

Research Report

by Ruaidhri David Cook

FILE	4148_RUAIDHRI_DAVID_COOK_RESEARCH_REPORT_95675_301722758.DOCX (32.68K)		
TIME SUBMITTED	25-JAN-2018 11:02PM (UTC+0000)	WORD COUNT	2500
SUBMISSION ID	82394047	CHARACTER COUNT	15950

Research Report

Name: Ruaidhri Cook

Student Number: P148970

Module: SYSTEM SOFTWARE & MULTI-USER APPLICATIONS ENGINEERING

Module Tutor: Dr Tim Bashford

Table of Contents

Introduction	3
What is a multi-programming system?.....	3
What are the basic computer operations?	3
Processes.....	3
What is a process?	3
How do multi-process systems work?	4
How do programs relate to processes?	4
Scheduling.....	4
Inter -Process Communication.....	5
Synchronisation.....	5
Threads	6
What is a thread?	6
Scheduling in relation to threads.....	6
Thread Communication	6
Thread Synchronisation	6
Summary	7

Introduction

This report is going to investigate multi-programming systems and how they interact with the hardware of a computer. The report will look at two key parts of multi-programming systems, these being Processes and Threads, and explore why they are important. The findings of this investigation will then be summarised, and an overview will be given of multi-programming systems.

What is a multi-programming system?

A multi-programming system can be described as a system that allows multiple applications to essentially run at the same time. The programs will appear to be running at the same time however it is impossible for programs to be executed simultaneously. The reason it appears these programs seems to run at the same time is because of interleaved execution, this means small parts of each program are executed consecutively. Part of this also involves executing instructions for one job (program) on the CPU whilst another job is doing an I/O(Input, Output) operation to increase efficiency and to ensure the CPU is utilized at its maximum potential.



What are the basic computer operations?

The basic operation that the computer performs is the 'fetch-execute' cycle.

The first part of this cycle is the fetch part, this involves the computer fetching an instruction from its main memory and storing it in the temporary memory of the CPU: the register(s). After the instruction has been fetched and stored it must then be decoded, this is done so the CPU can understand the instruction and thus execute it.

The second part of the cycle is the execution part, this can only happen once the instruction has been decoded and the CPU understands the action it needs to perform. A CPU will execute groups of instructions known as 'instruction sets' by performing an action. These actions are what make up the functions of a program.

Processes

What is a process?

A process, or 'job' as it is otherwise known, is an instance of a computer program that is being executed by the CPU. Each process has its own specific set of data and resources which are used to keep track of what is going on with the process, in addition to this the process has its own statistics that as associated with it, these statistics are used to make scheduling as efficient as possible. As previously mentioned in the section 'What are the basic computer operations?' the fetch-execute cycle is used to load the program's data and instructions from the disk so that the process can be executed. A process has multiple threads of execution to help complete the task, this is further explored in the section 'Threads'.

A process can assume one of three states, these states are:

- Waiting/Blocked – In this stage the process does not have the required resources available to run and is awaiting user input or a disk file.
- Ready – At this stage the process does have the required resources it needs to be executed however it is not active on the CPU for one reason or another.
- Running – In this state the process is code that is that is being executed on the CPU.

[1]

A process is moved between these states by the OS (Operating System), the process typically moves between the two states ready & running. When a process is transferred from the ready state to the running state this is referred to as scheduling or being scheduled, similarly when a process is moved from the running state to the ready state this is referred to as descheduling or being descheduled. A process can become blocked (be in waiting) as a result of a I/O operation being initiated and once blocked the process will stay like that until an event occurs, this event is often the completion of an I/O operation, which then prompts the OS to transfer the process back into the ready stage. [2]

How do multi-process systems work?

Multi-process systems differ from single-process systems as they offer an environment where multiple programs can run simultaneously. This can happen because multiple processes are able to be stored in memory and operations are carried out using microprocessors, which are essentially a tiny CPUs.

