Volume: 2 Issue: IX Month of publication: September 2014
DOI:

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

Threads in Operating System

Kirti Sharma[1], Himanshu Saini[2], Girish Mehta[3]

[1,2,3] *Student (B.tech Vth sem) Department of Computer science*

Dronacharya College Of Engineering,Gurgaon-123506

***Abstract*-This paper basically deals with threads used in an operating system. We have focused on the working and the ways of multithreaded system, how they are used to write a program in an efficient and effective way. The first part of the paper explains you what is an operating system and its history. And the second part emphasizes the concepts of Threads in an operating system(OS).**

Keywords: Threads, multithread, effective, efficient, OS.

I. INTRODUCTION

An operating system (OS) is software that controls computer hardware and software resources and provides common services for computer program.. The operating system is an essential component of the software program in a computer system. Application programs usually require an operating system to function. For hardware functions such as input and output and memory allocation the operating system acts as an intermediary between programs and the computer hardware, although the application code is usually executed directly by the hardware and will frequently make a system call to an OS function or be interrupted by it.

Examples of popular modern operating systems include Android, ios, Linux, Microsoft windows and many more.

II. HISTORY

Operating systems have evolved through a number of distinct phases or generations which corresponds roughly to the decades.

A. The 1940's - First Generations

The earliest electronic digital computers had no operating systems. Machines of the time were so primitive that programs were often entered one bit at time on rows of mechanical switches (plug boards). Programming languages were unknown (not even assembly languages). Operating systems were unheard of .

B. The 1950's - Second Generation

By the early 1950's, the routine had improved somewhat with the introduction of punch cards. The General Motors Research Laboratories implemented the first operating systems in early 1950's for their IBM 701. The system of the 50's generally ran one job at a time. These were called single-stream batch processing systems because programs and data were submitted in groups or batches.

C. The 1960's - Third Generation

The systems of the 1960's were also batch processing systems, but they were able to take better advantage of the computer's resources by running several jobs at once. So operating systems designers developed the concept of multiprogramming in which several jobs are in main memory at once; a processor is switched from job to job as needed to keep several jobs advancing while keeping the peripheral devices in use.

D. Fourth Generation

With the development of LSI (Large Scale Integration) circuits, chips, operating system entered in the system entered in the personal computer and the workstation age. Microprocessor technology evolved to the point that it become possible to build desktop computers as powerful as the mainframes of the 1970s.

III. THREADS

A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes.

In many respect, threads are popular way to improve application through parallelism. The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel. Like a traditional process i.e., process with one thread, a thread can be in any of several states . Each thread has its own stack. Since thread will generally call different procedures and thus a different execution history. This is why thread needs its own stack. An operating system that has thread facility, the basic unit of CPU utilization is a thread.

A thread is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers.

- Traditional processes have a single thread of control - There is one program counter, and one sequence of instructions that can be carried out at any given time.
- As shown in Figure 4.1, multi-threaded applications have multiple threads within a single process, each having their own program counter, stack and set of registers, but sharing common code, data, and certain structures such as open files.

We use threads due to many reasons. Threads plays a key role in the designing of an operating system. A process with multiple threads make a great server for example printer server. Because threads can share common data, they do not need to use inter process communication .Because of the very nature, threads can take advantage of multiprocessors.

Threads are cheap as They only need a stack and storage for registers therefore, threads are cheap to create. Threads use very little resources of an operating system in which they are working. That is, threads do not need new address space, global data, program code or operating system resources. Context switching are fast when working with threads. The reason is that we only have to save and/or restore PC, SP and registers.

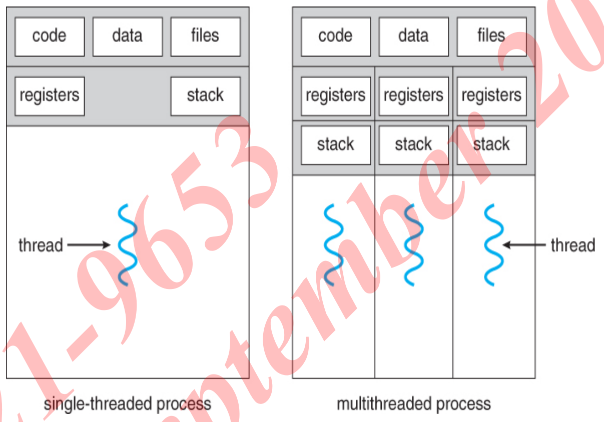

Fig. 1 Single-threaded processes and Multithread processes

IV. MULTI THREAD PROCESS

Multithreading is mainly found in multitasking operating systems. Multithreading is a widespread programming and execution model that allows multiple threads to exist within the context of a single process. These threads share the process's resources, but are able to execute independently. The threaded programming model provides developers with a useful abstraction of concurrent execution. Multithreading can also be applied to a single process to enable parallel execution on a multiprocessing system.

