Improving the Operating System Resource Management by Processes Simulation

Muhammad Imran[1], Wadee Alhalabi[2]

Department of Computer Science
King Abdulaziz Univeristy, Jeddah, Saudi Arabia

mimran@stu.kau.edu.sa[1]
wsalhalabi@kau.edu.sa[2]

Abstract— Multiprocessing systems are in focus for every high performance processor family. With multiprocessing comes the problem of scheduling which we have tried to address in this paper. In this paper, we have explained the resources and processes simulation that we have designed to optimize the process scheduling task of operating systems. This explains how to model and simulate a system to evaluate its performance and utilization and then analysing the results to go for the improvements in the real system. We have taken three independent processes and five resources which consist of one or more instances, for our experiment and evaluation task. This paper presents the implementation done in the C language provides option for the work to be adapted for modeling and simulation of any other scheduling tasks that need to be performed by the OS. Then a proper set of results can be interpreted for target system improvements.

Index Terms— FCFS Scheduling, Multiprocessing, Operating System, Process Management, Processes Simulation, Process Scheduling, Resources Management

1 Introduction

THIS paper is an outcome of the simulation of processes and resources management in an operating system. A design is prepared for the simulation purpose which is synonymous to the operating system process management where processes are being loaded into the memory and they keep on requesting for the resources to complete their executions. One of the aspects taken care while the designing was to make is reusable where it might be made a component of some advanced process scheduler added in the OS. That process scheduler will be using the simulation and use the output for its input to generate the optimized sequencing and scheduling for the processes currently in the memory. This paper explains the experiment on the project in which three independent processes accessing different resources having specified the number of resources required to complete its task. Every resource has one or more instances that can be allocated to the processes upon their request. Any process being loaded into the memory is supposed to have mentioned the resources it may request for its task completion. Thus each process in the memory needs a certain set of resources to get the work done. The complete cycle, that consists of some finite number of steps, which are needed to be performed keeping in view the performance, efficiency makes a good size problem that is modeled and simulated to get some results for taking the steps for improvement in the process scheduling.

Main goal of this project is to analyze the bottlenecks in dealing with process scheduling. As we are having multiple resources where the processes request one after another in a specified order, we need to figure out the best configuration (sequence) of these resources request. Processes complete specific percentage of their task by getting each resource and using it for some amount of time. We need to analyze the bottleneck resources which are causing more delays and increasing the waiting queue size. Moreover, by running the simulation multiple times by changing the number of resource instances (every instance with equal working ability), we will be looking for the impact on overall performance. Also, by adding Job Averaged Statistics and Time Averaged Statistics in the simulation, we will be having a larger picture for the performance analysis of our resources utilization.

2 Literature Review

The Operating System Simulation (OSS) system is an operating system simulation tool to simulate the multiprocessing operating system which contains six major components, which are process management, scheduler, memory management, message management, semaphore management and resource management [1]. OSS is used to get the clear understanding of how an operating system works and to study the impact of various algorithms on the performance of an operating system. Operating systems provide many functions in which the process management is one of the most important. For the process management, various types of scheduling algorithms are used. The scheduling algorithm's main goals include best CPU utilization, Throughput, T.T., Waiting Time (W.T.), Response Time (R.T.) and threads fairness. In the last thirty years there has been a huge amount of research in the area of disk scheduling [2]. The main objective was to design the scheduling algorithms with some verifiable properties. The CPU switches among processes in the multi programming environment and this is achieved by the use of CPU scheduling techniques/algorithms. The OS should allow processes as much as possible to run all time so the CPU utilization can be maximized. Thus a number of processes are kept in memory

concurrently and each process is selected by the OS so that it can occupy the CPU. Therefore, scheduling is an essential OS function and nearly all the computer resources are scheduled before they can be used [3], where the CPU is considered as a primary resource. FCFS is one of the scheduling algorithms known as the very basic for scheduling. It is a non-pre-emptive algorithm in which we allocate the CPU first to the process which comes first to access the CPU. A queue is made for this purpose and processes join the queue. The running process (one at a time) keeps the CPU and releases it only when it has completed its execution. Then SJF focuses on the shortest jobs/processes first which will take less time to do their processing, and if there is a tie then FCFS is used here. There exist pre-emptive and non-pre-emptive versions for this algorithm [4].

3 Problem Statement

In this modeling and simulation, processes, resources with several instances and each process having a resource request sequence are considered as a single system. Two main components of system are identified, which include resources and the processes requesting the resources.

This system consists of 5 resources, all are working in parallel. Each of the resource has one or more instances to serve the processes, depending on the request made for that resource. Table 1 shows the distribution of the instances that exists for each resource.