Multi-processing systems employ multiple processors to perform specific functions so that they can perform the task and then pass the instruction set onto another processor, thus freeing up the other processor to work on different functions. [3]

How do programs relate to processes?

Scheduling & Schedulers

Scheduling, as previously mentioned, is where a process is transferred from its ready state to its running state. This transfer is handled by a process manager that removes the running process and promptly selects another process to run on CPU in a strategic way that is beneficial to efficiency of running the program.[4]

With multi-programming operating systems processing scheduling is essential, without it a system would not be able load more than one process into the executable memory at any one time.

Scheduling queues are used to organise processes and can be described as a “single queue of processes or if multiple priority is Single queue of processes or if multiple priority is used, multiple priority queues, all feeding into a used, multiple priority queues, all feeding into a common pool of processors”[5]. When a process is first initialised on a system it is placed into the job queue. When the process enters the ready state, and is thus present in the main memory it will be added to the ready queue. The other type of queue is the device queue, processes will be placed here when there is not a device available to perform I/O operations.

There are three types of schedulers. These are:

- Short term scheduler – This type of scheduler is sometimes known as the CPU scheduler and is the most frequently run. The objective of this scheduler is to make the performance of the CPU more efficient and increase the rate at which processes are executed.
- Medium term scheduler – This type of scheduler is used to remove any processes from memory and swap in a different process, moving the original process to a secondary form of storage to be swapped back in at another time to complete the execution of the process.
- Long term scheduler – This type of scheduler also goes by the name of job scheduler and is usually used the most infrequently, however it still does an important job; deciding which programs get into the job queue. Its purpose is to keep a balanced degree of multiprogramming, this means lining up and equalizing the average rate of created processes to the average rate of departure from execution memory.

Inter-Process Communication

Inter-process communication (IPC) is an essential part of a multiprocessing application and can be described as means of exchanging data between different processes. IPC allows one application to assume control of another, therefore this it allows multiple applications to access and share data without impeding each other. IPC permits the communication of data, this is done by “allowing processes to use segments, semaphores, and other methods to share memory and information”. [5]

Synchronisation

The synchronisation of processes is needed when one process needs to wait for a different process to complete before it can be executed. This is especially needed with I/O operations, for example “one process (called a writer) may be writing data to a certain main memory area, while another process (a reader) may be reading data from that area and sending it to the printer”. [5] Process synchronisation makes sure that the system resources are handled so they can be shared in way that allows concurrently running processes to access the shared data whilst also minimising the chances of data becoming corrupted.

Synchronisation is also used to deal with issues caused by a multi-programming system, such as the following:

- Starvation – This is where a process is waiting and is being perpetually denied the resources it needs for the process to be executed.
- Priority inversion – This is where a higher priority task cannot execute because it cannot access the resource which is being withheld by lower priority task, usually because the lower priority task can't release the resource until a medium priority task is completed.
- Deadlock – This is a situation where two processes stop working because they are both using the same resource and by doing so stop each other from using the resource.
- Busy waiting – This is a situation where a process is continually asking if a condition is true, using up CPU power until the condition is met or the process is terminated.

Threads

What is a thread?

A thread can be compared with sequential programs, this is because it shares some properties that are the same, for example a thread will always have a single point of execution whilst it is running, moreover it has a beginning and an end but also a sequence. Threads do differ from sequential programs, this is because they are not actual programs in themselves but rather run as a part of a program.[6] A thread is made up of a thread ID, a stack, a program counter and a set of registers.[7]

A single thread will allow a program to execute a sequence of instructions, this means that with multiple threads it will allow a program to execute multiple of these sequences of instructions all at once, which is why multi-threading can increase the speed and efficiency of a computer. Moreover, threads can be more advantageous than processes for several reasons, the first of which is its communication (which will be explored later in the report), this is because there is overhead needed for communication because of the threads sharing everything, most importantly the address space. Another reason that threads can be advantageous in comparison to processes is the fact that it takes less time when terminating a thread than it does terminating a process.

