International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 04 Issue 10
September 2017

User Level Simulation of Scheduling Algorithms in Multicore

[1]Mrs. Dhruva R. Rinku ; [2]Dr. M. AshaRani ; [3]N.Kavya,

[1]Associate Professor, Dept. of ECE, CVR College of Engineering, Hyderabad

[2]Professor, Dept. of ECE, JNT University, Hyderabad

[3]M.Tech Student, CVR College of Engineering, Hyderabad

rinkudhruva.ravi@gmail.com ; kavya.nalabolu@gmail.com ; ashajntu1@yahoo.com

Abstract:

Scheduler is an object of the kernel which allocates available resources to the process. It works differently for single core and multicore processors. In single core, scheduler allots the processes to CPU one at a time concurrently. In single core architecture it is time consuming process for execution of more number of processes. Performance can be improved by executing the processes parallel in multicore, scheduler allots simultaneously processes to each core for execution. There are different scheduling algorithms which are used to schedule the processes in multicore. In multicore the result of scheduling criteria is different for each scheduling algorithms. In this project we have proposed and observed the simulation of different scheduling algorithms at user level for single core and multicore and compared the different metrics like average waiting time, turnaround time, CPU utilization and number of context switches in graphical manner and also observed the improved performance of scheduler in multicore environment...

Keywords

Scheduler, Multicore, CPU utilization, Context switch.

1. Introduction

The scheduler is an operating system module that selects the next jobs to be admitted into the system and the next process to run. Operating systems may feature up to three distinct scheduler types: a long-term scheduler (also known as an admission scheduler or high-level scheduler), a mid-term or medium-term scheduler, and a short-term scheduler. The names suggest the relative frequency with which their functions are performed.

It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling. The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound.

In computing, scheduling is the method by which work specified by some means is assigned to resources that complete the work. The work may be virtual computation elements such as threads, processes or data flows, which are in turn scheduled onto hardware resources such as processors, network links or expansion cards.

A scheduler is what carries out the scheduling activity. Schedulers are often implemented so they keep all computer resources busy (as in load balancing), allow multiple users to share system resources effectively, or to achieve a target quality of service. Scheduling is fundamental to computation itself, and an intrinsic part of the execution model of a computer system; the concept of scheduling makes it possible to have computer multitasking with a single central processing unit (CPU).

A scheduler may aim at one of many goals for ex, maximizing throughput (the total amount of work completed per time unit), minimizing response time (time from work becoming enabled until the first point it begins execution on resources), or minimizing latency (the time between work becoming enabled and its subsequent completion), maximizing fairness (equal CPU time to each process, or more generally appropriate times according to the priority and workload of each process). In practice, these goals often conflict (e.g. throughput versus latency), thus a scheduler will implement a suitable compromise. Preference is given to any one of the concerns mentioned above, depending upon the user's needs and objectives.

In real-time environments, such as embedded systems for automatic control in industry (for example robotics), the scheduler also must ensure that processes can meet deadlines; this is crucial for keeping the system stable. Scheduled tasks can also be distributed to remote devices across a network and managed through an administrative back end.

The long-term scheduler, or admission scheduler, decides which jobs or processes are to be admitted to the ready queue (in main memory); that is, when an attempt is made to execute a program, its admission

to the set of currently executing processes is either authorized or delayed by the long-term scheduler. Thus, this scheduler dictates what processes are to run on a system, and the degree of concurrency to be supported at any one time whether many or few processes are to be executed concurrently, and how the split between I/O-intensive and CPU-intensive processes is to be handled. The long-term scheduler is responsible for controlling the degree of multiprogramming.

In general, most processes can be described as either I/O-bound or CPU-bound. An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations. A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations. It is important that a long-term scheduler selects a good process mix of I/O-bound and CPU-bound processes. If all processes are I/O-bound, the ready queue will almost always be empty, and the short-term scheduler will have little to do. On the other hand, if all processes are CPU-bound, the I/O waiting queue will almost always be empty, devices will go unused, and again the system will be unbalanced. The system with the best performance will thus have a combination of CPU-bound and I/O-bound processes. In modern operating systems, this is used to make sure that real-time processes get enough CPU time to finish their tasks.

