A hybrid approach to Enhancing Process Scheduling in multiple core Systems

Dr. Anthony Luvanda
School of Science, Alupe University College, P. O Box 845-50400, Busia, Kenya
E-mail: luvanda@gmail.com

Abstract

Process scheduling within computer systems and with regards to the CPU always encounters bottle necks due to over reliance on single processor scheduling techniques. The presence of multicore processor systems attempts to increase throughput without however operating at full optimum capacity, hence the need for the proposal of a more efficient scheduling approach for use in multiple core systems. This paper conducted a comparative analysis of the rate of efficiency of existing scheduling algorithms with the aid of secondary data. It then employed CPU user benchmark analysis to asses the effectiveness of the proposed approach. Quad-core processor systems are most suitable for the proposed approach which basically implements two approaches one where scheduling decisions are handled by a master processor while other processors execute the user code thus always ensuring that all processors are busy and always utilized.

Keywords: Process scheduling, multicore systems, system optimization

DOI: 10.7176/JIEA/11-2-04
Publication date: May 30th 2021

1. Introduction

For optimum performance to be achieved while using a computer system, all basic components must synchronize their activities and capabilities. This is however not usually the case with one reason for this being that CPUs still don't seem to perform at their best when dealing with actual processes. The bottle neck is caused by the fact that multicore systems still tend to rely heavily on single processor scheduling techniques thus denying them the capability to optimize their potential. This research aimed to propose a framework for optimizing multiple core processors systems by reducing the amount of idle time through the implementation of an enhanced process scheduling technique.

1.1 Single core processor systems vs multiple core processor systems

Computer systems in the olden days relied heavily on single processing core CPUs. This then begs the question, what is a single core processor system? Any microprocessor that contains a single core on a chip and is only able to run a single computing thread at any one time is considered to be a single core processor system (Bindal 2017).

Modern computers however are generally dominated by multiple core processor systems. Multiple core processor systems basically have the ability to run multiple computing threads at any one time. This is made possible by the fact that multiple core processor systems have several independent processors on a chip (Gerrit 1997). Which in turn provides the added advantage of increased throughput. In essence what this means is that the speed-up ratio with N processors is not N, however; it is less than N.

1.2 Process management and scheduling algorithms

Irrespective of its size, each and every program always has an alternating number of CPU instructions waiting for some form of input/output (John 2001). Single processor systems waste a lot of time when waiting for the input/output results. In most cases those CPU cycles are never recovered. These then leads to the need to implement some form of scheduling which allows for one set of instructions to access the CPU while another is waiting for input/output results thus almost eliminating idle time.

For us to appreciate how scheduling works we must first understand the concept of CPU burst and input/output burst. Consider a burst as a dash or a short sprint at an athletics meet, which is basically an athlete running as fast as they can till they can run no more. A CPU burst occurs when a processor is executing instructions and an I/O burst occurs when the said processor is fetching instructions. A CPU burst commences when the processor start running instructions from cache and ends when the processor finds the need to start fetching instructions or data from memory. An input/output burst on the other hand commences with the process of reading or writing data and ends when the requested data is written/read or when the space to store it runs out (Ahmet 2017) The

balancing act of managing these activities with the aim of maximize the use of the resources and minimize wait and idle time is what we are referring to as scheduling.
Due to the fact that idle time is inevitable the CPU scheduler has to pick another process from the ready queue to run next whenever the CPU is idle.

2. Methods and Materials

The research relied heavily on secondary data to perform a comparative analysis of common scheduling algorithms but in testing the hybrid scheduling approach a number of tests were run on a number of personal computers running on icore5 and icore7 processors with the aid of CPU user bench mark tool.