When a process allows more than one thread to exist this is called 'multi-threading', this is different to single threading as the OS can recognize a thread as a concept and thus supports multiple threads of execution within a process.

Scheduling in relation to threads

Thread scheduling is like process scheduling, this is because it is the thread that utilises the processor. For this reason, when a process consists of multiple threads all these threads will be assigned to the processor. The threads that are assigned to the processor compete for its usage, using all the threads from the active processes that are being executed by the system. Much like with the process scheduling the threads will be shuffled around the scheduling queues.

Thread Communication

The communication with threads allows it to operate more effectively because there is little overhead; this means that data and resources are shared easily, for example all threads will have access to any data created by one thread almost instantaneously. A typical kind of thread communication can be the sharing of variables, sharing of this kind of data is far faster than communication between processes due to shared memory. If there is a scenario in which there are not enough processors to go around, the threads use an assigned integer to determine which has priority, with higher integers meaning a higher priority. Threads that are created will inherit the integer of the thread that created it to determine its priority. With this system, threads with a lower priority cannot be executed until the higher priority threads stop or complete their execution. In the case of threads with the same assigned priority one thread will be chosen at random.

Thread Synchronisation

Thread synchronisation is used to ensure there is sufficient collaboration between threads, this is done to optimise the efficiency of the program and make it more consistent. Thread synchronisation is needed because threads share data and thus situations can occur where two or more threads are both trying to access the same resource simultaneously which can cause issues with concurrency; the issues with concurrency could happen due to corruption of the data shared by the multiple

threads, this could happen if two threads were accessing the same file because one thread may be writing to the file whilst another thread is reading it which would cause the data to be overwritten midway through one of the threads reading from file. Thread synchronisation works by only allowing access to a resource one thread at a time, negating any issues associated with two threads trying to access the same resource.

Summary

To summarise the findings of the report, both Processes and Threads are essential to the running and operation of a multi-programming systems. This is not done solely by the threads and processes themselves however, as it has been shown they both make use of concepts to ensure that a program runs without crashing and at optimal performance. They both use scheduling, communication and synchronisation to negate any problems that are thrown up because of using a multi-programming system rather than a single-programming system.

References

- [1] Teach-ICT.com, "What is a process?" [Online]. Available: http://www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_1/scheduling/miniweb/pg2.htm.
- [2] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, "The Abstraction: The Process," *Oper. Syst. Three Easy Pieces*, p. 666, 2015.
- [3] The Editors of Encyclopædia Britannica, "Multiprocessing," *Multiprocessing*. [Online]. Available: <https://www.britannica.com/technology/multiprocessing>.
- [4] Tutorialspoint, "Operating System - Process Scheduling." [Online]. Available: https://www.tutorialspoint.com/operating_system/os_process_scheduling.htm. [Accessed: 23-Jan-2018].
- [5] R. Moona, "Multi - processor Scheduling," pp. 1–142.
- [6] E. A. Lee, "The problem with threads," *Computer (Long. Beach. Calif.)*, vol. 39, no. 5, pp. 33–42, 2006.
- [7] UNIVERSITY OF ILLINOIS CHICAGO, "Threads." [Online]. Available: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4_Threads.html. [Accessed: 20-Jan-2018].

Bibliography

- [1] BBC, "The CPU and the fetch-execute cycle," BBC, [Online]. Available: <https://www.bbc.co.uk/education/guides/zws8d2p/revision/3>. [Accessed 2018 January 7].
- [2] WhatIs.com, "Multiprogramming," WhatIs.com, [Online]. Available: <http://whatis.techtarget.com/definition/multiprogramming>. [Accessed 2018 January 7].
- [3] D. THAKUR, "Definition of Multiprogramming Operating System," EComputer Notes, [Online]. Available: <http://ecomputernotes.com/fundamental/disk-operating-system/multiprogramming-operating-system>. [Accessed 2018 January 8].
- [4] WhatIs.com, "Computer Operator," WhatIs.com, [Online]. Available: <http://whatis.techtarget.com/definition/computer-operator>. [Accessed 2018 January 9].
- [5] TeachICT.com, "What is a process?," TeachICT.com, [Online]. Available: http://www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_1/scheduling/miniweb/pg2.htm. [Accessed 12 January 2018].
- [6] M. Rouse, "Process," WhatIs.com, [Online]. Available: <http://whatis.techtarget.com/definition/process>. [Accessed 12 January 2018].