Long-term scheduling is also important in large-scale systems such as batch processing systems, computer clusters, supercomputers, and render farms. For example, in concurrent systems, co-scheduling of interacting processes is often required to prevent them from blocking due to waiting on each other. In these cases, special-purpose job scheduler software is typically used to assist these functions, in addition to any underlying admission scheduling support in the operating system.

Scheduling of tasks is essential for every system. Scheduler is one of the most important parts of an Operating system[1]. Without scheduler, tasks may not execute in that order which a user or operating system itself want. Scheduling of tasks on single core processor is much easy by choosing existing scheduler programs like Round-Robin, Priority based, First Come First Serve bases, Shortest job First etc. Multi-core processors [2] have two or more processing elements or cores on a single chip. These cores could be of similar architecture (Synchronous Multi-core Processors, SMPs) or of different architecture (Asynchronous Multi-core Processors, AMPs). All the cores necessarily use shared memory architecture. Multi-core processors have existed previously in the form of MPSoC (Multi-Processor System on Chip) but they were limited to a segment of applications such as networking.

The easy availability of multi -core has forced software programmers to change the way they think and write their applications. Unfortunately, the applications written so far are sequential in nature. Multi-core processors [8] do not automatically provide performance improvements to applications the way faster processors did. Instead applications must be redesigned to increase their parallelism. Similarly, CPU schedulers must be redesigned to maximize the performance of this new application parallelism. CPU scheduling policy (and in a large part mechanism) is unimportant to a serial application running on its own machine. An important part of parallel processing in multi-core reconfigurable systems is to allocate tasks to processors to achieve the best performance.

The objectives of task scheduling algorithms are to maximize system throughput by assigning a task to a proper processor, maximize resource utilization, and minimize execution time. In a single-processor system, only one process can run at a time; any others must wait until the CPU is free and can be rescheduled. The objective of multi-tasking is to have some process running at all times, to maximize CPU utilization. Scheduling is a fundamental operating-system function. Almost all computer resources are scheduled before use. The CPU is, of course, one of the primary computer resources. Thus, its scheduling is central to operating-system design. CPU scheduling determines which processes run when there are multiple run-able processes.

2. System Analysis

Scheduler selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling. The primary objective of the job scheduler is to provide a balanced mix of jobs, such as/O bound and processor bound. Time-sharing operating systems have no long term scheduler. It works differently for single core and multicore.

Single CPU systems use scheduling and can achieve multi-tasking because the time of the processor is time-shared by several processes so allowing each process to advance in parallel as shown in fig 1. So a process runs for some time and another waiting gets a turn. Reassigning a CPU from one task to another one is called a context switch. In a synergy between hardware and operating system, the CPU is allocated to different processes several times per second and this act is called 'time-slicing' so not to be confused with Multi-threading. Time-slicing is older technology compared to Multi-threading. There are lot of benefits in using

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 04 Issue 10
September 2017

multithreading like more efficient cpu use, better system reliability. The scheduler in operating system will assign each task to core by priority.

Fig 1: single core function

In Multicore scheduling [4][5] each task is assigned to the each core as shown in fig 2. Average waiting time is decreases that increases the throughput, improving scheduling criteria.

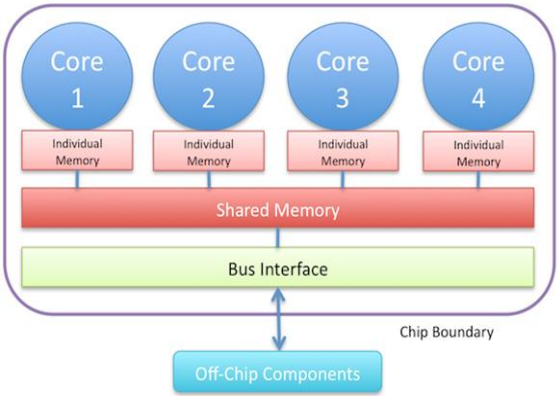

Fig 2: multicore function

Operating systems schedule threads either pre-emptively or cooperatively. On multi-user operating systems, pre-emptive multithreading is the more widely used approach for its finer grained control over execution time via context switching. However, pre-emptive scheduling may context switch threads at moments unanticipated by programmers therefore causing lock convoy, priority inversion, or other side-effects. In contrast, cooperative multithreading relies on threads to relinquish control of execution thus ensuring that threads run to completion. This can create problems if a cooperatively multitasked thread blocks by waiting on a resource or if it starves other threads by not yielding control of execution during intensive computation [7].

