Implementation and Experimentation of Producer-Consumer Synchronization Problem

Article *in* International Journal of Computer Applications · January 2011

DOI: 10.5120/1823-2398 · Source: DOAJ

4 authors, including:

Syed Nasir Mehmood Shah
Universiti Teknologi PETRONAS
34 PUBLICATIONS **113** CITATIONS
SEE PROFILE

Nazleeni Haron
Universiti Teknologi PETRONAS
53 PUBLICATIONS **153** CITATIONS
SEE PROFILE

Muhammad Younus Javed
HITEC University, Taxila, Pakistan
298 PUBLICATIONS **1,978** CITATIONS
SEE PROFILE

Implementation and Experimentation of Producer-Consumer Synchronization Problem

Syed Nasir Mehmood, Nazleeni Haron
Universiti Teknologi PETRONAS
Bandar Seri Iskandar
Perak, Malaysia

Vaqar Akhtar, Younus Javed,
National University of Science & Technology
College of Electrical & Mechanical Engineering
Pakistan

ABSTRACT
This paper presents the design and implementation of a simulator that allows user to study producer-consumer synchronization problem in three different contexts: simple producer-consumer problem, producer-consumer problem in a single processor system and producer-consumer problem in multi-processing system. The main contribution of our work is the facility to run the simulation in a multi-processors environment. This has been deemed appropriate since most of the current applications are distributed in nature and run in multi-processing environment. Additionally, the features offered by the simulator enables user to analyze and evaluate various synchronization problems in a repeatable and controllable environment. The experimental results are presented to show the viability of our proposed simulator.

General Terms
Simulation, Synchronization

Keywords
Bounded Buffer Problem, Producer Consumer, Single Processor, Multi- Processor.

1. INTRODUCTION
Producer-consumer problem is one classical example of bounder buffer synchronization problems. The synchronization is needed in order to ensure that the producer stops producing when the buffer is full and the consumer stops removing items from the buffer if it is empty. The variations of producer-consumer problem can be implemented in different type of applications [1], [2], [3], [4] and can be run on both single and multi-processors systems [5]. Due to its importance, many researchers have studied this problem and normally simulation is used in order to visualize the behavior of the problem [2], [6]. However, most of these simulators only allow user to experiment in the context of a single producer and a single consumer. To address this limitation, we propose a simulator that allows user to study producer-consumer synchronization problem in three different contexts such as simple producer-consumer problem, producer-consumer problem in a single processor system and producer-consumer problem in multi-processing system. This has been deemed appropriate since most of the current applications are distributed in nature and run in multi-processing environment. In addition, users can carry out comparative evaluation among different contexts using the simulator as to gain insights on the best model to use in certain situations. The simulator has been developed as a comprehensive software package which can run self-driven simulation, generates useful data and provides a user-friendly environment. Software design strategy is function-oriented and design is modular in nature. The system is designed to run on Windows and is written using Java.

This paper presents the implementation and experimental results of the proposed simulator for producer-consumer problem in three aforementioned contexts. In the next section, we describe the general producer-consumer problem. Section 3 presents the models of the three different contexts. Section 4 describes their implementations and section 5 provides the experimental results for each context and outlines some discussions. Section 6 concludes the paper.

2. PRODUCER-CONSUMER PROBLEM
In producer-consumer problem, there are two types of process involved: producer and consumer who are sharing a fixed size buffer [3]. The role of producer is to put the item one a time in the buffer and the role of consumer is to retrieve the item from the buffer. The problem is to make sure that the producer will not try to add the item once the buffer is full and that the consumer will not try to remove the item from an empty buffer [3].

According to [2], the normal behavior of this problem are (i) Random arrival of petitions to put item in the buffer; (ii) One consumer that immediately after getting an information from the buffer intends to get another one; (iii) Buffer of defined and finite size; (iv) Consumer waits when the buffer is empty; (v) Producer waits when the buffer is full; (vi) Second and subsequent productions wait when the customer is writing or held up.

3. PRODUCER-CONSUMER MODELS
This section presents the different models of the producer-consumer problem that can be simulated by the user.

3.1 Simple Producer-Consumer Problem
This problem describes one producer and one consumer sharing the same finite buffer. Producer produces items and place items in the buffer while consumer consumes items from the buffer.

