Implementation of New Improved Round Robin (NIRR) CPU Scheduling Algorithm Using Discrete Event Simulation

Chang Jan Voon and Idawaty Ahmad*
Department of Communication Technology and Network,
Faculty of Computer Science and Information Technology,
University Putra Malaysia, 43400 UPM,
Serdang, Selangor DE, Malaysia
*Email: idawaty [AT] upm.edu.my

Abstract— **Round Robin scheduling algorithm is the most widely used scheduling algorithm because of its simplicity and fairness [1]. However it has higher context switching, larger response time, larger waiting time, larger turnaround time, and lower throughput. Reference [1] proposed a new algorithm, called New Improved Round Robin (NIRR) to enhance the Round Robin scheduling algorithm. The proposed NIRR algorithm has shown improvement over the traditional Round Robin algorithm. However the lack of details of general NIRR simulation model is a clear limitation for the further improvement of the algorithm. The main objective of this research is to validate the NIRR algorithm by developing a comprehensive simulation model using Discrete Event Simulation (DES). An NIRR simulator is deployed and is validated by ensuring the output data closely resemble the output data published by [1]. Extensive experiments were done to validate the developed NIRR simulator. The success of the developed NIRR simulator was proven by the generated results.**

Keywords-CPU scheduling algorithm; Round Robin; Discrete Event Simulation (DES)

I. INTRODUCTION

An operating system is essential system software of a computer system. It manages the computer resources and provides services to the application programs. It acts as the intermediary between the application programs and computer resources. In the modern computer system, multi-tasking operating systems are used in which more than one application programs can run concurrently. It is the operating system responsibility to schedule tasks for efficient use of system resources.

CPU is one of the most important resources to be shared among the application programs. Therefore the CPU scheduler in the operating system plays an important role in time-sharing and dividing the processor time efficiently between multiple tasks or processes. CPU scheduling algorithms determines which task runs when there are multiple tasks ready to execute.

In general, there is no "best" scheduling algorithm. The choice of scheduling algorithm is based on the type of system used and the requirement. According to [1], Round Robin (RR) scheduling algorithm is the most widely used scheduling algorithm because of its simplicity and fairness. However it has higher context switching, larger response time, larger waiting time, larger turnaround time, and lower throughput.

A. *Problem Statement*

Reference [1] proposed a new algorithm, called New Improved Round Robin (NIRR) to enhance the existing Round Robin scheduling algorithm. The NIRR algorithm determines the time slice dynamically, taken from the average of the burst time of the tasks to be executed by the CPU. The NIRR algorithm is simulated and has shown improvement over the traditional RR algorithm. However the lack of details of general NIRR simulation model is a clear limitation for the further improvement of the algorithm.

B. *Research Objective*

The main objective of this research is to validate the NIRR algorithm by developing a comprehensive simulation model using Discrete Event Simulation (DES). An NIRR simulator is deployed and is validated by ensuring the output data closely resemble the output data published by [1]. The proposed simulation model in this research may led researcher to greater understanding of the algorithm and provides a platform for future investigations in NIRR scheduling algorithm.

II. LITERATURE REVIEW

In a multitasking operating system, the CPU scheduler plays an important role in time-sharing and dividing the processor time efficiently between multiple tasks or processes. CPU scheduler is often implemented so to keep the CPU busy and to maximize the CPU utilization. When the CPU becomes idle, the CPU scheduler will select one of the ready tasks in the queue to be executed. In normal circumstances, there are more than one ready tasks lined up in the queue waiting for the CPU. The rules to select which task to be executed first are determined by the CPU scheduling algorithm. The following section describes some of the fundamental CPU scheduling algorithms.

A. Fundamental CPU Scheduling Algorithms

First Come First Serve (FCFS) is the simplest algorithm. The CPU is allocated to the tasks based on their time of arrival to the ready queue. The task that arrived first will be executed first. The next task to be executed is at the head of the ready queue while newly arrived task will be placed at the tail of the ready queue. When the CPU is free, the task at the head of the queue will be executed and removed from the ready queue. This algorithm is very simple and scheduling overhead is low. However tasks with long processing time may hold up the CPU significantly. The average waiting time, turnaround time and response time are quite long.

Shortest Job First (SJF) algorithm requires the knowledge of all processing time or burst time of the tasks. The task with the least burst time will be scheduled first. Therefore the tasks will be arranged according to the burst time in the ready queue, where the shortest task is at the head of the queue. This algorithm is proven to be fastest algorithm, with short average waiting time, turnaround time and response time. However, it suffers from a situation called starvation, where tasks with long burst time never get to run when a lot of shorter tasks arrived and admitted to the ready queue.