Until the early 2000s, most desktop computers had only one single-core CPU, with no support for hardware threads, although threads were still used on such computers because switching between threads was generally still quicker than full-process context switches. In 2002, Intel added support for simultaneous multithreading to the Pentium 4 processor, under the name hyper-threading; in 2005, they introduced the dual-core Pentium D processor and AMD introduced the dual-core Athlon 64 X2 processor.

Processors in embedded systems, which have higher requirements for real-time behaviors, might support multithreading by decreasing the thread-switch time, perhaps by allocating a dedicated register file for each thread instead of saving/restoring a common register file.

Scheduling can be done at the kernel level or user level, and multitasking can be done pre-emptively or cooperatively. This yields a variety of related concepts.

At the kernel level, a process contains one or more kernel threads, which share the process's resources, such as memory and file handles – a process is a unit of resources, while a thread is a unit of scheduling and execution. Kernel scheduling is typically uniformly done pre-emptively or, less commonly, cooperatively. At the user level a process such as a runtime system can itself schedule multiple threads of execution. If these do not share data, as in Erlang, they are usually analogously called processes, while if they share data they are usually called (user) threads, particularly if pre-emptively scheduled. Cooperatively scheduled user threads are known as fibres; different processes may schedule user threads differently. User threads may be executed by kernel threads in various ways (one-to-one, many-to-one, many-to-many). The term "light-weight process" variously refers to user threads or to kernel mechanisms for scheduling user threads onto kernel threads.

A process is a "heavyweight" unit of kernel scheduling, as creating, destroying, and switching processes is relatively expensive. Processes own resources allocated by the operating system. Resources include memory (for both code and data), file handles, sockets, device handles, windows, and a process control block. Processes are isolated by process isolation, and do not share address spaces or file resources except through explicit methods such as inheriting file handles or shared memory segments, or mapping the same file in a shared way see inter process communication. Creating or destroying a process is relatively expensive, as resources must be acquired or released. Processes are typically pre-emptively multitasked, and process switching is relatively expensive, beyond basic cost of context switching, due to issues such as cache flushing.

Kernel thread is a "lightweight" unit of kernel scheduling. At least one kernel thread exists within each process. If multiple kernel threads exist within a process, then they share the same memory and file

resources. Kernel threads are pre-emptively multitasked if the operating system's process scheduler is pre-emptive. Kernel threads do not own resources except for a stack, a copy of the registers including the program counter, and thread-local storage (if any), and are thus relatively cheap to create and destroy. Thread switching is also relatively cheap: it requires a context switch (saving and restoring registers and stack pointer), but does not change virtual memory and is thus cache-friendly (leaving TLB valid). The kernel can assign one thread to each logical core in a system (because each processor splits itself up into multiple logical cores if it supports multithreading, or only supports one logical core per physical core if it does not), and can swap out threads that get blocked. However, kernel threads take much longer than user threads to be swapped.

Threads are sometimes implemented in user space libraries, thus called user threads. The kernel is unaware of them, so they are managed and scheduled in user space. Some implementations base their user threads on top of several kernel threads, to benefit from multi-processor machines (M : N model). In this article the term "thread" (without kernel or user qualifier) defaults to referring to kernel threads. User threads as implemented by virtual machines are also called green threads. User threads are generally fast to create and manage, but cannot take advantage of multithreading or multiprocessing, and will get blocked if all of their associated kernel threads get blocked even if there are some user threads that are ready to run.