3.2 Producer-Consumer Problem in Single Processor Environment
This model is the advanced form of the first model. In the first model, producer produces and places items in the buffer at random basis. Consumer simply consumes items from the buffer without acknowledging whether it is meant for them or not. But in this model, producer is producing items for the specific consumer and only that consumer is given access to that item. When that consumer is free, it will consume the items from the buffer. This module of the simulator works for single processor environment.

3.3 Producer-Consumer Problem in Multi-Processor Environment

This model is designed for multi-processor environment. Parallel processing technique has been employed in this module to enable *N* number of producers and consumers executing at the same time. This is to ensure that resources are fully optimized. In this model, producers can produce items/data for specific consumers and only that consumer can consume those items/data.

4. SIMULATOR DESIGN AND IMPLEMENTATION

4.1 Class Diagram

This section provides details on fundamental classes of the simulator, which are the building blocks of the simulator. The relationship among the classes is depicted in Figure 1.

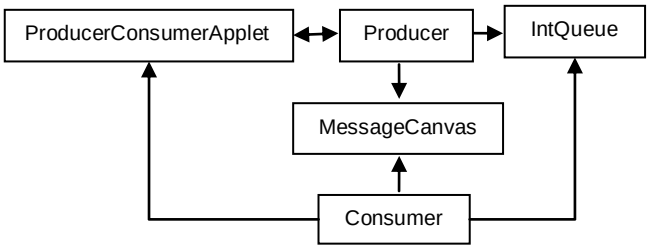

Fig 1: Simulator Classes

4.1.1 ProducerConsumerApplet Class

This is the main class of a Java applet file for the simulator. The GUI of this applet contains three parts: animation canvas, message canvas and a button panel. The animation canvas is where the producer/consumer animation is displayed. The message canvas is where the status of producers and consumers are displayed. The button panel has six basic buttons for the users to use while simulating the problems. The user also can select the number of the buffer, the number of producers and the number of consumers. Unless the user selects different values, the default number of buffer is 10, and default number of the producers and consumers are two.

4.1.2 IntQueue Class

This class contains the control of execution of the producer and consumer. It provides the bounded buffer of size 10. Producer and consumer call the enqueue and dequeue methods of this class to access the shared buffer. Enqueue method is used to place items in the buffer while dequeue is to remove the items from the buffer.

4.1.3 Producer Class

Producer's main activity is to call the enqueue method in IntQueue class. This class is inherited from the thread class and the run method of this class performs the major functionality. This method will start a loop which will continuously produce the items. The item value is produced by calling the built in random method. Then this method calls the enqueue method of the IntQueue class. This will place the produced item on the queue. Then this method also calls the updateTime method of MessageCanvas class to update the time on main canvas. This method also appends the produced item in the message window.

4.1.4 Consumer Class

Consumer's main activity is to call the dequeue method in IntQueue class. This class is also inherited from the thread class and the run method of this class performs the major functionality. This method will start a loop which will continuously check the need for consuming by calling the getConsmerSel method. If it needs consuming operation then it will call the dequeue method of IntQueue class. Then this method also appends message in the message window that an item is consumed and also display on the canvas board.

4.1.5 MessageCanvas Class

This class provides message canvas for the applet GUI. It will print the statuses of producers and consumers on the GUI. All of the other classes use the methods of this class to display the messages on the canvas.

4.2 Simulator Options and Features

The simulator allows user to explore the three different models visually through the use of 2D graphics. The menu system will allow the user to choose the desired options. A user will be provided with the following options and features of the simulator:

- Producer Process.
- Consumer Process
- Ability to change number of CPUs
- Ability to change the size of the buffer. (Maximum of: 20 slots and a minimum of: 5 slot)
- Ability to adjust the speed of the simulation.
- Ability to change the status of each of the processes even while the simulation is executing.
- Nine buttons are provided: Start, Stop, Pause, Continue, Faster, Slower and Save:
  - Start Button: Will activate the simulation and start the process(s) execution.
  - Stop Button: Will completely stop the simulation. This should happen until all the locks are released. Processes should be terminated naturally.
  - Pause Button: Will pause the simulation
  - Continue Button: Will resume the simulation after any pause
  - Faster Button: Will increase the speed of the simulation
  - Slower Button: Will decrease the speed of the simulation
  - Save Button: Will save the performance parameters in word/excel format
  - Pause Producer/Consumer : Will pause any chosen producer or customer
  - Resume Producer/Consumer : Will resume the simulation of chosen producer or consumer
- Messages to the user:
  - When the producer is the only process active in the simulation. (i.e. when the buffer is filled completely, a message will be displayed informing the user that a deadlock has occurred and that the Consumer must be added for the deadlock to be removed).
  - When the Consumer is the only process active in the simulation. (i.e. when the buffer is completely empty, a message will be displayed informing the user that a deadlock has occurred and that the Producer will be added for the deadlock to be removed).
  - The simulation cannot start unless a buffer exists.

