Simulator

EC-353 Operating System Project

Project guide: **Prof. P. Sateesh Kumar**

Computer Science and Engineering,

IIT Roorkee

1. Introduction

Scheduling is one of the most crucial aspects of Operating System design. CPU Scheduling is the basis of multi-programmed operating systems. By switching the CPU among processes, the operating system can make the computer more productive. In a single processor system, only one process can run at a time; any others must wait until the CPU is free and can be rescheduled. The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization [1].

CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated to the CPU. There are many different CPU Scheduling Algorithms. In our Simulator design we have covered the following algorithms: Shortest-Job-First (SJF), first-come-first-serve (FCFS), Priority Scheduling, Round-Robin method and Shortest-remaining-time-first (SRTF).

The wide variety of scheduling algorithms demands that there must be methods to select among algorithms. Simulation methods determine performance imitating the scheduling algorithms on a representative sample of processes and computing the resulting performance. However, simulation can add best provide an approximation of actual system performance; the only reliable technique for evaluating a scheduling algorithm is to implement the algorithm on an actual system and monitor its performance in a real-world environment.

2. Goals

As a part of the course project EC-353 Operating Systems, we have successfully demonstrated the various algorithmic techniques discussed in class. In order to test the different algorithms, namely Shortest-Job-First (SJF), Shortest-remaining-time-first (SRTF), first-come-first-serve (FCFS), Priority Scheduling and Round-Robin method, a simulation environment has been designed, coded and implemented. Further, a user-interface has also been integrated so as to improve the functionality of the coding environment. Our first major concern was to achieve this functionality with minimum time as well as space complexity and second concern was to keep interface design simple and user friendly to the maximum extent possible.

3. Design

Our simulator is designed in the form of an application that runs on the top of CPU provided. It takes different C++ programs as inputs and gives the final result in the form of Average Turnaround and Waiting times, based on the burst times and arrival times of each of the processes. The user can compare different scheduling algorithms based on the results provided and comment upon the usefulness of the particular algorithm for a particular set of input processes. Our design is divided in two parts, as shown below-

- **Main Scheduler**- It has been designed to take the algorithm as input from user and fetches the relevant data from the file named “process.txt” and gives out relevant details like turnaround time and waiting time of each process as well as the average values.
- **User Interface-** It has been designed to take as input user programs, runs them many times, calculates their average burst times and stores them in the file “process.txt” from where it could be fetched by main simulator.

4. Implementation

Our implementation of the simulator has been divided into two parts as per the design namely, 1.Main scheduler and 2.User Interface whose implementation details have been described below-

4.1 The Main Scheduler

The main scheduler is divided into various modules where each of them corresponds to a different scheduling algorithm. The modules have been implemented in the form of header files (.h extension) and are incorporated into the main program simulator.c. The details of each module are as follows:

(a) variable.h

It is the process descriptor file.

struct incoming_process includes the many variables/identifiers in its declaration: char name[50](name of process); float arrival_time; float burst_time; int priority; int array state[5](New, Ready, Running, Waiting, Terminated); float array termination_time[5]; float array waiting_time[5]; float array turnaroound_time[5]; float remaining_time.

Process process[50] array signifies the ready queue containing processes for execution.

*struct _data_* has been made for passing multiple arguments to the function from which the thread will start execution.

static int quantum initialized to 2, denotes the round robin scheduling time quanta. The value is subject to change as per the given requirement of the user.

(b) headers.h

It contains all the header files that are included in the simulator namely, <stdio.h>, <stdlib.h>, <string.h>, <pthread.h>, "variable.h", "function.h", "fcfs.h", "priority.h", "sjf.h", "rr.h", "srtf.h"

(c) function.h

The file consists the definition of the three functions:

get_detail_from_file(void) fetches the lines from the process_file that has been originally supplied as an input.

*split(char *line, char tab)* function splits the supplied line from the input file into various variables for further analysis.

print_process() prints out the process name, arrival time, burst time and priority for all the processes (total_process variable).