In Priority Scheduling (PS) algorithm, each task is assigned with a priority number. It schedules the task with the highest priority first. If some of the tasks have same priority number, FCFS will apply for these tasks. The average waiting time, turnaround time and response time depend on the priority of the tasks. This algorithm may also suffer from starvation where lower priority tasks never get to run when large number of higher priority tasks queuing for the CPU.

Round Robin (RR) algorithm is a fair time-sharing scheduling algorithm. Each task gets a small slice of CPU time, called time slice. Newly arrived task is admitted to the tail of the ready queue based on FCFS. RR schedules the task from the head of the queue to the CPU for the duration of a time slice. The task is preempted when its time slice expires. The scheduler will then allocate the CPU to the next task in the head of ready queue, while the unfinished task is moved to the tail of the ready queue. RR algorithm is starvation-free since each task gets a small slice of CPU time. However it has high context switching, and longer average waiting time, turnaround time and response time.

B. CPU Scheduling Criteria

Each CPU scheduling algorithm has its strengths and weaknesses. The properties of the algorithm need to be evaluated when selecting a suitable algorithm for a particular situation [13]. To compare the CPU scheduling algorithms, the following criteria are commonly used:

- Number of Context Switches: In some CPU scheduling algorithm, one task may need to be switched out of the CPU so that another task can run. When this happens, the state or context of the task is stored so that the execution can be resumed at the same point at later time. The process of storing and restoring the context of the preempted task is called context switch. Switching from one task to another has a cost in the system performance, and therefore should be minimized in the design of the scheduling algorithm.
- Throughput: It refers to the number of tasks the scheduler managed to complete per unit time. This is a measure of the productivity of the CPU, and therefore throughput should be maximized.
- CPU Utilization: This mean how busy is the CPU, ranging from 0 to 100 percent. Multi-tasking operating system is designed to execute more than one task and to maximize the utilization of the CPU.
- Turnaround Time: Turnaround time is a measure of the total time it takes for the CPU to complete the execution of a task. The time interval from the arrival time of a task until the completion time is the turnaround time. Turnaround time should be minimized.
- Waiting Time: This is the duration a ready task retained in the queue, waiting to access the system resources. The time interval from the task joins the queue till it leaves the queue is the waiting time. In the design of scheduling algorithm, it is desirable to minimize the waiting time of the tasks.
- Response Time: This is the amount of time the system takes to respond to a task requesting the access to CPU. It is measured from arrival time of a task until the task access to the CPU for the first time. In order to provide a responsive system, the response time should be minimized.
- Fairness: One of the main objectives of the scheduler is to allocate the resources to the tasks in a fair manner. In scheduling algorithm, fairness means each task is allocated fair share of resources so that no task perpetually lacks necessary resources, or the resources are allocated according to the priority and workload of the task.

C. Related Works on Round Robin Scheduling Algorithm

Generally, the performance of RR scheduling algorithm is directly impacted by the selection of time slice. If very large time slice is used, the performance of RR scheduling is similar to FCFS. On the other hand, selecting a very small time slice will incur high scheduling overhead, i.e. high number of context switches. Many researches focus on the mechanism to choose or compute the time slice in order to improve the performance and efficiency of RR scheduling.

Behera, Mohanty and Nayak [4] created an improvement of Round Robin scheduling algorithm, named as Dynamic Quantum with Readjusted Round Robin (DQRRR) algorithm. Firstly, the tasks in the ready queue are sorted in ascending order of the burst time. The median of the tasks burst time is used as the time slice. After the CPU executed the tasks for duration of the time slice, the uncompleted tasks and newly arrived tasks are admitted to the ready queue for the next round of execution cycle. The time slice is recalculated based on the

median of the tasks remaining burst time. This goes on until all the tasks are completely executed.

Lalit, Rajendra and Praveen [6] proposed new Round Robin scheduling algorithm that also sorts the tasks in ascending order of the burst time. However, it calculates the time slice using the average of the burst time of the tasks in the queue.

Noon, Kalakech and Kadry [12] uses only single queue in the newly improved Round Robin scheduling algorithm. Tasks are not sorted, and will be executed based on FCFS in RR fashion. The time slice is computed based on the average of the burst time of the tasks currently in the queue. The time slice is updated at the event of which the CPU is allocated to the next task at the head of the queue. At this instant, if the previous task is not completed, it will be added to the tail of the queue. The time slice is calculated based on tasks currently in the queue, and it changes every time the CPU executed a task.

Saeidi and Baktash [14] proposed a non-linear programming mathematical model to compute the optimum time slice for the Round Robin scheduling algorithm. The mathematical model is developed based on the number of tasks in the waiting queue and their respective burst time, with the objective to minimize the average waiting time of the tasks scheduled using Round Robin algorithm.