- There are two types of threads to be managed in a modern system: User threads and kernel threads.
- User threads are supported above the kernel, without kernel support. These are the threads that application programmers would put into their programs.
- Kernel threads are supported within the kernel of the OS itself. All modern OSes support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

- In a specific implementation, the user threads must be mapped to kernel threads, using one of the following strategies.

A. Many to one

Where many user level threads are mapped to as single kernel thread. In such model the thread is managed by thread library in the user space which makes such model efficient. However the entire threads can be blocked if a thread makes a blocking system call. Also; with multiprocessors system; threads can access the kernel with only a single thread at a time and as such; multiple threads are unable to take advantage on multiprocessor and run in parallel on multiprocessors. With such model, developer will be able to create as many threads as required; however the true concurrency is not implemented since the kernel can schedule only one thread at a time.

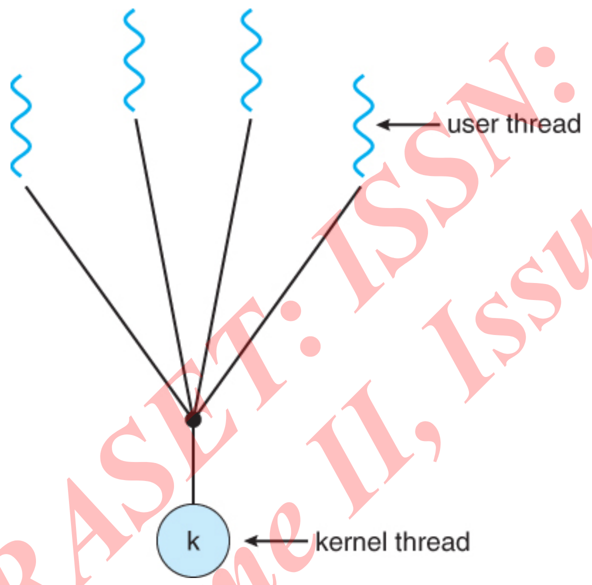

Fig. 2 Many to One thread

B. One to one

Where each user thread is mapped to the kernel thread. Such mechanism provides concurrency operation than the many-to –many model where other threads are allowed to even when one of these threads makes a blocking system call. Also such system allows multiple threads to run in parallel on multiprocessors. Since such model creates a corresponding kernel thread with every creating of user thread, on overhead of creating kernel threads can affect the performance of an application and as such this implementation can be restricted with the number of threads that can be supported be the system.

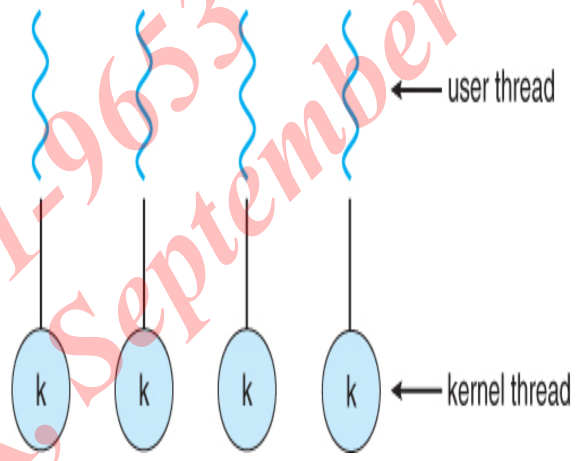

Fig. 3 One to One Thread

C. Many to many

The many to many model multiplexes any number of user threads onto an equal or smaller , number of kernel threads , combining the best of the one-to –one and many to one models.

Users have no restrictions on the number of threads created .Blocking kernel system calls do not block the entire process.

Processed can be split across multiple process. Individual processes may be allocated variable number of kernel threads.