2.1 *Comparative analysis of the most common scheduling algorithms*

First Come First Serve

Considered to be the simples and easiest scheduling algorithm to implement basically due to the fact that based on the FIFO queue, the process that requests information first is the first to get CPU allocation. (Abraham 2015)
Consider the following table of arrival time and burst time for three processes P1,P2 and P3:

Process	Arrival time ms	Burst time in (ms)
P1	0ms	11ms
P2	2ms	7ms
P3	5ms	20ms

Table 2.1 FIFO scheduling scenario burst times

In FCSF, the Processes are executed in the order of arrival i.e. P1, P2 then P3 respectively.
- Waiting time for P1 = 0, P2 = 11, P3 = 18
- Average waiting time (0 + 11 + 18) / 3 = 9.999ms

What about in reverse order. P 3, P2, P1
- Waiting time P1 = 27, P2 = 20, P3 = 0
- Average waiting time (27 + 20 + 0) /3 = 33.66ms

The second case (reverse order) is poor due to the convoy effect: latter processes are held up behind a long-running first process as the average waiting time suggests

Shortest Job First

In this scheduling algorithm, the processes with the shortest execution time is the one that is next scheduled for processing. The scheduling itself can either be preemptive (where timeslots of a CPU are created and divided among processes) or non-preemptive (where processes occupy the CPU until termination of the process or the process is pushed to the waiting list). This in turn significantly reduces the average waiting time for other processes awaiting execution. (Abraham 2015)

Process	Execution time in (ms)
P1	10
P2	5
P3	8

Table 2.2 Shortest job first scenario execution times

The SJF scheduling algorithm allows for **P2** to be processed first then **P3** and finally **P1**.

Priority Scheduling

In this scheduling approach the scheduler relies on pre assigned priorities which in most cases are prioritized as system internal processes, interactive processes (which can be further sub divided) and batch processes with system internal processes having the highest priority and batch processes having the least priority, jobs with equal priorities are carried out on round-robin or FCFS basis. This approach however runs the risk of a low priority job never getting access to the processor. (Deitel 2015)

Round Robin Scheduling
It's the oldest and most common multitasking algorithm. It employs a time-sharing system that relies on the preemptive scheduling scheme. It defines a small fixed unit of time called a quantum (or time-slice) typically 10-100 milliseconds it has some properties as below
• Fair: given n processes in the ready queue and time quantum q, each process gets 1/nth of the CPU.
• Live: No process waits more than (n – 1) q time units before receiving a CPU allocation.
• Typically get higher average turnaround time than SRTF, but better average response time. (Kevin 2005)
Consider the following set of processes that arrive at time 0 ms.

Given a scenario where a set of processes arrive at time 0 ms

Process	**Burst time in (ms)**
P1	20
P2	3
P3	4

Table 2.3 Round robin scheduling scenario burst times

If we use time quantum of 4ms then calculate the average waiting time using R-R scheduling where scheduling processes are executed in FCFS

P1	P2	P3	P1	P1	P1	P1
0 4	7	11	15	19	23	27

Table 2.4 scheduled execution of processes based on FCFS

According to R-R scheduling processes are executed in FCFS order. So, firstly P1(burst time=20ms) is executed but after 4ms it is preempted and new process P2 (Burst time = 3ms) starts its execution whose execution is completed before the time quantum. Then next process P3 (Burst time=4ms) starts its execution and finally remaining part of P1 gets executed with time quantum of 4ms.
• Waiting time of Process P1: 0ms + (11 – 4)ms =7 ms
• Waiting time of Process P2: 4ms
• Waiting time of Process P3: 7ms
• Average Waiting time: (7 + 4 + 7)/3 = 6ms.

Multilevel Queue Scheduling
This algorithm separates the ready queue into various separate queues. In this method, processes are assigned to a queue based on a specific property of the process, like the process priority, size of the memory, etc. However, this is not an independent scheduling algorithm as it needs to use other types of algorithms in order to schedule the jobs (Maciej 2009)

3 Conclusion
In multiple processor systems processors maybe homogeneous (identical) or heterogeneous (unidentical). The scheduling itself can take two approaches, one being where all scheduling decisions are handled by a single processor called Master Server (processor) and other processors execute the user code with the second being where multiprocessing is used in that each processor is self-scheduling. All processes may be in a common ready queue or each processor has its own private queue for ready processes.

The proposed enhancement is suitable for quad-core processor systems where in all cases its operating system will view each CPU core as a separate processor. The algorithm takes two cores as slaves, which are seen to combine or detach from other cores. By letting slave cores be "Low level" processing units and other cores as "High level" processing units.

To schedule processes among High level and Low-level processing units, a burst time Range is calculated by finding the difference between the process with the highest burst time and the process with the lowest burst time. The range is compared to each process's burst time;

i. If the burst time is greater than or equal to the Range then that process is assigned to High level processing units.
ii. Else if the burst time is less than the Range that process is assigned to low level processing units.