Rajput and Gupta [13] proposed a Round Robin scheduling algorithm combined with Priority Scheduling. It uses fixed time slice. Newly arrived tasks are allocated to the CPU in RR fashion for one time slice. The unfinished tasks are assigned with priority based on the remaining burst time. Task with shortest remaining burst time is assigned with highest priority. These tasks are then executed based on the assigned priorities.

Another variation of Round Robin algorithm, proposed by Dawood [5], which computes the time slice by first sum up the minimum and maximum burst time of the tasks in the queue and then multiply the summation by (80) percentage. The tasks are allocated to the CPU based on Shortest Job First in Round Robin fashion.

Soraj and Roy [15] presented a new algorithm that dynamically determines the time slice called smart time slice (STS). The tasks are arranged based on the burst time in which the task with smallest burst time will be given highest priority. There are two way to calculate the STS. When the number of tasks in the ready queue is odd number, the time slice takes the mid of the tasks burst time. Otherwise, the time slice is calculated based on the average of the tasks burst time.

Mishra and Khan [10] proposed an improvement in the Round Robin algorithm and name it Improved Round Robin (IRR). The time slice is a fixed value. The CPU executed each task for duration of one time slice. Before the task is preempted and the CPU is allocated to another task, it checks if the currently executed task's burst time is less than the time slice. If so, the CPU will continue executing the task until completion. Else the current unfinished task will be moved to the tail of ready queue and the CPU will be allocated to the next task at the head of ready queue.

Mishra and Rashid [11] further improved the Round Robin algorithm with varying time slice. The tasks are arranged in the ready queue in ascending order of the burst time. For every execution cycle, the CPU is allocated to the tasks in the ready queue using Round Robin fashion and the burst time of the first task in the head of the ready queue is used as the time slice.

Abdulrahim, Abdullahi and Sahalu [1] introduced a New Improved Round Robin (NIRR) scheduling algorithm, improving on IRR algorithm developed by Mishra and Khan [10]. The time slice is calculated using the ceiling of the average of the burst time of the tasks in the ready queue, instead of fixed time slice in IRR. The tasks in ready queue are allocated to the CPU for duration of one time slice. After executing for one time slice, it checks if the currently executed task remaining burst time is less than half of a time slice. If so, the CPU will continue executing the task until completion. Otherwise the current unfinished task will be moved to the tail of arrival queue and the CPU will be allocated to the next task at the head of ready queue.

III. Materials and Methods

A simulation model is developed to study the behaviour of a system and to predict the performance of the system under varying set of circumstances [3]. The advantage of developing a simulation model is that once validated, the model can be used to investigate a wide variety of the behaviour of the system without accessing to or interrupting the real system.

Discrete Event Simulation (DES) is defined as a method of modeling a system in which the states of a model change in instantaneously at separate point in time [3] [9]. The operation of a system is modeled as a discrete sequence of events in time. The occurrence of an event at a particular instant in time marks one or more changes of the states in the system. Between two consecutive events, no change in the system is assumed to occur and thus the simulation can advanced in time from one event to the next.

In this research, the NIRR scheduling is modeled using DES. NIRR scheduling is by nature a discrete sequence of events in time. The model simulates the flow of a stream of tasks into the system which waiting to be executed by the CPU. There are two events, Task Arrival event and Task Execution event. The states of the tasks change at the occurrence of these events. Between two consecutive events, there are no changes in the states. The development of NIRR simulator is following the simulation life cycle described by [2] [16].

A. Study Definition Phase

Every study should begin with a statement of problem [9]. In this phase, the problem of interest and the objectives of the study are identified. The objectives shall indicate the question to be answered by the simulation. The input and output of the system are also identified in this phase.

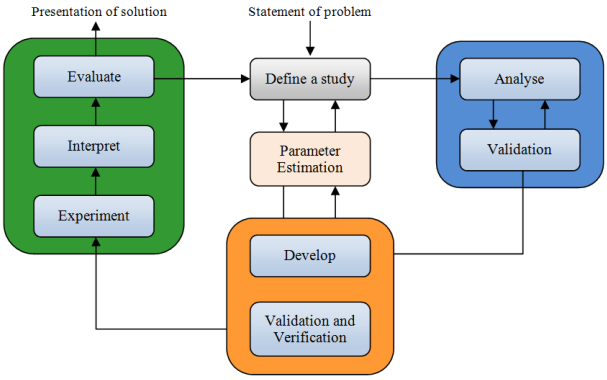

Figure 1. Simulation Life Cycle [2] [16]

B. *Analysis Phase*

The analysis phase establishes the groundwork for developing the simulation model. According to [8], "a simulation model should be a simplification of the real system, with just enough details to answer the question of interest". In this phase, a conceptual model is defined, in which the main components such as entities, queues, events and resources are identified. The logical relationships among these components are defined in this phase.

