CSC 301 – OPERATING SYSTEMS II (2 UNITS) LECTURE 1

BALOGUN JEREMIAH ADEMOLA

ASSISTANT LECTURER,

DEPARTMENT OF COMPUTER SCIENCE AND MATHEMATICS

MOUNTAIN TOP UNIVERSITY, OGUN STATE, NIGERIA

COURSE OUTLINE

- **Concurrency**
  - States and State Diagrams Structures
  - Dispatching and Context Switching
  - Interrupts
  - Concurrent Execution
  - Mutual Exclusion Problem and Some Solutions
- **Deadlock**
  - Models and Mechanisms (Semaphores, monitors etc.)
  - Producer – Consumer Problems and Synchronization
  - Multiprocessor Issues
- **Scheduling and Dispatching, Memory Management**
  - Overlays, Swapping and Partitions
  - Paging and Segmentations Placement and Replacement Policies
  - Working Sets and Trashing
  - Caching

THREADS

- A **thread** is a path which is followed during a program's execution.
  - Majority of programs written now a days run as a single thread.
- Lets say, for example a program is not capable of reading keystrokes while making drawings.
  - These tasks cannot be executed by the program at the same time.
  - This problem can be solved through multitasking so that two or more tasks can be executed simultaneously.
- Threading is used widely in almost every field.
  - Most widely it is seen over the internet now days where we are using transaction processing of every type like recharges, online transfer, banking etc.
- Threading is a segment which divide the code into small parts that are of very light weight and has less burden on CPU memory so that it can be easily worked out and can achieve goal in desired field.
  - The concept of threading is designed due to the problem of fast and regular changes in technology and less the work in different areas due to less application.
  - Then as says "need is the generation of creation or innovation" hence by following this approach human mind develop the concept of thread to enhance the capability of programming.

THREADS...

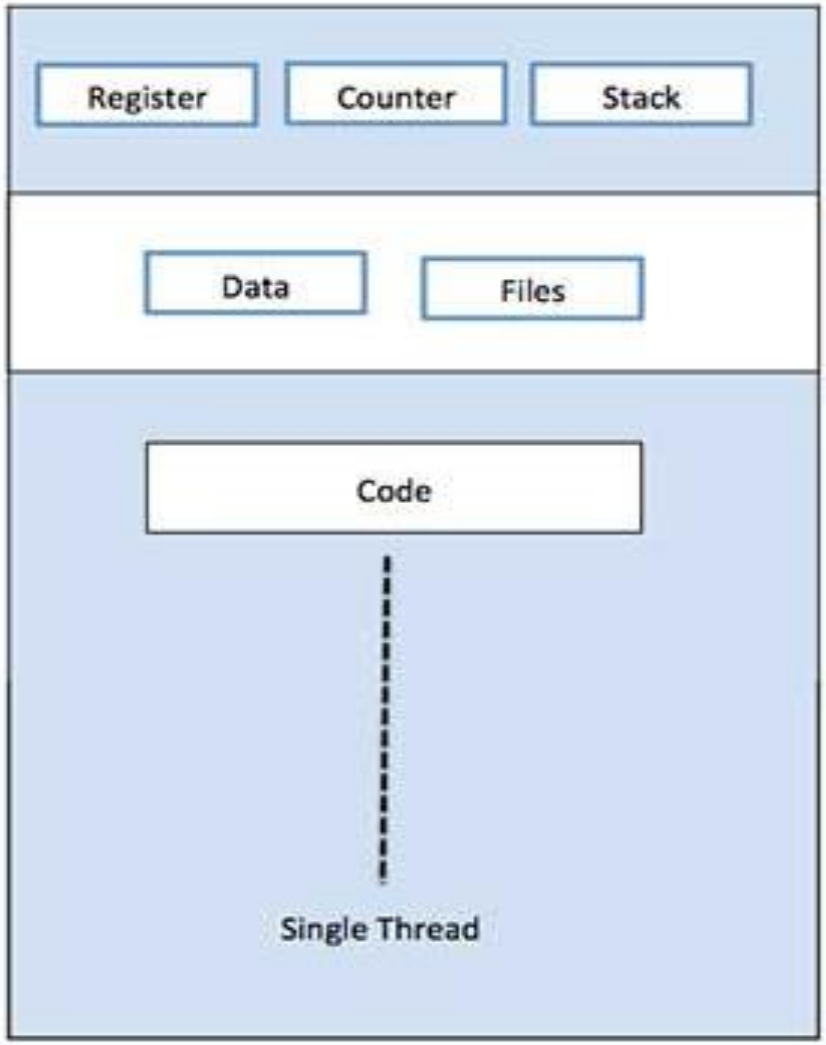
- A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

- A thread shares with its peer threads few information like code segment, data segment and open files.
  - When one thread alters a code segment memory item, all other threads see that.

