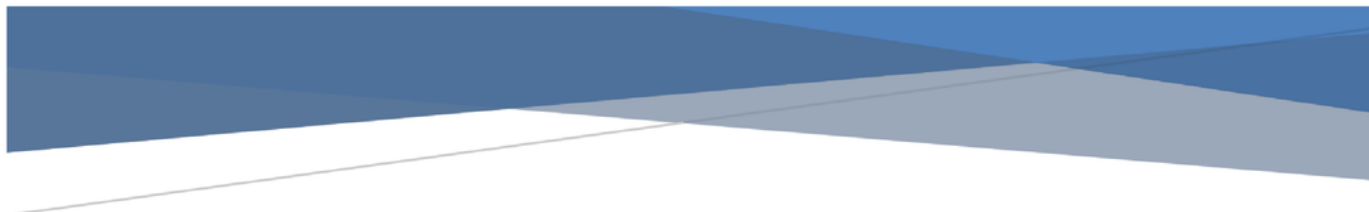


Assignment - Nick Hinder - p148207

by Nicholas Hinder

FILE	2927_NICHOLAS_HINDER_ASSIGNMENT_-_NICK_HINDER_-_P148207_95675_724911809.DOCX (119.11K)		
TIME SUBMITTED	25-JAN-2018 11:45AM (UTC+0000)	WORD COUNT	1862
SUBMISSION ID	82357439	CHARACTER COUNT	12356



RESEARCH REPORT

University of Wales Trinity Saint David

NICK HINDER

P148207

Table of Contents

Table of Contents.....	1
Introduction	2
Processes.....	2
Scheduling.....	2
Long term scheduler	2
Mid term scheduler.....	2
Short term scheduler	2
Algorithms.....	3
Inter-Process Communication	3
Threads	3
Scheduling.....	4
Communication.....	4
Synchronisation.....	4
Conclusions	5
Bibliography	6

Introduction

The CPU operates using a fetch-execute cycle whereby the processor reads data and instructions from memory and places them in its own registers for fast access during operations, once the operations are complete the data is returned to memory, allowing the registers to be used once again.

Multi-processing systems allow for multiple processes to execute concurrently despite the fact that only one CPU is present in a device. This is achieved through multiplexing the CPU between system processes through an implementation of time-slicing; this means that each process receives an allocated period of time (time-slice), in which the processor will perform the commands it requires, before the time-slice ends and another processes time-slice begins.

This report will provide an overview of processes and threads, exploring the scheduling and communication methods for each.

Processes

Each process in a multi-processing will exist in one of the following three statuses:

- Active; an active process is the process that is currently being executed by the processor, once the time-slice expires the next ready process will become the active process and the current process will become ready.
- Ready; a ready process requires a time-slice from the CPU as it is ready to execute commands.
- Waiting; a waiting process is one which will wait for input or access to some form of file. Once this has been occurred the process will shift to the ready status.

Scheduling

There are a number of algorithms used in order to facilitate multi-processing systems and there are often 3 different schedulers:

Long term scheduler

The long term scheduler is utilised in order to decide which processes are permitted to the ready queue, this scheduler can either allow or delay a process joining the ready queue and as such dictates what processes and how many of them can be running on a system.

Mid term scheduler

The mid term scheduler allows for processes to be removed from primary memory and to be placed in secondary memory and vice versa. This therefore allows for re-scheduling of processes by temporarily swapping them out or in.

Short term scheduler

The short term scheduler makes far more scheduling decisions than the long and mid term schedulers and operates on an interrupt level in order to facilitate immediate priority processes to run in spite of the previous schedule.

There are a great number of algorithms used for scheduling and each have different characteristics, however these can be assessed by their efficiency, throughput, turnaround time, waiting time, response time and fairness.

1

Algorithms

The simplest scheduling algorithm is the First Come First Serve which works on a simple queue, a process can be added to the tail end of the queue and each process prior to it will be executed before it. This algorithm is too simple and as such is actually a very inefficient solution with no prioritization of important processes.