C. *Parameter Estimation Phase*

The parameters used in the model are determined by the objective of the study, the specific issues to be address and the performance measures of interest. The values of the parameters quantified the effect in the simulation. To develop a valid and credible model, these parameters must be able to reflect the real system. In this research, the simulation parameters are adopted from the benchmark model, which is the NIRR algorithm by [1].

D. *Model Development Phase*

The next stage in the simulation life cycle is the model development phase. This phase can be divided into three steps, which are model translation, verification and validation. In model translation, the conceptual model is translated into a computer program called computerized model. Verification is to determine if the input parameters and conceptual model has been correctly translated into the computerized model [8].

Validation is an important step to determine whether a simulation model is accurately representing the system for a particular objective of study [7] [8] [9]. According to [2] [7] [8] [9], the most definitive test of a simulation model's validity is by establishing that its output data closely resemble the output data that would be observed from the benchmark model. In this research, an NIRR simulation model is developed and is compared with the output data published by [1].

E. *Experimentation and Result Analysis Phase*

Experimentation and result analysis phase is the final stage. In this phase, the simulation model is tested with several sets of parameters and repeated for a number of independent runs in order to provide the required confidence in the measure of the performance. The simulation results are analyzed. The conclusions drawn from the analysis are presented in the conclusion section.

IV. NIRR Simulation Framework

NIRR simulation framework, as shown in Figure. 2, consists of three main components:

- DES Simulation component
- Entities and Resources component
- Scheduling Algorithm component

A. *DES Simulation Component*

This is the core component that executes the developed NIRR simulator. The flow chart in Figure. 3 illustrates the structure and the execution of the simulator. DES models all shares a number of common subcomponents which consists of the event list, event scheduler, time advancing mechanism, Termination of Simulation and report generator.

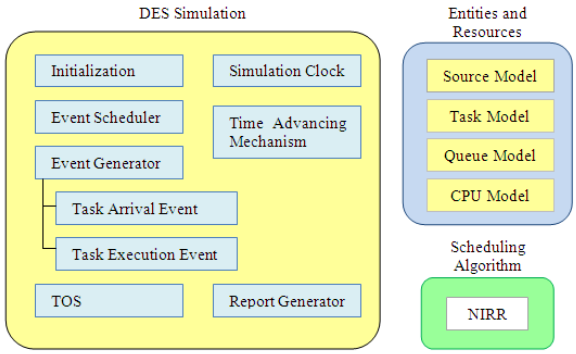

Figure 2. Simulation Life Cycle [2] [16]

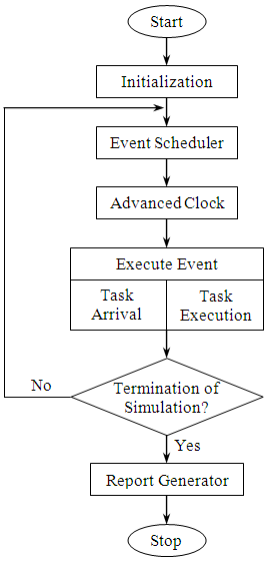

Figure 3. Flow Chart of NIRR Simulator

The simulator maintains an event list, in which it contains a list of each type of events that will occur and their time of occurrence. There are two types of events, which are Task Arrival event and Task Execution event. The event list contains the list of tasks about to arrive and the task currently executed by the CPU.

The event scheduler scans the event list and selects the event with the earliest time of occurrence, called the imminent event. The imminent event is removed from the event list and the simulation clock is advanced to the occurrence time of the imminent event. The event routine of the imminent event will be executed. The system state variables affected by this event are updated. This process of advancing simulation clock from one event time to another is continued until Termination of Simulation conditions are met.

Termination of Simulation (TOS) is critical in determining the validity of the acquired results. The developed NIRR simulator is tested with 5 to 1000 tasks. Therefore the simulation is terminated after the CPU executed 1000 tasks.

B. Entities and Resources Component

A simulation model is a well-defined collection of entities and resources [9]. These entities and resources are the objects of interest in the system [2] [7] [16]. The NIRR simulator conceptual model in Figure. 4 shows the entities and resources, and their logical relationship.

