ENHANCEMENT OF MULTIPROCESSING ENVIRONMENT FOR THE E POULTRY SYSTEM

Ahsan Raza Sattar[1]*, Muhammad Younus Javed[2], Tasleem Mustafa[1] and Sugra Bibi[1]
[1]Departmet of Computer Science, University of Agriculture, Faisalabad
[2]Department of Computer Engineering, College of E&ME, NUST, Peshawar Road, Rawalpindi
*Corresponding author's e-mail: uaf_raza@hotmail.com

The complete precision poultry system works over the number of multiple logical processes used inside the processors. The efficiency and accuracy can be enhanced by simply optimization of these processes used in processors. This high speed computing requires multitasking and parallel processing to increase the throughput of the system and optimizes use of system resources. In the multitasking environment, when a great number of processes execute in parallel, there may be some data structures like semaphores which are common or shared between different numbers of processes. Due to this sharing, different critical sections are produced. The research work presented in this paper resolves the critical section problem of multiple processes using semaphore implementation.

The development of a simulator for the resolution of critical section problem is presented. It is a comprehensive tool which runs a simulation in real time and generates useful data for evaluation. A user friendly and mouse-driven GUI has also been integrated. The developed system has been put through extensive experiments. Results are taken using different sets of processes for different number of common variables on a number of processes. The evaluation results are very promising and could be used to further enhance performance of multi-user and multitasking operating systems under different processors to deal with common semaphores.

Keywords: Multiprocessing, E poultry, precision, optimization, simulator

INTRODUCTION

One of the major objectives of multiprogramming and parallel processing is to maximize the resource utilization which is achieved by sharing system resources among multiple users/processes (Silberschatz, 2002). Shared memory solves the inter-processes communication problem. Concurrent access to shared data may result in data inconsistency. Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes (Ferleson and Rodolph, 1990). The overlapping portion of each process where the shared variables are being accessed is known as Critical Section (CS). A solution to the critical section problem must satisfy the following three requirements (Silberschatz, 2002): mutual exclusion**,** progress and bounded waiting.

This work focuses on the development of a simulator for resolution of CS problem for multiple common semaphores using multiple processes. This simulator can be used for measuring the performance of the system with different number of processors in terms of throughput of the system. It simulates concurrent execution of various processes on different data sets. It demonstrates managing of a number of queues and scheduling of processes for different processors. It not only successfully resolves CS problem but also provides in-depth understanding of parallel processing that avoids race condition and handles multiple common semaphores in the multiprocessor environment. Performance in a simulated scenario where sender is constrained by the multiprocessor that if two paths are used for processing, the lower quality (i.e., higher loss rate) path degrades overall throughput of an system (Lyengar, 2004).

Simulator Design

The system is designed to mimic the dynamic behavior of the system over time. It runs self driven simulation and uses a synthetic workload that is artificially generated to resemble the expected conditions in the modeled system. The data of processes (i.e process name, process priority number, processes burst time, processes arrival time) is taken as input from the user. It simulates the creation of processes, their assembling in different waiting queues, concurrent execution and their termination. A user friendly simulator generates useful data that represents the behavior of the system to handle multiple semaphores for different tasks of the user. The data is recorded in a data file for analysis and archival purposes. The Graphical User Interface (GUI) of the simulator provides the user an opportunity to either get a pictorial view of concurrent execution of processes related to different common variables or present a console-based view of the running processes. The GUI displays the real time view of the waiting queues related to different common variables and number of tasks used for the simulation. An entry of the number of processes along with their arrival time, priority and burst time is made from the keyboard or from the file. These features make the system attractive for obtaining experimental results and to demonstrate behavior of the system as the processes increase or decrease.

Software Development

The Critical Section Solution Simulator (CSSS) has been developed with a view to develop a software tool which can be used to study concurrent execution and resolution of CS problem for multiple common semaphores using a number of processors. CSSS is, in fact, a comprehensive software package which runs a simulation in real time by creating an interactive environment due to powerful facility of its GUI. The generated data and results provide complete understanding of its design and various parameters related to concurrency.

Software design strategy is function-oriented and modular in nature. The CSSS is built to run on a PC under WINDOWS environment using Java language. First of all, user selects the file to get the stored data which he/she wants to use for the simulation. In the initialization portion of the system, user gives the values of different processes such as processes name, process burst time (time to run that process), process priority token, process common file to share etc. Initialization of processes takes place against every common semaphore. At the end user can press the start button to make this simulator in working position.

Simulator and Gui Modules

A number of simulation modules have been utilized in the design and development of CSSS. The first module uses one waiting queue and one semaphore.

Two queues along with semaphores are used in the second module. Third module is more complex which uses three waiting queues along with their semaphores. The graphical user interface for the multiprocessor Simulator application consists of five major components. These components are the reader/writer queue information, set control panel (start, save and load options), component panel, title bar and status panel.