Some other examples of scheduling algorithms are Shortest Job First scheduling, Round Robin scheduling and Priority scheduling. Each of these have both benefits and drawbacks inherent in them, the shortest job first algorithms reduce waiting time drastically, however they can lead to larger jobs never being processed due to a continuous stream of smaller jobs being available. Round robin scheduling is a fair approach, however the time interval of the round can be incorrectly set which can lead to inefficiencies and higher wait times. Priority scheduling executes higher priority processes first, which means that important processes receive increased time scheduling, however it also means that low priority processes may never be scheduled in deference to the higher priority processes.

Whilst each of these algorithms independently suffer substantial drawbacks, some form of meta-algorithm, utilizing key elements from various algorithms, would mitigate some of these drawbacks. For example a round robin algorithm with deference to very high priority processes would ensure that all processes receive adequate time slicing, however high priority processes will still have some level of preference over other scheduled processes.

3

Inter-Process Communication

Inter-process communication is required in a multi-processor system and can be performed in numerous methods, the most common of which is shared memory communication. This method means that processors actually share memory and this memory is used to communicate between the processes, however this can be slow due to the fact that only one process can access a piece of memory at a time, meaning the other process may have to wait to access the piece of memory before proceeding. This memory locking is very important however as without it a process could modify memory in use by another process, and this could lead to incorrect behavior. Other methods are serial-parallel, this is often used where processors have different communication links, serial-serial communication, this is used where processors have serial communication links, and parallel-parallel communication, this is used with special purpose processors that have parallel links and is seen as being the best inter-processor communication method.

Threads

A thread is a lightweight process that operates in its own right, however these lightweight processes are actually constituent parts of a larger process; this process would run multiple threads at the same time allowing for these threads to use the resources of the process and potentially lead to faster running

speeds for the user as the multiple threads run independently of one another, therefore if one is awaiting input, the other threads can continue to process.

Scheduling

Threads initially receive the scheduling of the parent process, however the schedule for each thread but there are a few methods through which these are modified to leverage the benefits of being a thread. Threads can be distributed over the processors within a multi-processor system to benefit from better load balancing; this also requires no further scheduling as the threads can operate simultaneously during the parent process's timeshare, however it is worth noting that due to the parallel nature care needs to be taken to avoid race conditions arising. Another potential scheduling method for multi-threaded applications is dedicated processor assignment; a thread is assigned a dedicated processor that will remain exclusive to the thread, this can speed operations of individual threads somewhat but can also mean that a number of processors can remain idle for some time. The final method of scheduling threads is dynamic scheduling; the number of threads in a process are altered dynamically and load is adjusted to optimize utilization.

Dynamic scheduling appears to be the optimal method as it leverages the advantages of load balancing and facilitates continuous, dynamic re-scheduling of jobs to ensure optimal running conditions.

Communication

Whilst each process shares nothing with another process, this is not the case with threads; threads share process resources with other threads that are also running within the process. An example of such a resource is a global variable; these shared resources are very useful tools to use whilst programming as all the threads will have access to the variable when desired and allow communication between threads. However the nature of these shared resources can mean that unexpected behavior can be exhibited if one thread modifies a resource value whilst another is using it or if another was expecting a previous value to still be there. This means that that multi-threaded applications need to be designed carefully in a way as to reduce potential for race conditions to arise.

Synchronisation

Due to the nature of multi-threading it is worth considering synchronization of threads; locking shared resources creates overheads which are not always required if there's no contention over access for these resources. If one thread utilizes a resource heavily (as a dominant thread) but rarely shares it with other threads this is known as asymmetric thread synchronization. Such synchronization utilizes biased locks whereby the dominant thread has a reduced overhead when accessing the shared data at the cost of the non-dominant threads exhibiting an increased overhead.

Another method of synchronization is through use of Transactional Memory systems (TM's). TM's utilize a conflict detection subsystem whereby to use shared resources a transaction must be created, this transaction is allocated a number which can be uniquely identified among other transactions. When a transaction conflict arises the conflict detection system will resolve the conflict by assessing the work done, number of retries and other properties of both transactions; one of the processes will be placed on a wait before retry or aborted whilst the other will be allowed to continue processing.

It appears that Transactional Memory systems are most favoured and in recent years there have been numerous improvements and iterations made upon its foundations such as token transactional memory which instead employs token coherency to detect conflicts.