- Source Model: Source represents the load generator of the system under study, which models the task arrival process. The source generates 5 to 1000 tasks and initializes two attributes: arrival time and burst time. The arrival time is the time which the task is injected into the system while the burst time is time required to complete the task. In this research, the arrival time of all the tasks are set to zero. The burst time of these tasks follows uniform distribution U(1,50). At the beginning of the simulation, the event list is initialized with a stream of task arrival event in which the tasks arrival time are used as the event occurrence time.
- Task Model: Task is the main object of interest in the system. Figure. 5 summarized the life cycle of the task. A task is injected into the system by during the Task Arrival event. Upon arrival, the task is placed in the queue, waiting for its turn to access the CPU. The execution of the task is taken care in the Task Execution event. When it is the turn for the task, the CPU executes the task for the duration of a time slice. If the task is not completed after the duration of time slice, the task is preempted, moved to the queue and wait for next turn. These processes continue until the task is completed.
- Queue Model: The tasks are retained in the queue while waiting for the CPU. In NIRR simulation model, the tasks queue in two stages: Arrival Queue and Ready Queue. Newly arrived tasks and preempted tasks are retained in the Arrival Queue. Ready Queue stores the tasks that waiting for the CPU. The tasks in Ready Queue are sorted based on ascending order of the remaining burst time of the tasks. In this research, both queues are unlimited in size and these buffers are never full.
- CPU Model: The resource in this simulation model is a single CPU. The CPU is represented by either IDLE state or BUSY state. When the CPU is IDLE, the scheduler will select a task from the Ready Queue to be executed and the CPU is changed to BUSY state. Only one task is executed at a time. The task will be preempted after running for duration of a time slice and will be moved to the Arrival Queue. The scheduler will select the next task in Ready Queue to be executed. When all tasks have completed execution and there are no tasks waiting in both queues, the CPU will be changed to IDLE state.

C. Scheduling Algorithm component

The most important action in NIRR algorithm takes place in the routine of the events. The scheduling algorithm in this research is adopted from the benchmark NIRR scheduling algorithm by [1]. There are two events in NIRR simulator: Task Arrival event and Task Execution event.

The Task Arrival routine generates a stream of tasks and injects into the system. When a task is generated, the state of the CPU is checked. If the CPU is BUSY, the task will be admitted to the tail of the Arrival Queue. On the other hand, if the CPU is IDLE, the task is executed immediately. When the CPU is IDLE, both Arrival Queue and Ready Queue are empty. Since there are no other tasks in the queue, the time slice is given the value of the current task's burst time. The state of the CPU is changed to BUSY state.

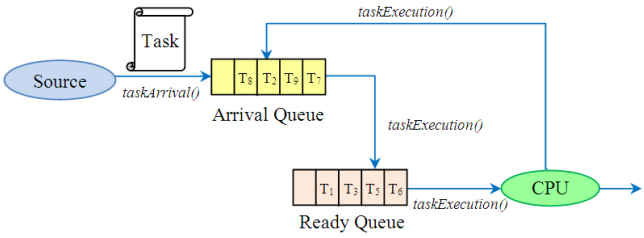

Figure 4. NIRR Simulation Model

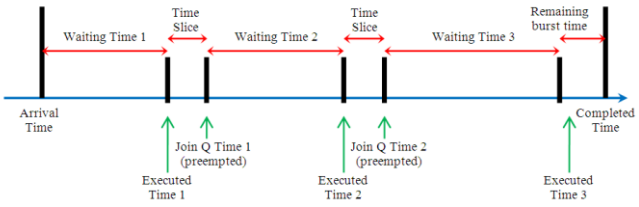

Figure 5. Task Model

Figure. 6 is the flowchart of the task execution routine. The routine can be divided into five stages.

- Stage 1: Task execution routine begins with checking whether the currently running task has run for its remaining burst time or it has run for duration of a time slice. The remaining burst time of the task is compared against the time slice. If remaining burst time is smaller or equal to time slice, the task has finished its execution. The totalCompletedTask counter is incremented. It keeps track of the number of tasks executed by the CPU and is used to check for the Termination of Simulation criteria. The turnaround time is computed when the task has finished its execution.
- Stage 2: Upon completion of a task, the routine will select another task from the queue to be executed. The Ready Queue is checked first. If the Ready Queue is not empty, the task at the head of the queue will be executed by the CPU. The waiting time and response time of the currently running task are computed. The routine will then schedule for the next Task Execution event.
- Stage 3: However, if the Ready Queue is empty, the Arrival Queue will be checked. If the Arrival Queue is not empty, all the tasks in the Arrival Queue are transferred to the Ready Queue. The tasks in the Ready Queue are sorted in ascending order of the remaining burst time of the tasks. The time slice is determined after these tasks are transferred to the Ready Queue. The time slice is taken from the ceiling of the average of the remaining burst time of the tasks in the Ready Queue. Then the task at the head of the Ready Queue will be executed by the CPU. The routine jumps to the algorithm in Stage 2 to select another task from the queue to be executed.
- Stage 4: If both Ready Queue and Arrival Queue are empty, meaning there are no tasks waiting to be executed, the state of the CPU is changed to IDLE.
- Stage 5: Referring to Stage 1, if the remaining burst time is greater than time slice, this indicates that the task was executed for duration of time slice. The remaining burst time is deducted by time slice and the new remaining burst time is compared with half of time slice. If the new remaining burst time is larger, the task is preempted and moved to the Arrival Queue. Then the task at the head of the Ready Queue will be executed by the CPU. The routine jumps to the algorithm in Stage 2 to select another task from the queue to be executed. Otherwise, if the new remaining burst time is smaller, the CPU continues executing the task until completion. The routine will then schedule for the next Task Execution event to take place after duration of remaining burst time.