The title bar consists of a monogram and the caption of the multiprocess simulator. The control panel, just present in the bottom left corner of the screen, is a panel of save, load and start buttons which can be depressed to select functions. All control panel buttons duplicate functionality that is available in the menu bar, but provides more convenient access to these functions. The result panel is the output information area for the multiprocess simulator, in the bottom right of the screen, just adjacent to the control panel. It is made up of a text boxes and label boxes, where multiprocess simulator components can be placed on the middle and upper portion. The component panel is just below the title bar. It provides an easy interface for selecting options and for data entering for multiprocess simulator components. Finally, the status panel is a message board at the bottom of the window where the multiprocess simulator application relays important information to the user regarding errors, hints for operation, and the basic status of the multiprocess simulator methods. To overcome the need for busy waiting, the definition of the wait and signal semaphore operations has been modified. When a process executes the wait operation and finds that the semaphore value is not positive, it must wait. However, rather than busy waiting, the process can block itself. The block operation places a process into a waiting queue associated with the semaphore and the state of the process is switched to the waiting state. Then control is transferred to the CPU scheduler which selects another process to execute.

A process that is blocked, waiting on a semaphore, should be restarted when some other process executes a signal operation. The process is restarted by a wakeup operation which changes the process from the waiting state to the ready state. The process is then placed in the ready queue (The CPU may or may not be switched from the running process to the newly ready process depending on the CPU-scheduling algorithm).

RESULTS AND DISCUSSION

The developed system runs a simulation in the real time and generates useful results to see effectiveness and efficiency of the system. The following processes have been used to obtain results through simulation, as shown in Table 1. And the detailed results are shown in Table 2.

Table 1. Processes along with their initial values used for simulation

1	Rdr1	5000	1000
2	Rdr2	5000	2000
3	Rdr3	5000	1000
4	Rdr4	5000	2000
5	Rdr5	5000	1000
6	Rdr6	5000	1000
7	Rdr1	5000	2000
8	Rdr1	5000	1000
9	Rdr1	5000	2000
10	Rdr1	5000	1000
11	Rdr1	5000	1000
12	Rdr1	5000	2000
13	Wtr1	5000	1000
14	Wtr2	5000	2000
15	Wtr3	5000	1000
16	Wtr4	5000	1000
17	Wtr5	5000	2000
18	Wtr6	5000	1000
19	Wtr7	5000	2000
20	Wtr8	5000	1000
21	Wtr9	5000	1000
22	Wtr10	5000	2000
23	Wtr11	5000	1000
24	Wtr12	5000	2000

Single queue along with single semaphore

Throughput of the system is less because only single queue is used for the simulation of mutual exclusion. In this case, throughput of the system totally depends upon the nature and burst time of the processes. Average Waiting Time (AWT) of the processes increases, as shown in Figure 1.

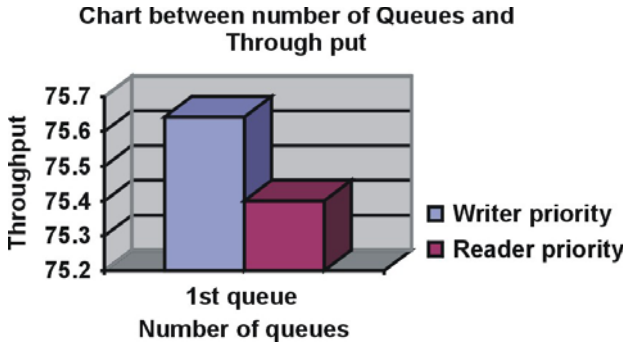

Fig. 1. Throughput with one queue

Two queues along with their semaphores

When two queues are used, half of the processes will go to one queue and the remaining half of the processes will use 2nd queue for the waiting purpose and are then loaded into the ready queues respectively. Overall throughput of the system is 119.43 processes per minute when writers have priority over reader processes and 129.64 processes per minute when readers have priority over writer processes. All these processes execute as per their burst timings resulting different Average Waiting Times (AWTs) and throughputs. The AWT of the processes is low and less starvation of low priority processes has been observed. Throughput increases as shown in Figure 2. The burst time and throughput are shown in Table 2.

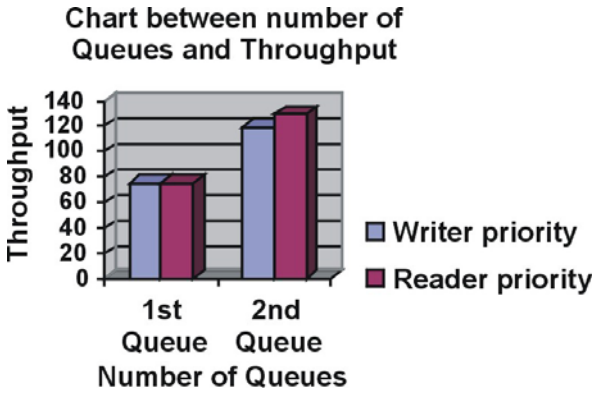

Fig. 2. Throughput with two queues

Three queues along with their semaphores

When three queues are used, one third processes will be processed by each queue and then the overall throughput of the system is 158.36 processes per minute when writers have priority over reader processes. It is 174.92 processes per minute when readers have priority over writer process. These processes execute as per their burst timing, resulting different AWTs and throughputs. The AWT of the processes is low and it provides less starvation of low priority processes. Table 2 shows that the throughput increases from 119.43 to 158.38 processes per minute when writers have priority. Similarly, it reaches to 174.92 from 129.64 processes per minute when readers have priority over writer processes. Detailed results are shown in Table 2 and Figure 3 provides throughput with three queues.