4.3 Modes of Operations

The simulator can be run in 3 different modes as follows:

4.3.1 Simple Producer Consumer

In this mode, the simulation uses only one CPU and the rest of the parameters such as Buffer Size, Number of Producers and Consumers must be given by the user. The user interface for this mode is as shown in Figure 2.

Figure 2 displays a scenario where a user selects buffer size ten, chooses three producers and consumers respectively and clicks on START button. Producers P1, P2 and P3 start producing items and consumers C1, C2 and C3 start consuming items produced by the producers. A circular buffer has been maintained. Two logical pointers have been shown, *In* and *Out*. *In* pointer points to the position where item is to be placed where *Out* pointer points to the position from where item is to be removed. Simulator also shows the status of buffer during execution. When buffer becomes full it gives a message "Buffer Status: Full" and in case of empty buffer it gives a message "Buffer Status: Empty".

User may also interact with the simulation at run time as to attain information such as buffer size, empty slots, available items, status of each producer and each consumer. A user may pause or resume any producer or any consumer at any instant during execution of simulator. Initially all producers and consumers are in sleeping state. When buffer becomes full, then producer moves in waiting state. When buffer becomes empty then it is the consumer's turn to be in waiting state. Sleeping state and waiting states are highlighted with yellow color. Active/running states of producers and consumers are represented with blue color. Pause states of producers/consumers are shown with red color.

4.3.2 Producer-Consumer Problem in Single Processor Environment

All options as discussed in the first mode are also available in this mode. However, additional information is available in this mode and as shown in Figure 3. In this mode, the status for each producer is added with information on the corresponding consumer who is expecting the item. Figure 3 illustrates a scenario where two producers (P1 and P2) are producing items for two consumers (C1 and C2) with a shared buffer of size ten.

The simulator will also display the information on the producer-consumer activities throughout the simulation. User may save the complete history for later analytical use in Word or Excel format. Figure 4 shows example of simulation activities saved in Word format. In addition, the simulator also captures the elapsed time that can be used to measure the throughput of the system.