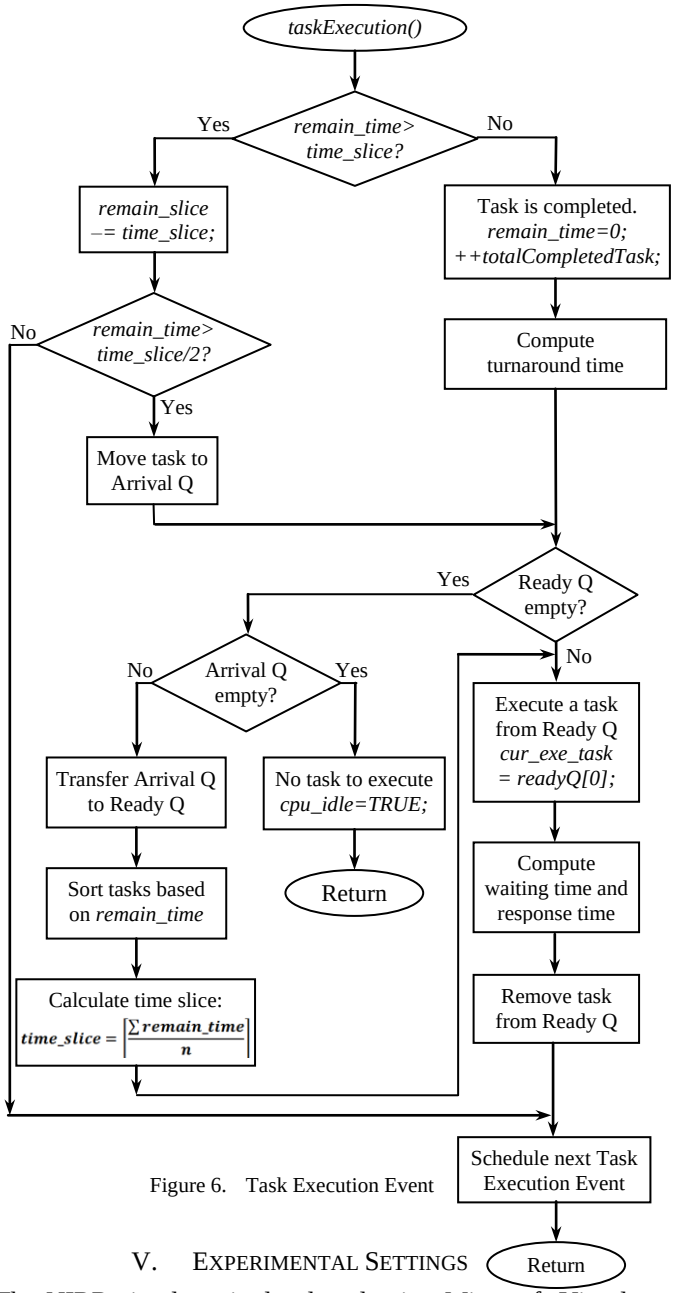

Figure 6. Task Execution Event

V. EXPERIMENTAL SETTINGS

The NIRR simulator is developed using Microsoft Visual C++ 2008. Extensive experiments were carried out on a Toshiba Notebook with Intel Core i5-3210M processor at speed of 2.50GHz and with 4.0GB Random Access Memory (RAM), running 64-bit Windows 8 operating system.

The simulation parameters are adopted from the benchmark NIRR scheduling algorithm by [1]. Table I summarizes the simulation parameters used in the developed NIRR simulator.

TABLE I. SIMULATION PARAMETERS

Parameters	Values
Number of tasks	5 to 1000 tasks
Task Arrival Time	0 milliseconds
Task Burst Time	Uniform (1, 50)
Resources	Single CPU
Termination of Simulation	CPU executed 1000 tasks
Experiment Repetition	100 times

A task generator was built to generate 100 task sets. Each task set contains 1000 tasks. The task arrival time is set to 0 milliseconds for all the tasks in these task sets. The burst times of the tasks are generated using uniform distribution range between 1 – 50 milliseconds.

The developed NIRR simulator runs the experiment for 100 times. Each experiment utilized a different task set to generate a set of intermediate results. The final results are generated from the average of the results of these experiments.