For scheduling process in single core or multicore scheduler must reach the scheduling criteria. The scheduling criteria includes Parameters like

Scheduling Criteria

- **CPU Utilization**: It is the average fraction of time, during which the processor is busy.
- **Throughput**: It refers to the amount of work completed in a unit of time. The number of processes the system can execute in a period of time. The higher the number, the more work is done by the system.
- **Waiting Time:** The average period of time a process spends waiting. Waiting time may be expressed as turnaround time less the actual execution time.
- **Turnaround time**: The interval from the time of submission of a process to the time of completion is the turnaround time.
- **Response time:** Response time is the time from submission of a request until the first response is produced.
- **Priority:** give preferential treatment to processes with higher priorities.
- **Fairness:** Avoid the process from starvation. All the processes must be given equal opportunity to execute.

This project is implemented on Linux operating system and simulated scheduling algorithms in single and multicore and comparing the different metrics and verifying improvement of scheduling criteria. The block diagram is shown in fig 3.

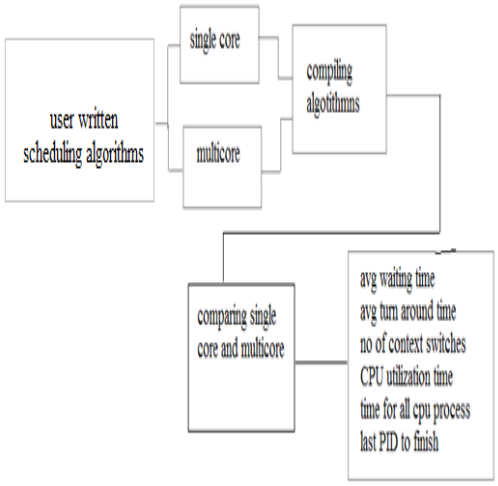

Fig 3: block diagram

3. Scheduling Algorithms

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms [3] [9]. There are six popular process scheduling algorithms which we are going to discuss in this chapter

- First-Come, First-Served (FCFS) Scheduling.
- Shortest-Job-Next (SJN) Scheduling.
- Priority Scheduling.
- Shortest Remaining Time.
- Round Robin (RR) Scheduling.
- Multiple-Level Queues Scheduling.

These algorithms are either non-pre-emptive or pre-emptive. Non-pre-emptive algorithms are designed so that once a process enters the running

state, it cannot be pre-empted until it completes its allotted time, whereas the pre-emptive scheduling is based on priority where a scheduler may pre-empt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve Scheduling

- Jobs are executed on first come, first serve basis.
- It is a non-pre-emptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Shortest Job Next Scheduling

- This is also known as shortest job first, or SJF
- This is a non-pre-emptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Priority Based Scheduling

- Priority scheduling is a non-pre-emptive algorithm and one of the most common scheduling algorithms in batch systems[10].
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Shortest Remaining Time Scheduling

- Shortest remaining time (SRT) is the pre-emptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be pre-empted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

Round Robin Scheduling

- Round Robin [6] is the pre-emptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a quantum.
- Once a process is executed for a given time period, it is pre-empted and other process executes for a given time period.
- Context switching is used to save states of pre-empted processes.

Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

Simulation of scheduling algorithms includes concept of enqueuer and dequeuer where the number of process used for simulation follows enqueuer and dequeuer concept. Enqueue is adding the process into the queue for execution and dequeue is removing the finished process from the queue as shown in the fig 4.

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 04 Issue 10
September 2017

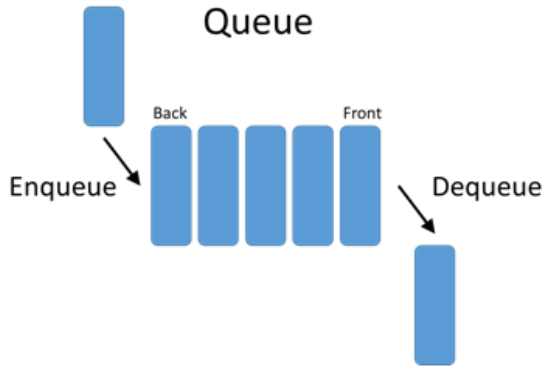

Fig 4: enqueue and dequeue

This project shows the simulation of first come first served algorithm and round robin algorithm.