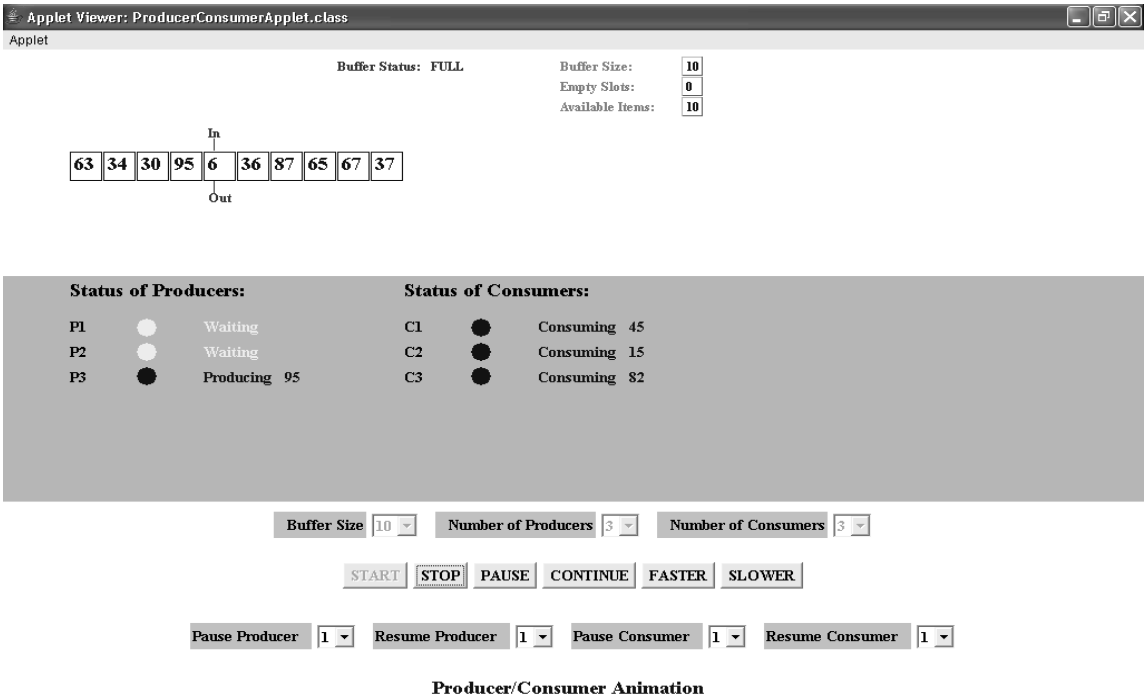

Fig 2: Interface for Simple Producer-Consumer Simulation

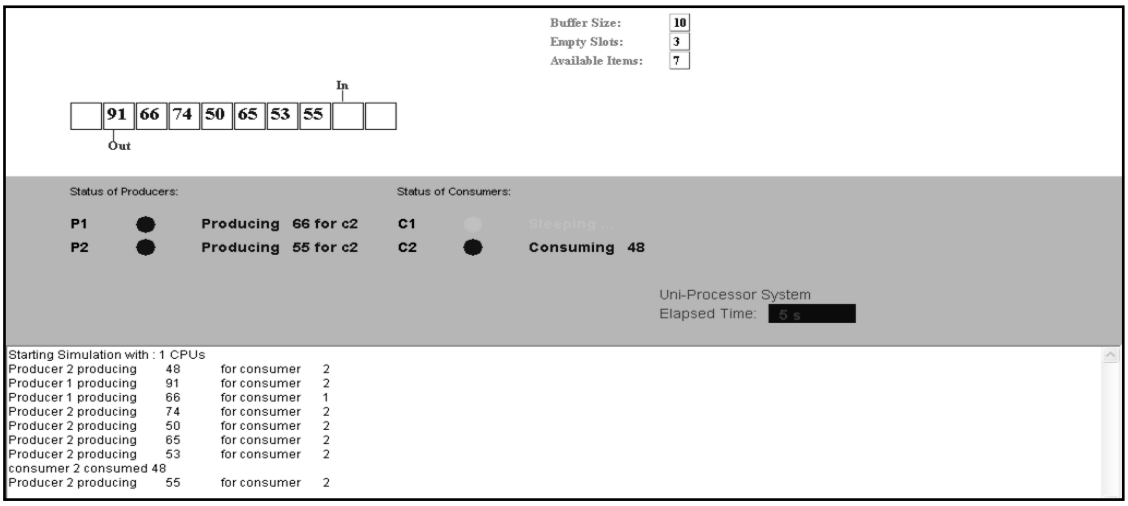

Fig 3: Interface for Producer-Consumer Simulation in a Single Environment

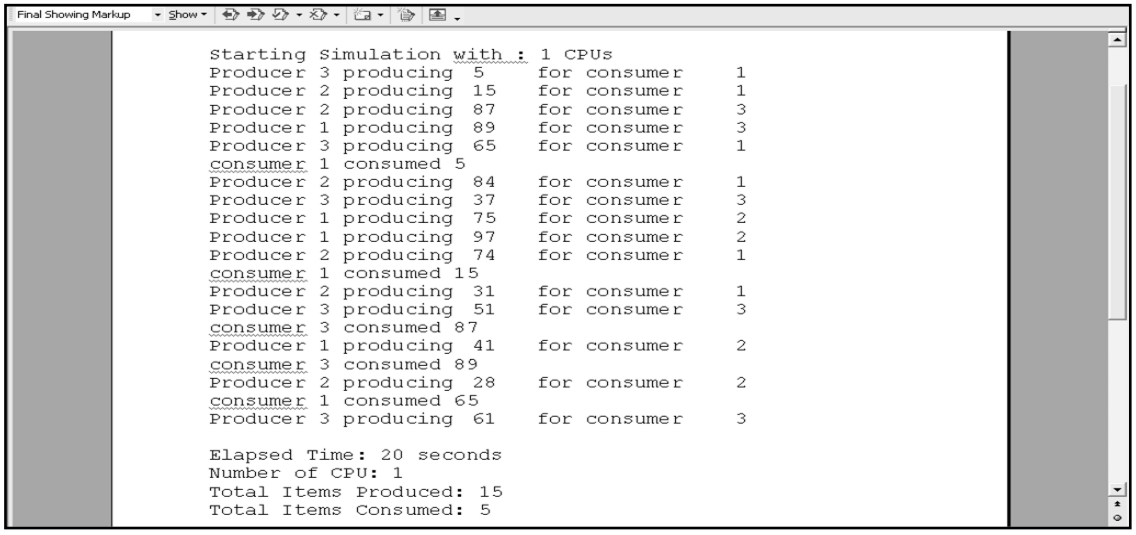

Fig 1: Sample of simulation activities in Microsoft Word format

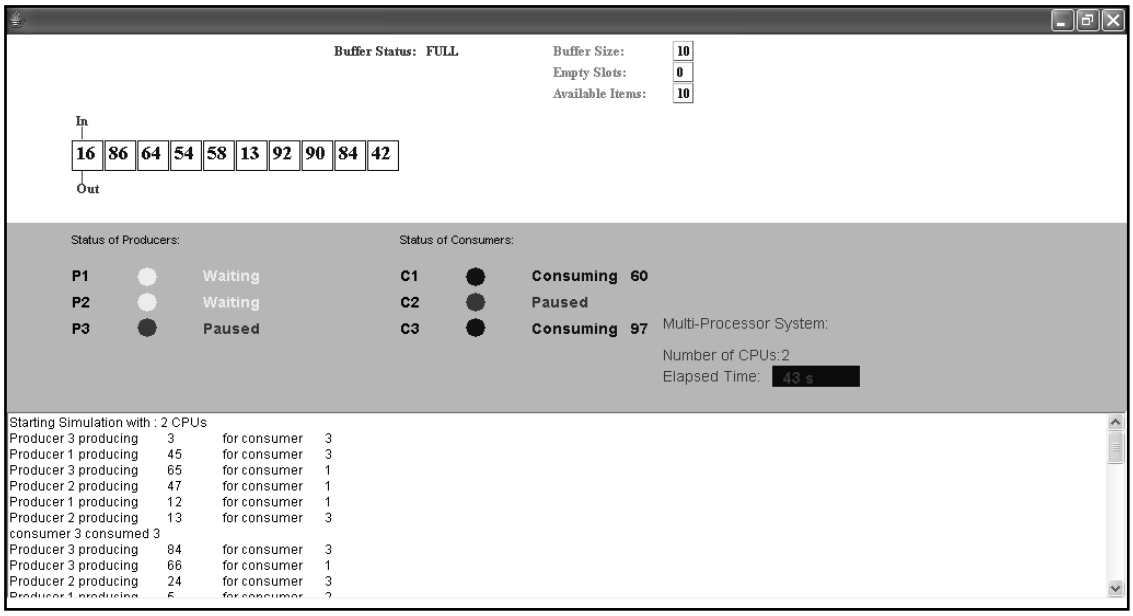

Fig 5: Interface for Producer-Consumer Simulation in Multi-Processor Environment

4.3.3 Producer-Consumer Problem in Multi-Processor Environment

In this mode, a user is given option to select number of CPUs in addition to pre-mentioned options. A user may run simulator with different number of CPUs and can watch the performance of implemented producer consumer algorithm. A user is given an option "SAVE" to save the simulation history. A user may record the performance parameters in Word format as well in Excel format.

Figure 5 displays a working simulator running this mode for a scenario of three producers and consumers using two CPUs.

5. EXPERIMENTAL RESULTS

This section presents the experiments that have been carried out to evaluate the efficiency of the simulator in simulating the different contexts of producer-consumer synchronization problem. We have performed a series of experiments by varying the input parameters as to study the following:

5.1.1 Relationship between Simulator Output and Elapsed Time

This experiment is aimed at quantifying the effect of elapsed time on the number of items produced and consumed. It is also meant to see the effects of elapsed time for the producer-consumer problem in single processor and multi-processor environments. For a single processor environment, we run three set of experiments with the following constant input parameters: three producers, three consumers, buffer size 10 and one CPU. We varied the elapsed time for each experiment. As for multi-processors environment, we have kept the same value of parameters except for number of CPUs. Figure 6 depicts the sample result of this experiment for three different elapsed time; 20, 40 and 60 seconds. The results attest that elapsed time have significant effects of number of items produced and consumed in both type of environments.

