

Operating System Notes

CSE - 309, Semester - 9

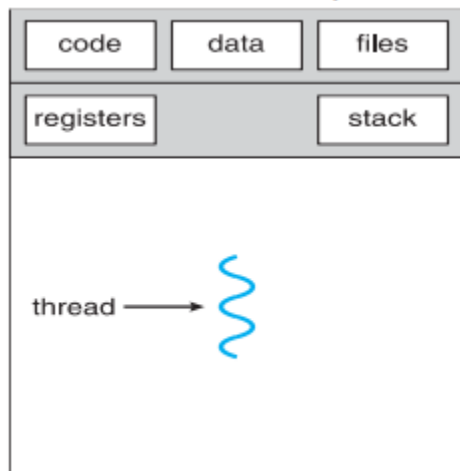
Written by Md. Rakibur Rahman

BSc. in CSE, Dhaka International University

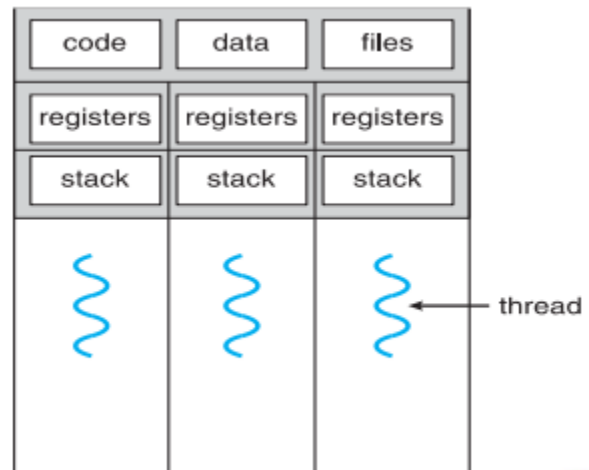
1. Single thread & multithread process with figures.

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

Single Thread: Traditional (heavyweight) 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.



single-threaded process



multithreaded process

Multi Thread: 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.

2. Challenges that multi-core chips present for application programmers.

There are five areas where multi-core chips present new challenges for application programmers:

Identifying tasks - Examining applications to find activities that can be performed concurrently.

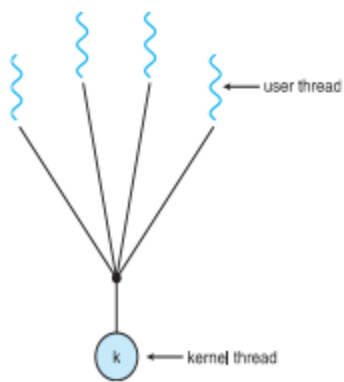
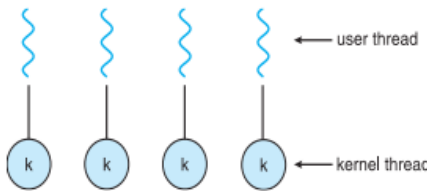
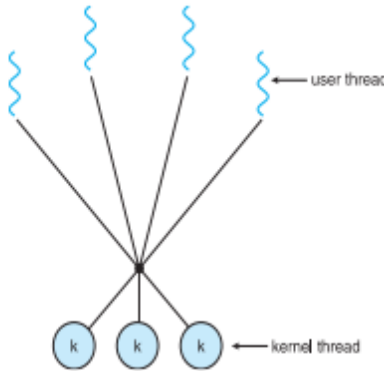
Balance - Finding tasks to run concurrently that provide equal value.

Data splitting - To prevent the threads from interfering with one another.

Data dependency - If one task is dependent upon the results of another, then the tasks need to be synchronized to assure access in the proper order.

Testing and debugging - Inherently more difficult in parallel processing situations, as the race conditions become much more complex and difficult to identify.

3. Different types of multithreading models with appropriate figures.

<p>Many-to-One</p> <ul style="list-style-type: none"> • Many user-level threads mapped to single kernel thread • One thread blocking causes all to block • Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time • Few systems currently use this model • Examples: Solaris Green Threads, GNU Portable Threads 	<p>One-to-One</p> <ul style="list-style-type: none"> • Each user-level thread maps to kernel thread • Creating a user-level thread creates a kernel thread • More concurrency than many-to-one • Number of threads per process sometimes restricted due to overhead • Examples Windows, Linux, Solaris 9 and later 	<p>Many-to-Many Model</p> <ul style="list-style-type: none"> • Allows many user level threads to be mapped to many kernel threads • Allows the operating system to create a sufficient number of kernel threads • Solaris prior to version 9 Windows with the ThreadFiber package 
--	---	--

4. Definition of Dispatcher, Dispatcher Latency, throughput and turnaround time.

Dispatcher: Dispatcher is responsible for loading the selected process by Short Term scheduler on the CPU (Ready to Running State). Short-term scheduler is also known as a dispatcher.

Dispatch latency – The time it takes for the dispatcher to stop one process and start another running

Throughput – The number of processes that complete their execution per time unit (may be in seconds, milliseconds, or hours).

Turnaround time – The amount of time to execute a particular process; i.e., waiting time + execution time

5. Priority & Roun Robin Algorithm Math.

6. Process Synchronization, Critical Section, Critical Section Problem Definition and Conditions to solve Critical Section Problems.

Process Synchronization: When two or more process cooperates with each other, their order of execution must be preserved otherwise there can be conflicts in their execution and inappropriate outputs can be produced.

Critical Section: The critical section refers to the segment of code where processes access shared resources, such as common variables and files, and perform write operations on them. Since processes execute concurrently, any process can be interrupted mid-execution.

Critical Section problem: The critical section problem is to make sure that only one process should be in a critical section at a time. When a process is in the critical section, no other processes are allowed to enter the critical section.

Conditions to solve Critical Section Problems

i. Mutual Exclusion- If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.

ii. Progress- If no process is executing in its critical section and there exists some processes that wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision of which will enter its critical section next,

- if no process is in critical section, can decide quickly who enters
- only one process can enter the critical section so in practice, others are put on the queue.

iii. Bounded Waiting- There must exist a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted

- The wait is the time from when a process makes a request to enter its critical section until that request is granted.
- in practice, once a process enters its critical section, it does not get another turn until a waiting process gets a turn (managed as a queue)

7. Peterson's Solution for critical problem solution with disadvantages.

Peterson's Solution is a classical software-based solution to the critical section problem. In Peterson's solution, we have two shared variables.

Peterson's Solution preserves all three conditions:

Mutual Exclusion is assured as only one process can access the critical section at any time.

Progress is also assured, as a process outside the critical section does not blocks other processes from entering the critical section.

Bounded Waiting is preserved as every process gets a fair chance.

Disadvantages of Peterson's Solution:

- It involves Busy waiting
- It is limited to 2 processes.

LECTURE - 7

1. Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Consider an example when two trains are coming toward each other on same track and there is only one track, none of the trains can move once they are in front of each other. Similar situation occurs in operating

systems when there are two or more processes hold some resources and wait for resources held by other(s).

2. **Conditions that turn a system into deadlock:**

If a graph contains no cycles => no deadlock

If the graph contains a cycle =>

- if only one instance per resource type, then deadlock
- if several instances per resource type, possibility of deadlock

3. **Deadlock Prevention**

Mutual Exclusion – not required for sharable resources; must hold for nonsharable resources

Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources

- Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none
- Low resource utilization; starvation possible

No Preemption –

- If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
- Preempted resources are added to the list of resources for which the process is waiting
- Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting

Circular Wait – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration