Raman Shukla

2020201098

AOS Assignment 1

Threads

Thread is a basic unit of CPU utilization. As the process was the instance of a program in execution, a process can also be assumed as a single thread if multithreading is not present in that system. Thus, a thread can be considered as the flow of control / execution of a program with some level of independence with respect to its peer threads.

A process, gets allocated memory in 4 parts:

* + Code Segment: Where all the code written in the program is stored.
  + Data Section: Where global and static variables are stored.
  + Stack Section: Where local variables are stored and this part is utilized by the user for recursive programs.
  + Heap Section: Where dynamic allocation takes place.

Similarly, a thread is also assigned the above types of memory, but shared by its peers if any. A thread only gets its own registers, Program Counter, Thread Id, and Stack. The rest of the memory and resources are shared among the threads.

The picture [3] illustrates how threads share memory. Earlier to execute, the similar function, we used fork system call, but it was very taxing for the memory as well as the CPU to make new copy and context switch among them. Therefore, threads are chosen for doing similar tasks multiple time.  
For example, writing the code of a server, the request handling can also be done by creating a new process every time a new client connects, but a better approach would be to create a new thread for new client connection.

User Thread and Kernel Thread

User threads are supported in the user space. They are needed to give a program some independence but cannot provide any parallelism, as the user threads distribute the time quantum given to a single process amongst themselves. Since, the OS is not informed of these changes made by the process, it will consider all the user threads as a part of single process. User threads since are operating in user space need the help of kernel threads for performing tasks which are below the user level. If one thread gets blocked then rest thread are also blocked, since they are considered as a single process by the OS.

Kernel threads are generated by the OS, to handle the kernel space tasks. Since, the OS recognizes this thread, it can be scheduled accordingly by the Scheduler. Unlike, user thread there is more overhead in using Kernel threads and system call are required. We use them for frequently blocking tasks.

In conclusion, user threads are independent of the OS whether the OS supports multithreading or not. Since, it does not require informing the OS while creating the user thread, they tend to be a lot faster than the kernel thread.

The benefit of kernel level thread during execution is that they are created by OS and thus can be allocated time efficiently as well as will not be blocked if its peers are blocked.

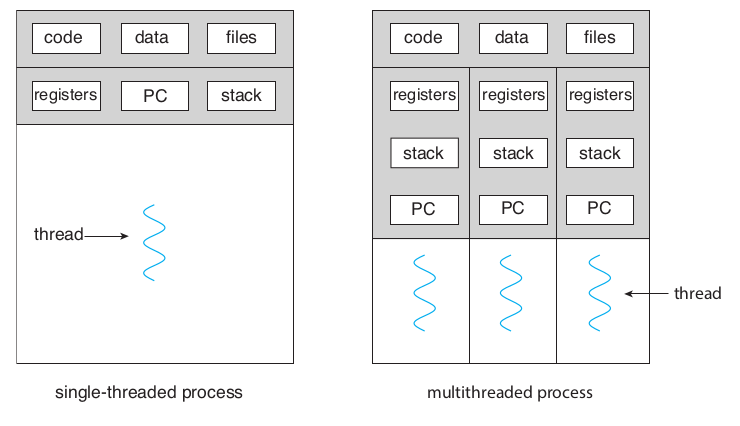
Concurrency

A concurrent system supports more than one task allowing all the task to make progress while a parallel system supports more than one task simultaneously. The concurrency in threads depends upon the architecture of the relation between kernel and user threads.[1]

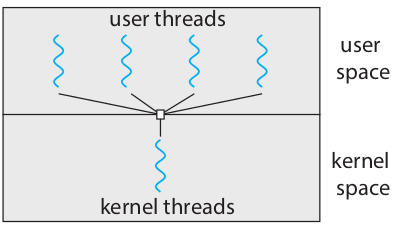
Many to one model [3]: Multiple user threads are mapped to a single kernel thread.In this case parallelism cannot be achieved and concurrency too cannot be achieved for the blocking calls.

One to one Model [4]: Every user thread is mapped to a kernel thread. This increased the concurrency as other threads can run even if one is blocked. But it requires same amount of kernel threads thus increasing the overhead.

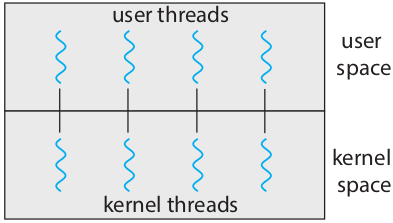
Many to Many Model [5]: User threads are multiplexed with kernel threads. They can choose from the already present fixed number of kernel threads. The kernel threads numbers are dependent on program or the OS. This overcomes both the shortcomings of above two models, if one kernel thread is blocked scheduler can assign other kernel thread to the user thread.



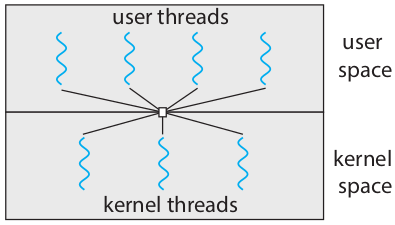
[3]



[4]



[5]



[6]

References

* 1. Operating System Concepts ISBN 978-1-119-32091-3 [Chapter 4]
  2. <http://www.cs.iit.edu/~cs561/cs450/ChilkuriDineshThreads/dinesh%27s%20files/User%20and%20Kernel%20Level%20Threads.html>