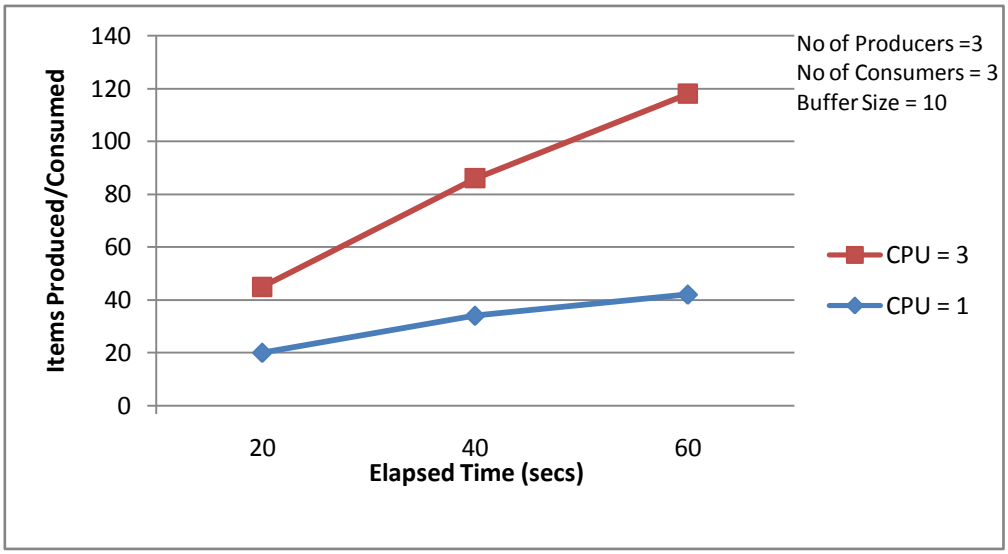

Fig 6: Simulation results for relationship between items produced/consumed and elapsed time for single and multi-processors environments

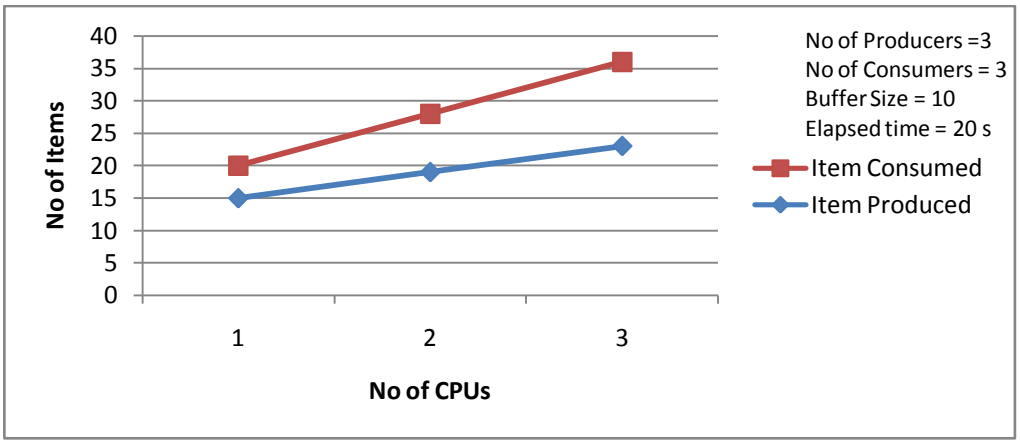

Fig 7: Simulation results for relationship between items produced/ consumed and number of CPUs

