process will be executed next by the processor. In this paper we concentrate on the *short-term* scheduling.

A process could be in one of the following states: *new* (it has just been created and not yet been admitted to the pool of executable processes), *ready* (it is prepared to execute once the processor is assigned to it), *running* (currently has the processor and it is being executed), *blocked* (cannot execute until some event occurs), and *exit* (it has been released from the pool of executable processes). State transition refers to the change of a process from one state to another. For example a process has a ready to running state transition when the operating system assigns the CPU to that process.

The objective of scheduling algorithms is to assign the CPU to the next ready process based on some predetermined policy. We study the following scheduling algorithms:

- *First-Come First-Served* (FCFS) is a nonpreemptive algorithm that assigns the CPU to the process in the ready queue that has been waiting for the longest time. This is a simple algorithm and it is not used very often in modern operating systems. A long process could cause a delay for all other processes that arrive after that process.
- *Shortest Process (Job) Next* (SJN) is another nonpreemptive algorithm that attempts to decrease the average waiting time (and response time) of the system. This algorithm performs better than FCFS, however, it is not fair to long processes. In general preemptive scheduling algorithms are preferred due to their abilities to switch the CPU to another process even when the current running process is not completed.
- In *Round Robin* (RR) scheduling a time slice is defined and the CPU is assigned to a process for a maximum of one time slice or until the process releases the CPU (whichever

comes first). This algorithm requires more overheads but it is fair to all processes and performs better than nonpreemptive scheduling algorithms.

- *Shortest Remaining Time Next* (SRTN) is a preemptive version of SJN algorithm where the remaining processing time is considered for assigning CPU to the next process.
- *Highest Response Ratio Next* (HRRN) selects a process with the largest ratio of waiting time over service time. This guarantees that a process does not starve due to its requirements.
- *Feedback Queue* (FQ) scheduling algorithm partitions the ready processes into several separate queues and the processes are assigned to one queue and they are allowed to move between queues. Each queue has its own scheduling algorithm.

We use the following scheduling criteria to evaluate and compare the effectiveness of different scheduling algorithms.

1. *Average turnaround time* (the time between submission of a process and the completion of the process) this parameter is related to waiting time and response time.
2. *Waiting time* (the sum of the time periods a process waits in the ready queue)
3. *Response time* (the time between the submission of a process and the producing of the first output response)
4. *Throughput* (the number of processes completed per unit of time)
5. *Processor utilization*: One of the main objectives is to keep the processor as busy as possible. In real systems processor utilization usually ranges from 40 percent to 90 percent.

An effective scheduling algorithm will maximize the processor utilization while it reduces the turnaround time, waiting time, and response time. Other factors such as *fairness* and lack of *starvation* may be considered when choosing a particular scheduling algorithm.

3. Simulation Architecture

This section provides an overview of the simulation architecture for studying processor scheduling algorithms. The simulation software is written in Java which is platform independent. A schematic of the design is given in Figure 1.

The simulation program is modular and is composed of a number of classes. The simulation starts at the main class, *Sim*. Here we instantiate a number of processes (of type *Process*) and one or more processors (of type *Processor*). A queue allows Processor to access the Process. Each derived class of Processor (for example, *FCFS*) implements a scheduling algorithm. Simulation proceeds by picking the next process from the process queue and invoking the scheduling algorithm. Upon completion, the results are available in a results object *Results*. A utility class, *Putil* implements statistical distribution to be used for process arrival patterns (such as exponential interarrival time).

Sim is the driver class. The program starts by creating a number of processes (each is of type *Process*) with random execution times and puts them in a data structure (Java type *Vector*). This is achieved by invoking the method, *Vector createProcessSet* of *Sim*. This method takes in

process generation parameters such as the interarrival distribution times and number of processes to be simulated.

createProcessSet instantiates processes. For each process that enters the system, we have an instance of Process. Each process has a number of parameters some of which is common regardless of the scheduling algorithm to be used while others are specific to the algorithm. Some of these parameters are arrivalTime, cpuTime, processID, queueID and the like.

The class Processor implements a CPU. Its data members include a queue (that processes are lined up in) and other local attributes of the processor. The processes to be run on this processor are added to its queue using *addProcess* method of Processor. There can be many instances of Processor. It is implemented as a thread and Processor picks up the next process from its queue and executes it according to the chosen scheduling algorithm.

As noted in Section 2, a number of scheduling algorithms can be simulated using our software. The scheduling algorithm that is to be simulated is passed as a parameter to the *run* method of Processor. As depicted in Figure 1, each algorithm is a separate derived class of Processor. For example, *FCFS* extends Processor.

As processes make progress through the system, the algorithm updates relevant statistics information which are maintained in the class, *Results*. This class maintains a number of cumulative performance metrics such as number of processes completed, total CPU time used, histogram of CPU use and the like. Once simulation ends, these quantities can be refined to generate a number of steady state statistics.