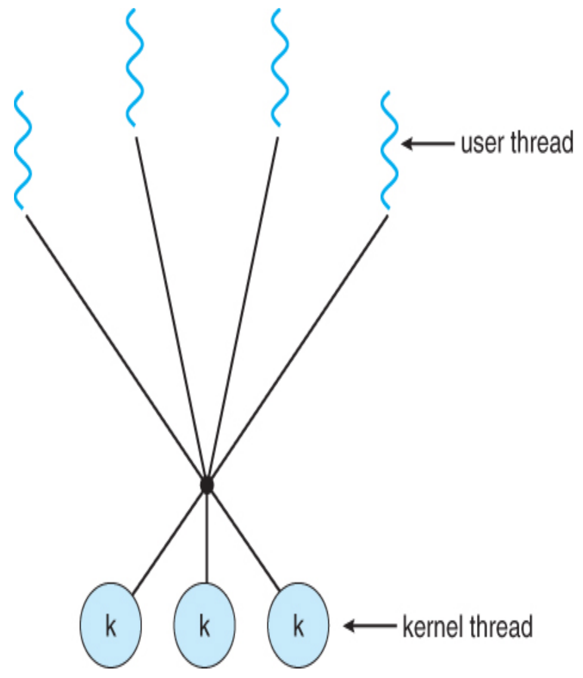

Fig. 4 Many to many Thread

V. ADVANTAGES

Threads are very useful in modern programming whenever a process has multiple tasks to perform independently of the others This is particularly true when one of its task may block, and it is desired to allow the other tasks to proceed without blocking.

For example in award processor, background thread may check spelling and grammar while a foreground thread processes a user input , while yet a third thread loads images from the hard drive , and the fourth does periodic automatic backups of the file being edited.

Another example is a web server-

Multiple threads allow multiple requests to be satisfied simultaneously, without having to service requests sequentially or to fork off separate processes for every incoming request.

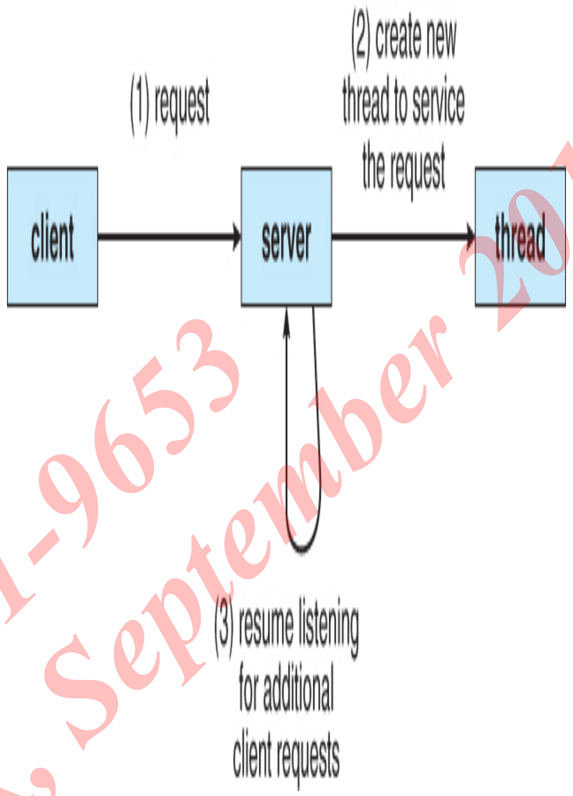
(1) request
(2) create new thread to service the request
client
server
thread
(3) resume listening for additional client requests

Fig. 5 Resume listening for additional client requests

VI. CONCLUSION

A thread is a flow of control with a process and it is more efficient and more productive for a process to have multiple threads to achieve the maximum efficiency of any computing system (Titus, 2004). For example, with a server that can support multithreaded processes, such server can create several threads based on the client's requests. Multithreading allows application to be more interactive since the program can continue running even when part of such program's thread is blocked or is involved in a lengthy operation. Also, with multiprocessor architecture that is exist in modern computing system, different threads can run in parallel on different processors .

References

[1] F.J. Cazorla, P.M.W. Knijnenburg, R. Sakellariou,E. Fernandez, A. Ramirez, and M. Valero, “Predictableperformance in SMT processors: synergy between theOS and SMTs,” Computers, IEEE Transactions on, vol 55, no. 7, pp. 785 – 799, july 2006.

[2] F.N. Sibai, “Performance effect of localized thread schedules in heterogeneous multi-core processors,” in Innovations in Information Technology, 2007. IIT ’07.4th International Conference on, nov. 2007, pp. 292 –296.

[3] O. Mutlu and T. Moscibroda, “Parallelism-aware batch scheduling: Enabling high-performance and fair shared memory controllers,” Micro, IEEE, vol. 29, no. 1, pp.22 –32, jan.-feb. 2009.