VI. RESULTS

The most definitive test of a simulation model's validity is establishing that its output data closely resemble the output data that would be observed from the benchmark model [2] [7] [8] [9]. Using the same assumptions and experimental settings in [1], NIRR simulator generates average waiting time, average turnaround time and average response time to be validated against the benchmark NIRR.

Figure. 7 compares the average waiting time generated by the NIRR simulator with the benchmark NIRR. The graph shows that the average waiting time increases as the number of tasks increases. The results generated by the NIRR simulator closely resemble the data from the benchmark NIRR and have the same trend. Table II shows the details of the comparison.

Turnaround time measures of the total time it takes for the CPU to complete the execution of a task. It includes the waiting time and execution time of the task. Therefore, average turnaround time demonstrates the same trend as average waiting time. Figure. 8 compares the average turnaround time generated by the NIRR simulator with the benchmark NIRR. The average turnaround time increases as the number of tasks increases. The graph shows that output generated by the NIRR simulator closely resemble the data from the benchmark NIRR and have the same trend. Table III shows the details of the comparison.

Figure. 9 shows the graph of the average response time generated by the NIRR simulator and the benchmark NIRR. Similar to average waiting time and average turnaround time, the graph shows that the average response time increases as the number of tasks increases. The results generated by the NIRR simulator and the benchmark NIRR have the same trend. Table IV shows the details of the comparison.

TABLE II. AWT OF BENCHMARK MODEL AND NIRR SIMULATOR

No. of Tasks	Benchmark NIRR (sec)	NIRR Simulator (sec)	Difference (%)
200	1.7	1.8	5.9
400	3.5	3.7	5.7
600	5.1	5.5	7.8
800	7.1	7.4	4.2
1000	8.9	9.2	3.4

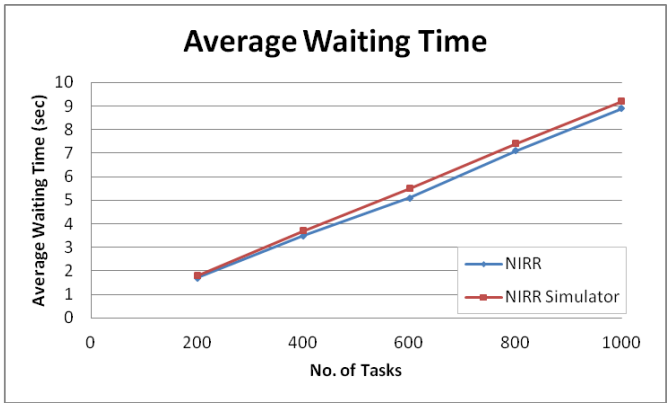

Figure 7. Graph of Average Waiting Time

TABLE III. ATAT OF BENCHMARK MODEL AND NIRR SIMULATOR

No. of Tasks	Benchmark NIRR (sec)	NIRR Simulator (sec)	Difference (%)
200	1.8	1.9	5.6
400	3.4	3.7	8.8
600	5.1	5.6	9.8
800	7.2	7.4	2.8
1000	8.9	9.2	3.4

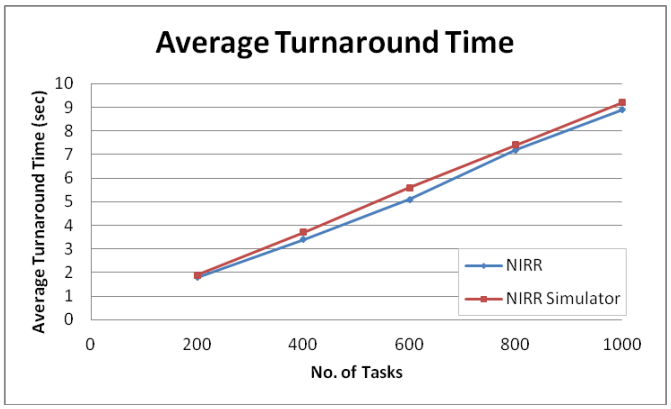

Figure 8. Graph of Average Turnaround Time

TABLE IV. ART OF BENCHMARK MODEL AND NIRR SIMULATOR

No. of Tasks	Benchmark NIRR (sec)	NIRR Simulator (sec)	Difference (%)
200	1.5	1.6	6.7
400	2.9	3.2	10.3
600	4.3	4.9	14.0
800	6.1	6.5	6.6
1000	7.7	8.1	5.2