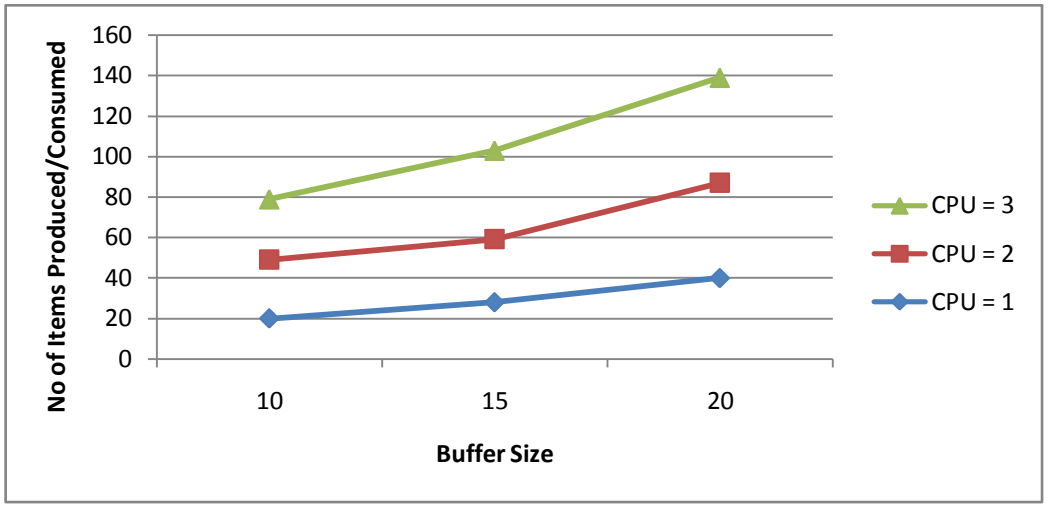

Fig 8: Simulation results for relationship between items produced/ consumed and buffer size running in three different environments

5.1.2 Relationship between Simulator Output and Number of CPUs

To study this relationship, we performed three sets of experiments with the following constant input parameters: three producers, three consumers, buffer size 10 and elapsed time of 20 seconds. Each set of experiment has been run with different number of CPUs. Output of the simulator is measured in terms of number of items produced and items consumed. Figure 7 shows a sample result for this experiment. The results show that the output of the system increases as the number of CPUs increases.

5.1.3 Relationship between Simulator Output and Buffer Size

The performance of the simulator with regards to buffer size was computed via this experiment. In this case, the other entire parameters are kept constant and only buffer size is varied. Three sets of experiment have been conducted for three types of environment; single (set number of CPU to 1) and multiprocessor (set number of CPU to 2 and 3). The results for this experiment are depicted in Figure 8 with three different buffer sizes : 10, 15 and 20 . The results demonstrate that for all environments, simulator output (number of items produced and consumed) increases as the buffer size increases. They are not linearly proportional although the buffer size was increased linearly. Significant increased can be seen when buffer size was increased from 15 to 20.

6. CONCLUSION

This simulator provides a facility for the user to experiment with three different contexts such as simple producer-consumer problem, producer-consumer problem in a single processor system and producer-consumer problem in multi-processing system. This has been deemed appropriate since most of the current applications are distributed in nature and run in multi-processing environment. In addition, users can carry out comparative evaluation among different contexts in a repeatable and controllable environment as to gain insight on the best model to use in certain situations. All developed modules of the simulator guarantee synchronization of processes and satisfy necessary requirements to provide solution to the critical section problem.

This simulator has been designed to run self-driven simulations which are intrinsically limited in accuracy. It is because the input data on which the simulation runs is generated artificially to model the target system. As such, we plan to incorporate the trace-driven simulation in the existing simulator for more accurate results. Trace-driven uses as input, a trace of actual events collected and recorded on a real system. Capability of running a trace-driven simulation can be incorporated in the existing system by developing a software module that can translate the trace of events recorded on a real system, into the format of input data file. The input data file so generated then can be used to run the simulation.

7. REFERENCES

[1] Stefano, A. D., Bello, L. L., Santoro, C.1997. Synchronous Producer-consumer transactions for real-time distributed process control. In Proceedings of the IEEE International Workshop on Factory Communication Systems.

[2] Juiz, C., Puigjaner, R. 1995. Improved performance model of a real-time software element: the producer-consumer. In Proceedings of the Second International Workshop on Real-Time Computing Systems and Applications.

[3] Zhang, Y., Zhang, J., Zhang, D. 2009. Implementing and testing Producer-consumer problem using aspect-oriented programming. In Proceedings of the Fifth International Conference on Information Assurance and Security.

[4] Shen, C. 2000. Discrete-event simulation on the Internet and the web. Future Generation Computer Systems, vol. 17, 187-196.

[5] Hilzer, Jr., R. C. 1992. Synchronization of the Producer/Consumer problem using semaphores, monitors, and the Ada Rendezvous. Operating Systems Review, vol. 26, 31-39.

[6] Robbins, S. 2000. Experimentation with bounded buffer synchronization. In Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education.