Conclusions

Despite adding many layers of complexity to applications, multi-threading is too useful a tool to forego, and as such will be here to stay for the foreseeable future. Multi-threading allows for increased utilization of current systems potential by facilitating load sharing and parallel processing; due to the technology there are clearly observable increases in efficiency and processing time.

When working with multi-threaded applications, it is important that the usage of the multi-threading is appropriate for what is required, well implemented and will not lead to any issues down the line. Race-conditions and other such issues can be misleading and time consuming errors to pin down and maintain, these are best avoided by initial design.

However, despite being a mature technology at the time of writing, thread synchronization and scheduling continue to be areas in which research is carried out, as is evidenced by the recency of some papers on the topic. Research continues to improve and modify the algorithms in order to find potential improvements which could see use in future operating systems to improve efficiency and running costs.

Bibliography

1. **Blundell, Barry, et al.** *FastTrack: Computer Systems and Networks*. s.l. : Cengage Learning, 2007.
2. **Silberschatz, Abraham, Galvin, Peter Baer and Gagne, Greg.** *Operating system concepts*. 7th. s.l. : Wiley India Pvt. Limited, 2006.
3. **Goel, Neetu and Garg, R.B.** A Comparative Study of CPU Scheduling Algorithms. *International Journal of Graphics & Image Processing* . November, 2012, Vol. 2, 4.
4. **Dash, Amar Ranjan, Sahu, Sandipta kumar and Samantr, Sanjay Kumar.** AN OPTIMIZED ROUND ROBIN CPU SCHEDULING ALGORITHM WITH DYNAMIC TIME QUANTUM. *International Journal of Computer Science*. February, 2015, Vol. 5, 1.
5. **Sherman, Stephen, Baskett III, Forest and Browne, J.C.** Trace-Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System. *Communications of the ACM*. December, 1972, Vol. 15, 12.
6. **CLEARY, JIMMY, et al.** Fast Asymmetric Thread Synchronization. *ACM Transactions on Architecture and Code Optimization*. January, 2013, Vol. 9, 4.
7. **Junfeng, Yang, et al.** Making Parallel Programs Reliable with stable multithreading . *Communications of the ACM* . March, 2014, Vol. 57, 3.
8. **Hossain, M A and Tokhi, M O.** *INTER-PROCESSOR AND INTER-PROCESS COMMUNICATION IN REALTIME*. s.l. : IFAC Publications, 2002.
9. **Kamakoti, V. and Korgaonkar, Kunal.** Thread Synchronization: From Mutual Exclusion to Transactional Memory. *IETE Technical Review*. Jul - Aug, 2011, Vol. 28, 4.

Assignment - Nick Hinder - p148207

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

48/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4



Comment 1

Direct from source - cite.



Comment 2

Direct from source.



Comment 3

Make sure your spell checker is set to UK English - this is US spelling.

PAGE 5

PAGE 6

PAGE 7

GRADING FORM: SMU GENERIC_GJ

NICHOLAS HINDER

48

RESEARCH-INFORMED LITERATURE

 Your source list is excellent, composed exclusively of appropriate journal articles. The mark in this section is hugely limited, however, because there are no inline citations in this report. When you defer to a source, you must indicate that you have cited it, by putting the source's index into a square bracket after the material. [1] Similarly, make sure any material taken directly from a source is "quoted". We can have a chat about this to ensure your academic practice is up to speed for the next assignment.

15

KNOWLEDGE AND UNDERSTANDING

 Very good knowledge and understanding displayed - but a little brief in places. You're a fair bit shy of the word count, and it somewhat shows in the "Threads" section and its subsections in particular.

21

ANALYSIS

 A lack of synthesis evident, again due to the lack of inline citations. Conclusions, however, are appropriate and well-structured.

5

PRACTICAL APPLICATION AND DEP.

N/A - no marks assigned to this section.



SKILLS FOR PROFESSIONAL PRAC.



You write very well - the spelling, punctuation and grammar are very good in this work. Make sure your spellchecker is set to UK, however, you've mostly used US spelling in this work. A nicely-presented document, but don't be afraid to include some figures to help illustrate key points where appropriate.

7