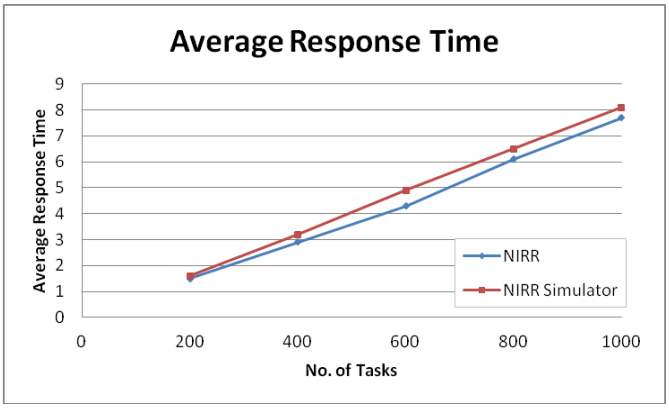

Figure 9. Graph of Average Response Time

VII. CONCLUSION AND FUTURE WORKS

Extensive experiments were done to verify and validate the simulation model. The output data generated by the developed NIRR simulator is compared with the data from benchmark NIRR using the same experimental setting in [1]. The obtained results using the simulation model closely resembles the data from benchmark NIRR. The success of the developed NIRR simulator was proven by the generated results.

In the future work, this research can be extended into a few directions like:

- The simulation model provides a platform for further investigation of NIRR algorithm. The algorithm may be tested on different statistical distribution of the burst time and arrival time of the tasks, and evaluate on different performance criteria like CPU utilization, throughput and fairness.
- Another direction is to develop an enhanced version of the algorithm for multi-processor environment. The algorithm may be tested on tasks which may or may not be dependent on each other.
- The algorithm can be extended to real-time system and compare with other scheduling approaches. In real-time system, tasks have priorities and deadline constraints. Real-time system related metrics shall be measured, like the deadline miss ratio, accrued utility ratio, and CPU utilization.
- Another interesting future direction is to apply the algorithm to the processors or controllers of Internet of Things.

REFERENCES

[1] Abdulrahim, A., Abdullahi, S. E., & Sahalu, J. B. (2014). A New Improved Round Robin (NIRR) CPU Scheduling Algorithm. International Journal of Computer Applications, 90(4).

[2] Ahmad, I., Subramaniam, S., Othman, M., & Zulkarnain, Z. (2011). A discrete event simulation framework for utility accrual scheduling algorithm in uniprocessor environment. Journal of Computer Science, 7(8), 1133.

[3] Banks, J., Carson, J., Nelson, B. and Nicol, D. 2000. Discrete Event System Simulation, 3rd Edition, Prentice Hall.

[4] Behera, H. S., Mohanty, R., & Nayak, D. (2011). A New Proposed Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis. International Journal of Computer Applications, 5(5), 10-15.

[5] Dawood, A. J. (2012). Improving Efficiency of Round Robin Scheduling Using Ascending Quantum and Minimum-Maximum Burst Time. Journal of University of Anbar for Pure Science, 6(2).

[6] Lalit, K., Rajendra, S., & Praveen, S. (2011). Optimized Scheduling Algorithm. International Journal of Computer Applications, 106-109.

[7] Law, A. M. (2009, December). How to build valid and credible simulation models. In Simulation Conference (WSC), Proceedings of the 2009 Winter (pp. 24-33). IEEE.

[8] Law, A. 2003. How to Conduct a Successful Simulation Study, Proceedings of the 2003 Winter Simulation Conference: 66-70.

[9] Law, A. and Kelton, W. 2000. Simulation Modeling and Analysis, 3rd Edition, McGraw-Hill Higher Education.

[10] Mishra, M. K., & Khan, A. K. (2012). An Improved Round Robin CPU Scheduling Algorithm. Journal of Global Research in Computer Science, 3(6), 64-69.

[11] Mishra, M. K., & Rashid, F. (2014). An Improved Round Robin CPU Scheduling Algorithm With Varying Time Quantum. International Journal of Computer Science, Engineering and Applications (IJCSEA), 4(4).

[12] Noon, A., Kalakech, A., & Kadry, S. (2011). A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum using the Mean Average. International Journal of Computer Science, 8(1).

[13] Rajput, I. S., & Gupta, D. (2012). A Priority Based Round Robin CPU Scheduling Algorithm for Real Time Systems. International Journal of Innovations in Engineering and Technology, 1(3), 1-11.

[14] Saeidi, S., & Baktash, H. A. (2012). Determining the Optimum Time Quantum Value in Round Robin Process Scheduling Method. International Journal of Information Technology and Computer Science (IJITCS), 4(10), 67.

[15] Soraj, H., & Roy, K. C. (2012). Adaptive Round Robin scheduling using shortest burst approach, based on smart time slice. International Journal of Data Engineering (IJDE), 2(3).

[16] Watkins, K. 1993. Discrete Event Simulation in C, McGraw-Hill.