The performance principals of the proposed algorithm are as follows:

i. The multiple core systems should have four or more cores.
ii. Round Robin technique is employed in both High and Low-level processing units.
iii. If High level processing units have completed executing processes and Low-level processing units still have ongoing processes then Low-level processing units' merges with the High-level units and they execute those processes as a single processing unit.
iv. If a new process arrives and it's found to be for Low level processing units, the Low-level units detaches itself from the High-level processing units leaving all executing processes to it so as to service the new process.
v. But if this new process is for the High-level processing units, the cores then resume as a single processing unit.
vi. If two or more processes arrive at the same time, a new Range is computed and used to schedule those processes among the High level and Low-level processing units.

The Roaming Core concept therefore implies that the Low-level processing units can merge or detach themselves from the High-level processing units thus keeping the processor busy and well utilized.

An analysis of the performance of the hybrid approach was conducted as below

Given following set of processes P1, P2, P3, P4, P5, and P6.

Process (P)	Burst time (ms)
P1	10
P2	8
P3	20
P4	5
P5	16
P6	6

Table 3.1 burst time of 6 processes under the hybrid scheduling approach

✓ *Burst time range (P3 – P4) = 15 ms.*
✓ *Each process' burst time is compared with the range and assigned appropriately to either High or Low level processing units as below.*

HIGH LEVEL (>= 15 ms)	LOW LEVEL (< 15 ms)
P3	P1
P5	P2
	P4
	P6

Table 3.2 burst time comparison and processor allocation

Suppose a new process Px arrives with a burst time of 18 ms, then Operating system will compare it with the burst time range. Since 18 ms is above the range, it will be assigned to high level processing units.

Also given that P8, P9, and P10 arrives at the same time with the following burst time.

Process (P)	Burst time
P8	12
P9	8
P10	5

Table 3.3 Burst time of 3 different processes under the hybrid scheduling approach

✓ A new burst time range will be computed *(Since two or more processes have arrived at the same time)* which will be (P8 – P10) = 7ms

✓ The processes will be assigned as follows.

HIGH LEVEL (>= 7ms)	LOW LEVEL (< 7 ms)
P8	P10
P9	

Table 3.4 Burst time comparison and processor allocation for the 3 different processes

The proposed algorithm can act as a solution to the problem of low priority processes being starved as long as the higher priority processes continue being active, to a point where low priority processes may never get service, since it able to assign higher priority processes to HIGH level processing units while low priority processes are given to LOW level processing units. A framework for its implementation is illustrated below:

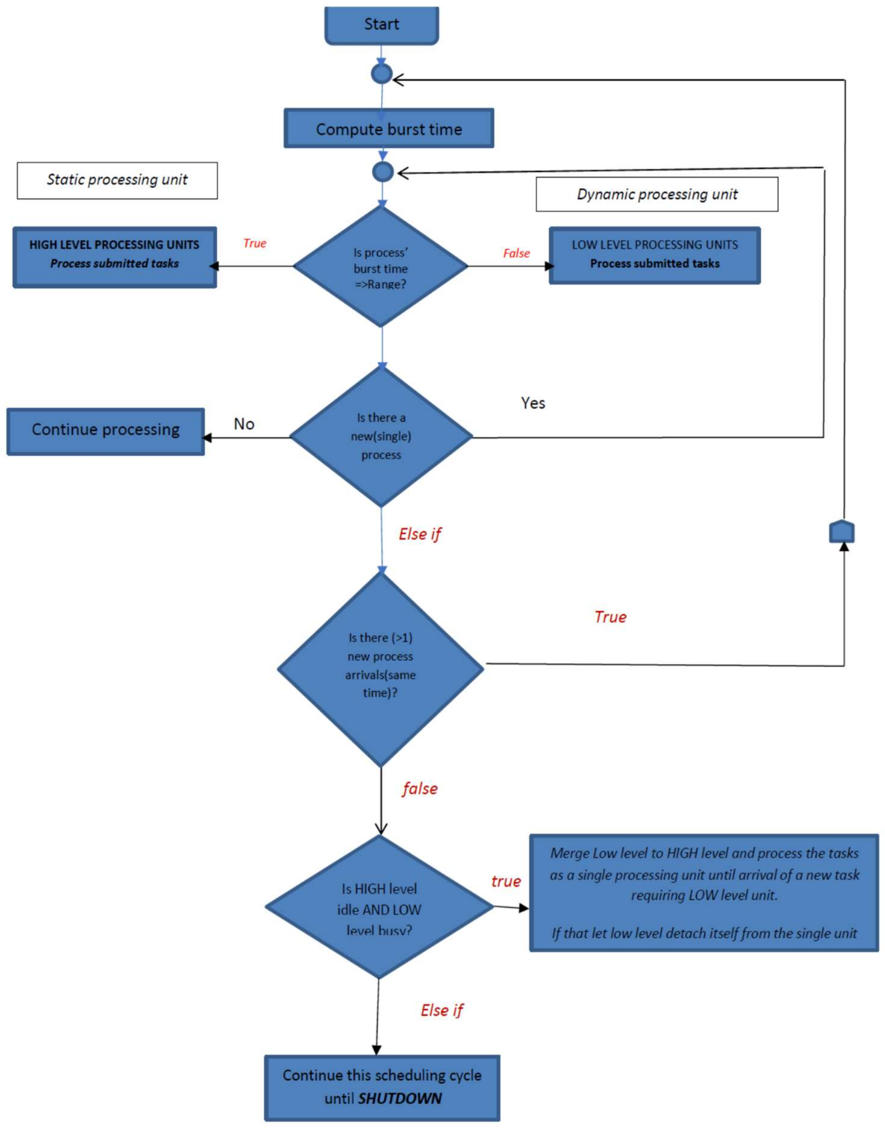
Start
Compute burst time
Static processing unit
Dynamic processing unit
HIGH LEVEL PROCESSING UNITS
Process submitted tasks
True
Is process' burst time =>Range?
False
LOW LEVEL PROCESSING UNITS
Process submitted tasks
Continue processing
No
Is there a new(single) process
Yes
Else if
Is there (>1) new process arrivals(same time)?
True
false
Is HIGH level idle AND LOW level busy?
true
Merge Low level to HIGH level and process the tasks as a single processing unit until arrival of a new task requiring LOW level unit.
If that let low level detach itself from the single unit
Else if
Continue this scheduling cycle until SHUTDOWN

Figure 3.1 A framework for its implementation a hybrid approach to Enhancing Process Scheduling in multiple core Systems

References

A. Tevanian, and M. Young, "Mach: A New Kernel Foundation for UNIX Development", Proceedings of the Summer USENIX Conference (1986), pages 93–112.

Abraham Silberschatz, Peter B. Galvin, Greg Gagne (2010) *Operating System Concepts with Java, 8th Edition* Wiley.

Ahmet Bindal (2017) *Fundamentals of Computer Architecture and Design*. Springer

D. L. Black, "Scheduling Support for Concurrency and Parallelism in the Mach Operating System", IEEE Computer, Volume 23, Number 5 (1990), pages 35–43.

Deitel, Harvey M.; Deitel, Paul; Choffnes, David (2015). Operating Systems. Pearson/Prentice Hall

Gerrit A. Blaauw, Frederick P Brooks, Jr. (1997) *Computer Architecture: Concepts and Evolution*. Addison-Wesley Longman Publishing Co.

IBM Systems Journal, Vol. 10, No. 3, c 1971, International Business Machines Corporation. Reprinted by permission of IBM Corporation.

Intel®Microprocessor Quick Reference Guide - www.intel.com. April 2016.

John Y. Hsu (2001) *Computer Architecture, Software Aspects, Coding, and Hardware*.

Kevin K. Leung (2005) *A Linux Process Scheduling Algorithm Animation Application*. California State University

M. Naghibzadeh (2005) *Operating System: Concepts and Techniques.* iUniverse

Maciej Drozdowski (2009) *Scheduling for Parallel Processing*. Springer

Pentium Processor User's Manual (1993) Architecture and Programming Manual, Volume 3.

Rakesh Kumar Phanden, Ajai Jain, J. Paulo Davim (2019) *Integration of Process Planning and Scheduling: Approaches and Algorithms*. CRC Press

Remzi H Arpaci-Dusseau, Andrea C Arpaci-Dusseau (2014) *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, LLC.

Tevanian et al. (1989) A. Tevanian, Jr., and B. Smith, "Mach: The Model forFuture Unix", Byte.