(d) fcfs.h

The file consists of the implementation of FCFS algorithm. To implement this algorithm, we utilise the functionalities of **pthreads**. The file includes two prime functions: FCFS(void) and void *fcfs_thread(void *data).

void FCFS() threads are created using **pthread_create**() function and a function call to fcfs_thread is made.

*void *fcfs_thread(* data)* the function implements the actual algorithm on threads to calculate the name, burst time and completion time of all the processes.

(e) sjf.h

The file consists of the implementation of shortest job first scheduling algorithm. An additional function (parallel to the two functions described in fcfs scheduling), *int shortest_process(float curr_time)* is introduced to calculate the identity of the shortest process in the waiting queue.

The function *void sjf_non_parts(void* data)* is utilized by the *pthred_create()* and is called inside the *void SJF()* function. The array *process[i]* in the end stores the final details (burst_time, waiting_time, turnaround_time etc.) of each process in the ready queue which are later displayed.

(f) rr.h

The file consists of the implementation of the round-robin algorithm. The implementation is divided into two functions: *void rr()* and *void *rr_thread(void *data)*. In a similar manner, the first function utilises pthreads to call the **rr_thread(void *data)*. Burst time of all processes is calculated. The *void *rr_thread(void *data)* function gets passed in the threads and arranges *process[i]* array in the round-robin method, signified by the *quantum* variable.

(g) priority.h

The file consists of the implementation of priority scheduling algorithm. It consists of three functions:

void priority_scheduling() implements the actual priority algorithm by using pthreads. It makes use of the other two functions.

int priority_order(float curr_time) selects the next process that is to be executed next, keeping in mind the priorities of the remaining processes.

*void *priority_scheduling_func(void *data)* is the function that is passed into *pthread_create().*

(h) srtf.h

While implementing the shortest-remaining-time-first algorithm, this module makes use of three functions:

void srtf(void) is the function that implements the actual SRTF algorithm by making use of pthreads and other two functions.

int srtf_shortest(float) selects the processs which is to be executed next by cheking which process has the shortest remaining time, whenever a new process arrives in the ready queue.

void *srtf_thread(void *) is the function that is passed into *pthread_create().*

(i) simulator.c

It is the main function that implements the simulator. It is called by the Command Line Interface (discussed later), and asks the user about which algorithm is to be implemented and based on the arrival and burst times of each processes, calculates the final result.

4.2 The User Interface

The interface.cpp file is designed to make the experience of our simulator a bit easier.

The simulator accepts a file in which all details are pre-filled for process name, Arrival time, Burst time and priority. Here the role of interface comes in.

What it does is, it takes input of real .cpp and .cc/.c files kept in the same master folder and calculates their burst times and takes input from user for its priority and arrival times, process name being the same as the file name.

Working:-

For the Burst Time –

We start a clock before running a program (process) and note the start_time. Then the process runs in another module and on returning we note the end_time by invoking the clock again. The difference gives us the burst time and then all the data is written in the file process.txt. This usually comes out to be a few ms (milliseconds)

For the Arrival time-

We directly take input of a processes time of arrival from the user in ms (milliseconds).

For Priority –

User has to give a unique integer for the order of its importance in Lowest-most important order.

The file is overwritten every time the file is run. This way no clashes occur and appending is prohibited. What can be done is that the simulator and interface can be integrated together for a more combined experience although it's better to have them separate to avoid race condition between simulator and UI.

4. Conclusions

We have successfully implemented the above design with minimum space as well time complexity possible but are still open to any changes which can help us minimise it further. Also we will like to simplify it further for better user experience. We will also like to improve its functionality and utility by adding the options for simulating more algorithms like queue and multilevel feedback queue scheduling etc.We will also be adding file system as well as memory management to the simulator design.

5. References

[1] Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne; Chapter 5, CPU Scheduling.

6. Contributors

Team members in order of Enrollment numbers-

12112010: Anil Kumar Saini

12114001: Abhijeet Gaur

12114007: Amandeep Chouhan

12114035: Kaushal Kishore

12114042: Nairitya Khilari

12114062: Shubham Gupta

12114063: Shubham Singh

12115081: Rajat Jain

Arikilla Priyathan

Arun