First come first served algorithm

First come first served algorithm in which process which will come first in queue will be executed first, depending upon number of cores which are used for execution the performance will be changed. The number of cores on which process has to execute will be selected manually at user level.

The code written for the algorithm includes sch-helper.h file which indicates about the maximum number of process given as input, number of cores, etc, It specifies different functions that can be shown in flowchart.

4. Design and Implementation

Scheduling algorithms first come first served ,round robin and multilevel feedback queue algorithms is simulated in single core and multicore and dual core by changing the number of processors at user level in sch-helper.h file which is included in main program while simulation, and input is taken from CPUdat.dat file which is created in the form of process pid, arrival Time, start Time, end Time, waiting time, current burst, no of burst, maximum bursts, priority, quantum remaining, current queue. Log file is also created which shows process of execution. There are different cases of simulation.

First Come First Served Algorithm

The implementation of first come first served algorithm is done in C programming language which is more suitable for systems and provides an efficient mapping to machine instructions. for implementation of first come first serve a process that request the CPU first is allocated to it first. Way the program actually measures the progress is through a variable step which is a part of process data structure. Each iteration increases the step of a working CPU until the cpu burst length or i/o burst length is reached. If that condition is satisfied the program fetches the next burst and repeat the whole process until there is no more process to be fed into the CPU.

After simulation of algorithm each process is sorted based on their arrival time and execution time for each process and average waiting time, total context switches, CPU utilization, average turnaround time, last process to finish is observed as shown fig 5 and fig 6. From above simulation it is observed that average waiting time is less that means waiting time for process to get CPU is very less that increases the throughput. Average turnaround time is less and CPU utilization is increased.

From all simulation cases it is clear that average waiting for each process is increased as number of processors decreased, CPU utilization time also decreases, average turnaround time is increases, and this improves the scheduling criteria as shown in fig 7. This project is implemented by taking any number of process as input. All these results comparison is shown in the form of charts shown below.

Algorithm	Avg waiting time	Avg turnaround time	Cpu%
FCFS	23.2	40492.6	78.91
RR	20.43	40489.83	78.92
MFBQ	20.17	40494.97	78.90