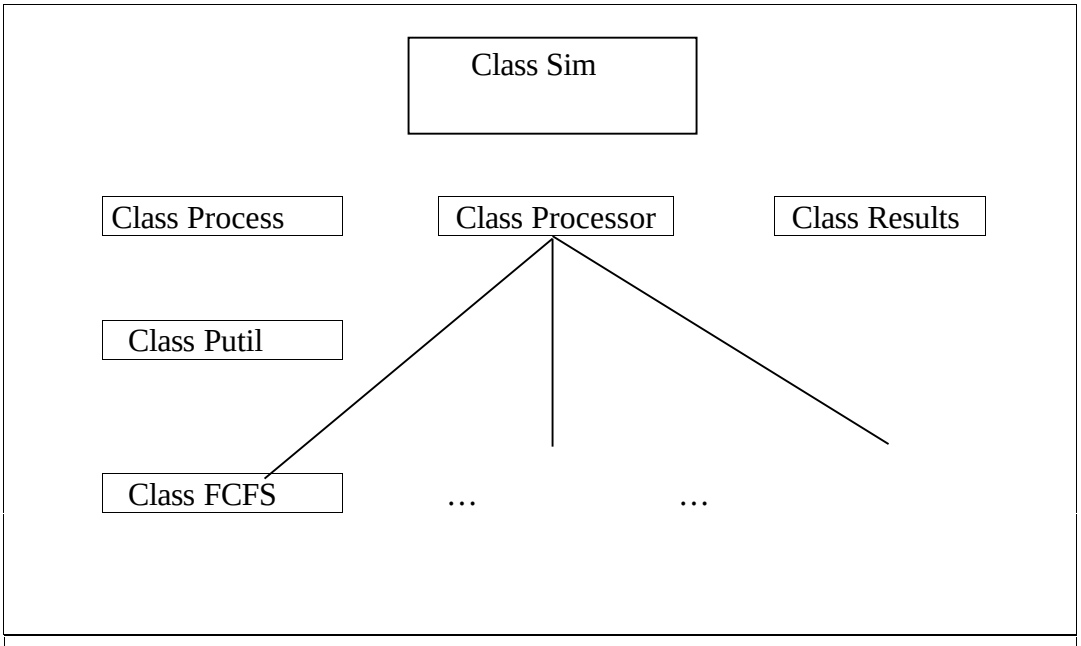

Figure 1: Class Diagram

4. Student Projects

This paper described an approach to simulate CPU scheduling algorithms in Java. Our approach is quite elegant and permits students to hang other scheduling algorithms off of the Processor class (as leaves of Processor, see Figure 1). Our modular design also permits addition of other performance metrics.

The instructor can make the program available for students use which includes two scheduling algorithms (FCFS and RR). Possible class projects typically ask the students to make additions to the code provided by the instructor. The software can be used for two possible types of class projects. In the first type, students are asked to run the simulation and do a walk through to answer specific questions about the algorithms being simulated. In the second type, students can enhance the functionality of the code provided by the instructor. Possible functionalities include simulation of other scheduling algorithms (such as SJF) and or inclusion of additional performance metrics (by modifying the class, *Results*).

The code for our simulation and a couple of detailed project ideas are available and may be requested from the authors.

5. Concluding Remarks

In this paper we talked about some scheduling algorithms that are discussed in an operating systems course. We have written a Java program to simulate a number of CPU scheduling algorithms. The architecture of this simulator was discussed in Section 3. It is a modular program and starts with its main class, *Sim* which is a driver class. The program creates a number of processes with random execution times. Each process has several parameters such as *processID, queueID, executionTime,* and *arrivalTime*. We believe our students are interested in completing the projects and the projects help students better understand CPU scheduling algorithms and their performance.

Bibliography

1. Silberschatz, A., Galvin, P., and Gagne, G., "Applied Operating System Concepts," (First Edition), Wiley & Sons, 2000.

2. Silberschatz, A., Galvin, P., and Gagne, G., "Operating System Concepts," (Sixth Edition), Wiley & Sons, 2002.

3. Stallings, W., "Operating Systems," (Third Edition), Prentice-Hall, 1998.

4. Tanenbaum, A. S., "Modern Operating Systems," (Second Edition), Prentice-Hall, 2001.

MOHAMMAD B. DADFAR
Mohammad B. Dadfar is an Associate Professor in the Computer Science Department at Bowling Green State University. His research interests include Computer Extension and Analysis of Perturbation Series, Scheduling Algorithms, and Computers in Education. He currently teaches undergraduate and graduate courses in data communications, operating systems, and computer algorithms. He is a member of ACM and ASEE.

MICHAEL BRACHTL
Michael Brachtl is a graduate student at the university of Salzburg. He received a Masters degree in computer science from the Bowling Green State University. His interests include database systems, simulation techniques and network programming.

SUB RAMAKRISHNAN
Sub Ramakrishnan is a Professor of Computer Science at Bowling Green State University. From 1985-1987, he held a visiting appointment with the Department of Computing Science, University of Alberta, Edmonton, Alberta. Dr. Ramakrishnan's research interests include distributed computing, performance evaluation, parallel simulation, and fault-tolerant systems.