TABLE 1
Instances Distribution at Resources

Resource #	Resource	No. of Instances
1	Resource 1	3
2	Resource 2	2
3	Resource 3	4
4	Resource 4	3
5	Resource 5	1

We have taken three process i.e. Process 1, Process 2, Process 3. This categorization is based upon different natures possible for any independent processes. Theya are listed in Table 2.

TABLE 2
Process Types Used

Category #	Process Type
1	Process 1
2	Process 2
3	Process 3

The Figures below, Fig.1, Fig.2 and Fig.3 depict the complete picture of the system, showing the resource request sequence of each process.

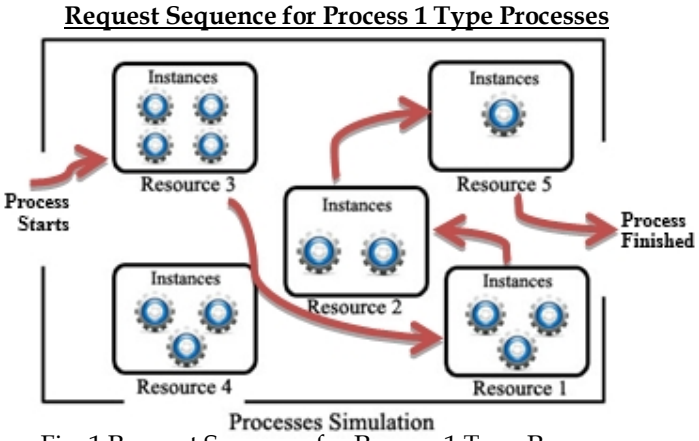

Fig. 1 Request Sequence for Process 1 Type Processes

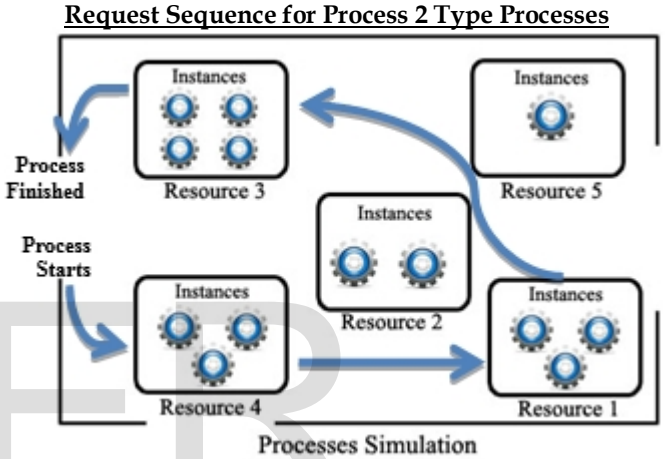

Fig. 2 Request Sequence for Process 2 Type Processes

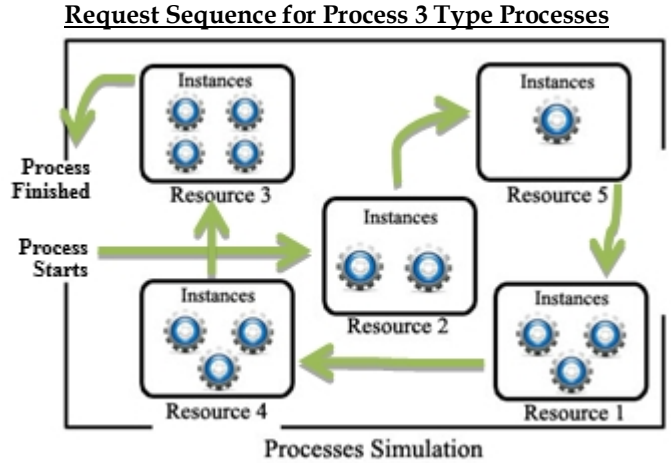

Fig. 3 Request Sequence for Process 3 Type Processes

4 Solution towards Modeling Experiment

As it has already been discussed that there are three types of independent processes, we assigned the probabilities to each of our processes. Process Type 1 has 30% chances of requesting the resources, Process Type 2 has 50% chances of requesting the resources and Process Type 3 has 20% chances of requesting the resources for its execution completion.

Table 3 shows the events that are to be occurred:

TABLE 3
Events to Be Occurred

Event Number	Event Description
1	Creation of a process in the System
2	Pre-emption of resource after process has completed its task.
3	Execution completion of the process

In the above events list, the pre-emption of resource is pre-emption of current resource, so it does not mean the process being terminated. The process does not terminate and keeps on requesting, acquiring and pre-empting the resources one after another until it acquires and pre-empts the last resource in its resource request sequence. Moreover, there are separate queues for each resource in the system and to have more information of the states during the simulation we used some data structure with the following attributes:

Process Type: contains the type of process
Time of arrival of process: the arrival time of the process to the current resource's waiting queue
Resource Number: it tells us about the current resource number which is acquired by the process.