Fig 5: Algorithms comparison in multicore

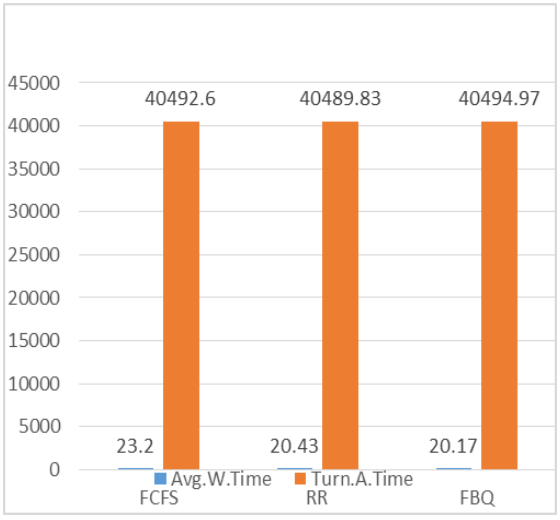

Fig 6: comparison of algorithms quad core

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 04 Issue 10
September 2017

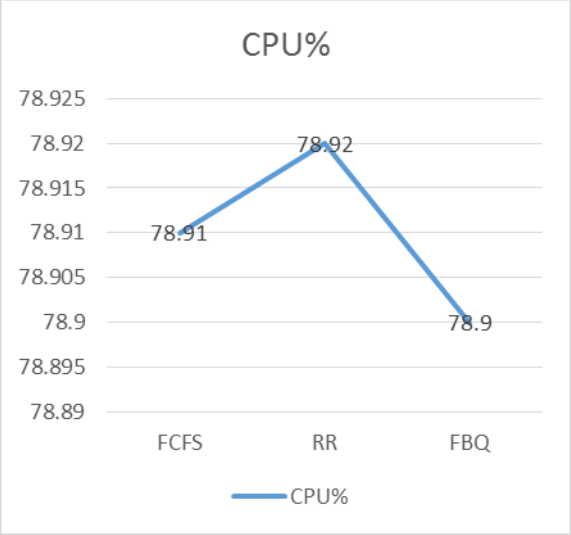

Fig 7: CPU% of algorithms in quad core

5. Conclusion and Future Scope

User level simulation of scheduling algorithms in single core and multicore has done. The results of scheduling algorithms in different cores are observed in different cases. Comparison of different metrics like average waiting time, CPU utilization, total turnaround time and number of context switches have observed and compared in different cores with graphs. It is observed that simulation of scheduling algorithms in multicore shows the improved scheduling criteria. By this observation it is concluded that the performance metrics of scheduler i.e. scheduling criteria is increases by increasing number of cores.

In this project scheduler performance for different scheduling algorithms is observed. Future scope of this project is designing scheduler to implement earliest deadline first algorithm to achieve execution of task with least deadline first. This project can be implement at kernel level and observe metrics in real time in bare metal hardware board with embedded cores.

6. References

1. Pankaj Gupta, Piyush Kumar, Sandeep, Saksham Wason, Vishal Yadav "Operating System" "International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online) Vol. 2, Issue 2, pp: (37-46), Month: April-June 2014.

2. Anil Sethi1, Himanshu Kushwah2 "Multicore Processor Technology- Advantages and Challenges" International Journal of Research in Engineering and Technology Volume: 04 Issue: 09 | September-2015.

3. Mahima Shrivastava, "Analysis and Review of CPU Scheduling Algorithms", International Journal of Scientific Engineering and Research (IJSER) ISSN (Online): 2347-3878 Volume 2 Issue 3, March 2014.

4. Suresh Siddha "Process Scheduling Challenges in the Era of Multi-core Processors" Volume 11 Issue 04 Published, November 15, 2007 ISSN 1535-864X DOI: 10.1535/itj.1104.0 Intel journals

5. Arpacci -Dusseau Multiprocessor scheduling (Advanced) 2014.

6. Debashee Tarai, "Enhancing Cpu Performance Using Subcontrary Mean Dynamic Round Robin (Smdrr) Scheduling Algorithm", International Journal of Scientific Engineering and Research (IJSER) Volume 2 Issue 3, March 2011.

7. Ravinder Jeet1, Upasna Garg2 "Selective Scheduling Based on Number of Processor Cores for Parallel Processing" International Journal of Science and Research (IJSR) ISSN (Online): 2319-7064 Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438 Volume 4 Issue 1, January 2015

8. Venkata Siva Prasad Ch. and S. Ravi "Scheduling Of Shared Memory With Multi - Core Performance In Dynamic Allocator Using Arm Processor "VOL. 11, NO. 9, MAY 2016 ISSN 1819-6608 ARPN Journal of Engineering and Applied Sciences ©2006-2016 Asian Research Publishing Network (ARPN).

9. Imran Qureshi, "CPU Scheduling Algorithms: A Survey", Int. J. Advanced Networking and Applications Volume: 05, Issue: 04, Pages: 1968-1973 (2014) ISSN: 0975-0290.

10. Ms. Rukhsar Khan1 , Mr. Gaurav Kakhani2 "Analysis of Priority Scheduling Algorithm on the Basis of FCFS & SJF for Similar Priority Jobs" International Journal of Computer Science and Mobile Computing Vol. 4, Issue. 9, September 2015.

Author details:
Author 1:

Mrs. Dhruva R. Rinku
Associate Professor
Dept. of ECE
CVR College of Engg.

Email: rinkudhruva.ravi@gmail.com

Author 2:

Dr. M. AshaRani
Professor
Dept. of ECE
JNTU, Hyderabad
Email: ashajntu1@yahoo.com

Author 3:

N.Kavya
Mtech-IIyr Embedded systems
Cvr College of engineering, Hyderabad.
Email: kavya.nalabolu@gmail.com