- [7] P. Krzyzanowski, "PK.org," The State University of New Jersey, 18 February 2015. [Online]. Available: <https://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html>. [Accessed 20 January 2018].
- [8] V. Bael, "Interprocess communication (IPC)," Webopedia, [Online]. Available: https://www.webopedia.com/TERM/I/interprocess_communication_IPC.html. [Accessed 23 January 2018].
- [9] University of Hamburg, "Thread Scheduling," University of Hamburg, [Online]. Available: <https://www.math.uni-hamburg.de/doc/java/tutorial/essential/threads/priority.html>. [Accessed 24 January 2018].
- [10] T. H. Cameron Hughes, "Dividing C++ Programs into Multiple Threads," Informit, 20 February 2004. [Online]. Available: <http://www.informit.com/articles/article.aspx?p=169479&seqNum=3>. [Accessed 22 January 2018].
- [11] Revolv, "Starvation (computer science)," Revolv, [Online]. Available: [https://www.revolv.com/main/index.php?s=Starvation%20\(computer%20science\)](https://www.revolv.com/main/index.php?s=Starvation%20(computer%20science)). [Accessed 24 January 2018].

Research Report

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

54/100

PAGE 1

PAGE 2

PAGE 3



Comment 1

This is true for single core, though of course, not for multi-core systems.



Comment 2

Use UK English

PAGE 4



Comment 3

This is a possessive, not a contraction - no apostrophe.



Comment 4

This one too.

PAGE 5



Comment 5

Far too much hangs on this single source.



Contraction

Don't use contractions in academic writing. E.g. you should use "do not" instead of "doesn't".

PAGE 6



Comment 6

Would've been nice to have a bit more detail here - e.g. you could talk about specific synchronisation methods, like mutexes.

GRADING FORM: SMU GENERIC_GJ

RUAIDHRI DAVID COOK**54****RESEARCH-INFORMED LITERATURE**

A reasonable application of literature in the construction of your work. You have a relatively small reference list, but a far more substantial bibliography. The sources in the reference list are of mediocre quality (at least 2, 5, 6 and to a lesser extent 3 and 7 seem OK), while the bibliography seems a tad weaker. I'd actually rather see a lightly longer reference list, with higher academic credibility - the bibliography is of far lesser value. Some of your references are incomplete. Access dates for online sources? Dates for things which have them? In application to your essay, you've generally done well. It'd be nice to see more use of multiple sources to develop a single point (I'll talk about this more in the analysis section), and in a few places, far too much hangs on the veracity of a single source - 5 in particular gets used very heavily, and while that's fine, it'd be nice to see some additional sources to back this up.

27**KNOWLEDGE AND UNDERSTANDING**



Good knowledge and understanding evident - you've obviously done a lot of reading around the topic. A few sections are a tad brief, and you do have a few hundred or so words to play with, while remaining in the count (removing titles, reference lists etc).

18

ANALYSIS



A point for major improvement is in the synthesis of material. As a rule try to read multiple sources on the same topic - form them into a unified argument, then cite them all, e.g. [5][6][7][8]. It's nice to see a conclusion as suggested in the brief, though you could actually go a little further, and consider including some concluding remarks at the end of sections etc. - this would help your work flow together better.

4

PRACTICAL APPLICATION AND DEP.



N/A - no marks assigned.

SKILLS FOR PROFESSIONAL PRAC.



Fair spelling, punctuation and grammar. Be careful with your use of apostrophes, semi-colons, and endeavour to avoid run-on sentences. The document is fairly nicely-presented.

5