To measure the delays in the queues of different resources, while pre-empting one resource and requesting for the other, we used some variables. To have the average delays (waiting of process) in the queues of each resource throughout the resource request sequence of process, following variables are used:

D1: delay in the queue of Resource 1
D2: delay in the queue of Resource 2
D3: delay in the queue of Resource 3
D4: delay in the queue of Resource 4
D5: delay in the queue of Resource 5

The above delays will be regardless of Process Type. These will only tell the delays that are there in specific queues. To find the Process Type specific delay we will be using separate variables for each process type resulting in having three more variables. Also keep in mind that these delays will now be regardless of resources.

5 IMPLEMENTATION

The computational model of the simulation has its basis upon the below discussed algorithms.

5.1 Request Function Algorithm

The 'request' function of the simulator is being used for two purposes:

1. Event Function, which is called upon the loading of the new process.
2. Called in the last when the process has completed its work with one resource, pre-empting that resource and needs to request the new resource.

Function ProcessStarted (isNewProcess)

1. *If this is a new process*
   a. *Schedule the new process event;*
   b. *Generate the process type for this new process;*
2. *Determine the appropriate resource to be acquired by this process based on process type*
3. *If all instances of this resource are allocated*
   a. *Put the process to the end of the waiting queue of this resource;*
4. *Else*
   a. *Set the delay for this process to 0;*
   b. *Make an instance, from that resource, Allocated and get the statistics;*
   c. *Schedule the finish event for that process;*

Return;

5.2 Depart Function Algorithm

The 'depart' function is called, depending on the process type and the current state of that process's resource request sequence, when the process finishes the task with one resource and needs to request the new resource or has completed its execution needing no more resources.

Function ProcessFinishedTask ()

1. *Get the Resources that process has recently pre-empted.*
2. *If the wait queue of this resource has got empty*
   a. *Set an instance of that resource to free/idle get the Statistics;*
3. *Else*
   a. *Remove the first process from the Queue of that resource;*
   b. *Compute the delay for this process and get the statistics;*
   c. *Schedule a finish event for this process;*
4. *Are there any more resources that this process needs to request and acquire*
   a. *Add 1 to resources for the finishing process;*
   b. *Call the 'ProcessStarted' for this process by setting the flag isNewProcess = false.*
5. *Else*

Return;

The main flow of the processes simulation program can be easily tracked from the main function of the program. The simulation program uses text files for the purpose of input and output of the running simulation. The input parameters are thus set in a text file which is read by the program in its beginning and then rest of the simulation is run and statistics are gathered, after which the results are written to a text file.

The algorithmic representation of the main flow is as below:

Function Main()

1. *Define the variables to be used for states representation of the system.*
2. *Get input parameters for the simulation from specified text file and store them locally*
3. *Initialize the things required for the simulation these things include:*
   a. *Storage allocation for the lists*
   b. *Initialization of these lists and setting their heads and tails*
   c. *Initializing the simulation clock*
   d. *Initializing the variables used for statistics purposes*
4. *Schedule the loading of the first process*
5. *Repeat until the end Simulation event occurs*
   a. *Determine the next event*
   b. *Call appropriate event function (processStart or processFinish)*
6. *Get all the statistics that were gathered and write them to a text file.*
7. *Terminate*

6 Simulation Results

With the specific configuration used for our specific experiment and detailed in this paper we ran the simulation with a certain input file and analysed it by looking at the statistics, we have concluded that the bottlenecks are at the resource 1, 2 and 4. The order of the severity depends on which statistics do we consider i.e. average number of processes in queue, utilization of the resource instances, or the average delay for the processes in the queue.

The output generated by the simulation run of processes simulation program is in Table 4 and Table 5.

- *Author Muhammad Imran is currently pursuing masters degree program in Computer Science at King Abdulaziz Univeristy, Saudi Arabia, E-mail: mimran@stu.kau.edu.sa*
- *Co-Author Dr. Wadee Alhalabi is currently Assistant Professor in Department of Computer Science at King Abdulaziz Univeristy, Saudi Arabia, E-mail: author_name@mail.com*

TABLE 4
Simulation Run Output

Process Type queue	Average total waiting time in waiting
1	Creation of a process in the System
2	Pre-emption of resource after process has completed its task.
3	Execution completion of the process

Overall average process total delay = 12.836

TABLE 5
Simulation Run Output

Resource #	Avg. No. in Queue	Avg. Utilization	Avg. Waiting Time in Queue
1	16.194	0.967	4.074
2	21.586	0.984	10.589
3	0.747	0.718	0.188
4	11.094	0.944	4.022
5	1.894	0.797	0.953

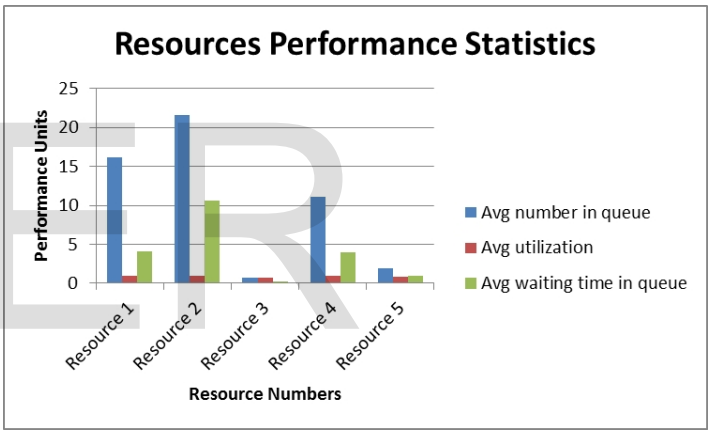

Fig.4 Resources performance graph

7 Conclusion

After running the simulation for the fine tuning of the processes scheduling and resources utilization, we ran the simulation for three more times by adding one more instance each time one by one to each of the above mentioned bottleneck resources i.e. 1, 2, and 4 (as the resource 3 and 5 seemed not to be the bottlenecks).We wanted to check that adding an instance for which resource will have the greatest impact on the better resource utilization and process scheduling. By looking at the Table 6, we note that an instance should be added to the resource 4 as doing this will decrease the job delay to maximum as compared to adding the instances to other resources.

The results from these three simulation runs are show in the Table 6.

TABLE 6
Simulation Run Results

	Number of instances of the resources in our experiment					Over all Avg. Job Total Delay
	Rsc. # 1	Rsc.# 2	Rsc.# 3	Rsc.# 4	Rsc.# 5	
The Original Configuration of instances	3	2	4	3	1	10.9
Adding instance to resource 1	4	2	4	3	1	8.1
Adding instance to resource 2	3	3	4	3	1	7.6
Adding instance to resource 4	3	2	4	4	1	7.5

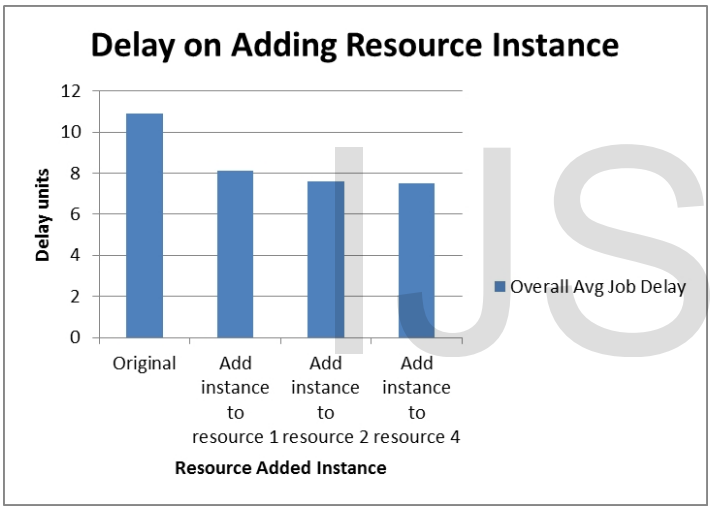

Fig.5 Adding Instances to Resources

On adding the instances to the resources, the above graph shows the results. We can see that with the original configuration the delay was high and the minimum delay was found when we added an instance to the Resource 4. Thus the bottleneck having more impact is the resource 4. And better performance in terms of less total average delay can be archived if we add one more instance of the resource 4.

REFERENCES

[1] Hu, Ming (1998) Operating system simulation (OSS) in Java: the system architecture. (Thesis) http://spectrum.library.concordia.ca

[2] Seltzer, M P. Chen and J outerhout, 1990.Disk scheduling revisited in USENIX. Winter technical conference.

[3] Sabrina, F.C.D, Nguyen, S.Jha, D. Platt and F. Safaei, 2005. Processing resources scheduling in programmable networks. Computer community, 28:676-687.

[4] E.O. Oyetunji, A. E. Oluleye, Research Journal of Information Technology 1(1): 22-26, 2009, "Performance Assessment of Some CPU Scheduling Algorithms"