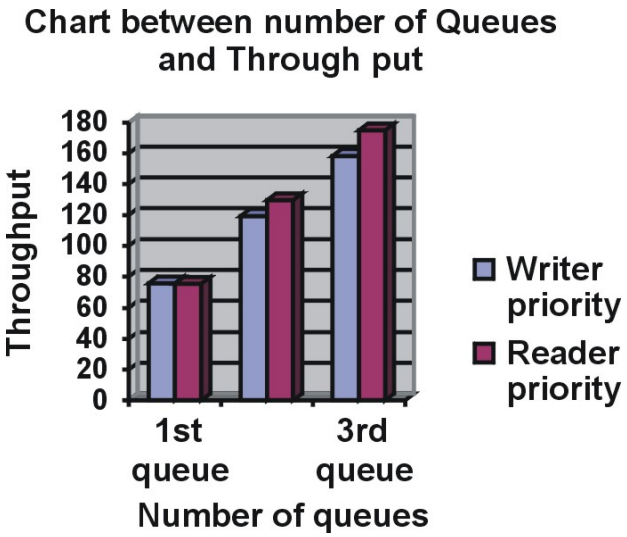

Fig. 3. Throughput with three queues

CONCLUSIONS

Parallel processing is the technique which can be used for the high speed computing in a multi-user environment. Parallel processing results in increasing the throughput of the system and optimizes use of system resources. In the multitasking environment, when a large number of processors execute in parallel, there may be some resources which are common or shared between different numbers of processes. Due to this sharing different critical sections are produced. These critical sessions provide inconsistency in a multiuser and multitasking environment. Critical session problem can be solved using different algorithms. CSSS guards against the race condition. The developed system guarantees synchronization of processes to resolve CS problems of using semaphores.

Table 2. Detailed results of simulator

Sr. No.	Module No.	Queues	Priority	Average waiting time (m secs)		CPU execution time (m sec)		Total time (sec)	Throughput
				Wrt	Rdr	Wrt	Rdr		
1	1	1	Wrt	7501	17123	17024	2013	19037	75.642
2	1	1	Rdr	9261	0	17025	2073	19098	75.400
3	2	2	Wrt	3410	8518	9023	3034	12057	119.432
4	2	2	Rdr	5674	2	9034	2073	11107	129.648
5	3	3	Wrt	1988	5924	6018	3074	9092	158.380
6	3	3	Rdr	4141	1	6029	2203	8232	174.927

The CSSS has been developed in such a way that when one process is executing in its critical section, it does not allow any other process to execute in its critical section. For this purpose, separate queues are created and maintained by the system and it blocks other processes in the appropriate queues when one or more than one process are executing in their critical sections. The major feature of the CSSS is to provide mutual exclusion for process synchronization and to resolve CS problem and finally this enhancement is implemented in a small model and then handed over to Poultry Department, University of Agriculture for further experiments and results.

REFERENCES

Caro, A.L., J.R. Lyengar, P.D. Amer, G.J. Heinz and R. Stewart. 2002. Using SCTP Multihoming for Fault Tolerance and Load Balancing, ACM SIGCOMM Computer Communication Review 32(3).

Ferleson, D. and L. Rodolph. 1990. Mapping and Scheduling in a shered Parallel Environment Using Distributed Hierarchical Control. Proceedings of the International Conference on Parallel Processing.

Lyengar, J.R., K.C. Shah, P.D. Amer and R. Stewart. 2004. Concurrent Multipath Transfer Using SCTP Multihoming in SPECTS 2004. Protocol Engg. Lab., CIS Dept., University of Delaware. pp.9.

Lyengar, J.R., P.D. Amer and R. Stewart. 2005. Receive Buffer Blocking in Concurrent Multipath Transfer. Global Telecommunications Conference, GLOBECOM '05, 1: pp. 6.

Lyengar, J.R., P.D. Amer and R. Stewart. 2007. Performance Implications of a Bounded Receive Buffer in Concurrent Multipath Transfer. Computer Communications 30(4): 818-829.

Silberschatz, A., P.B. Galvin and G. Gagne. 2002. Operating System Concepts, 6th Edition. John Wiley & Sons.

Tanenbaum, A.S. 1996. Modern Operating Systems, 2nd Edition. Prentice-Hall.

Tanenbaum, A.S. and A.S. Woodhull. 1998. Operating Systems, 2nd Edition. Prentice-Hall.