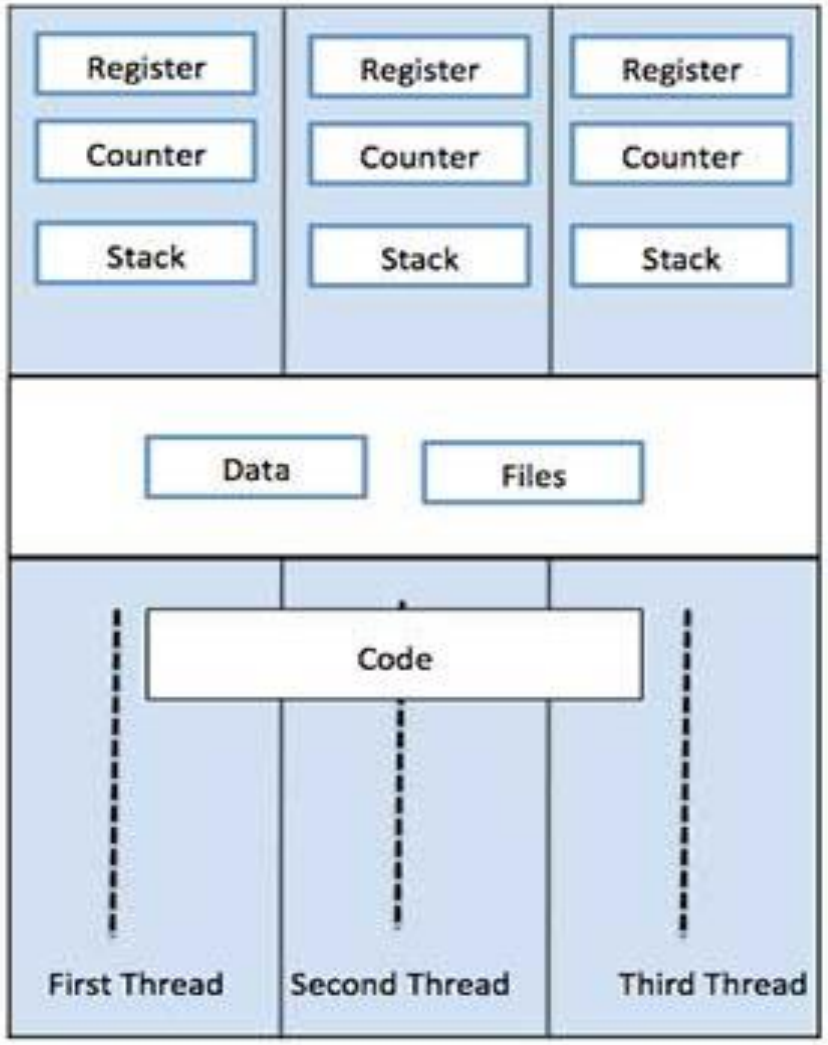
- A thread is also called a lightweight process.
  - Threads provide a way to improve application performance through parallelism.

- Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.
  - Each thread belongs to exactly one process and no thread can exist outside a process.
  - Each thread represents a separate flow of control.
  - Threads have been successfully used in implementing network servers and web servers.
  - They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors.

CLASSIFICATION OF THREADS



Single Process P with single thread



Single Process P with three threads

PROCESSES AND THREADS

S/N	Process	Thread
1	Process is heavy weight or resource intensive	Thread is light weight, taking lesser resources than a process
2	Process switching needs interaction with OS	Thread switching does not need to interact with OS
3	In multiple processing environments, each process executes the same code but has its own memory and file resources	All threads can share same set of open files, child processes
4	If one process is blocked, then no other process can execute until the first process is unblocked	While one thread is blocked and waiting, a second thread in the same task can run
5	Multiple processes without using threads use more resources	Multiple threaded processes use fewer resources
6	In multiple processes each process operates independently of the others	One thread can read, write or change another thread's data.

ADVANTAGES OF THREADS

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

- **Types of Threads**
  - **User Level Threads** – user managed threads
  - **Kernel Level Threads** – OS managed threads acting on kernels, an OS core.

USER-LEVEL THREADS

- In this case, the thread management kernel is not aware of the existence of threads.
  - The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts.
  - The application starts with a single thread.

- **Advantages**
  - Thread switching does not require Kernel mode privileges
  - User level thread can run on any OS
  - Scheduling can be application specific in the user level thread
  - User level threads are fast to create and manage

- **Disadvantages**
  - In a typical OS, most system calls are blocking
  - Multithreading applications cannot take advantage of multiprocessing.

KERNEL-LEVEL THREADS

- In this case, the thread management performed by the kernel.
  - There is no thread management code in the application area.
  - Kernel threads are supported directly by the OS.
  - Any application can be programmed to be multithreaded.
  - All of the threads within an application are supported within a single process.
  - The kernel maintains context information for the process as a whole and for individual threads within the process.
  - Kernel threads are generally slower to create and manage compared to user level threads.

- **Advantages**
  - Kernels can simultaneously schedule multiple threads from the same process on multiple process.
  - If one thread in a process is blocked, the kernel can schedule another thread of the same process.
  - Kernel routines themselves can be multithreaded.

- **Disadvantages**
  - Kernel threads are generally slower to create and manage compared to user threads.
  - Transfer of control from one thread to another within the same process requires a model switch.

MULTITHREADING

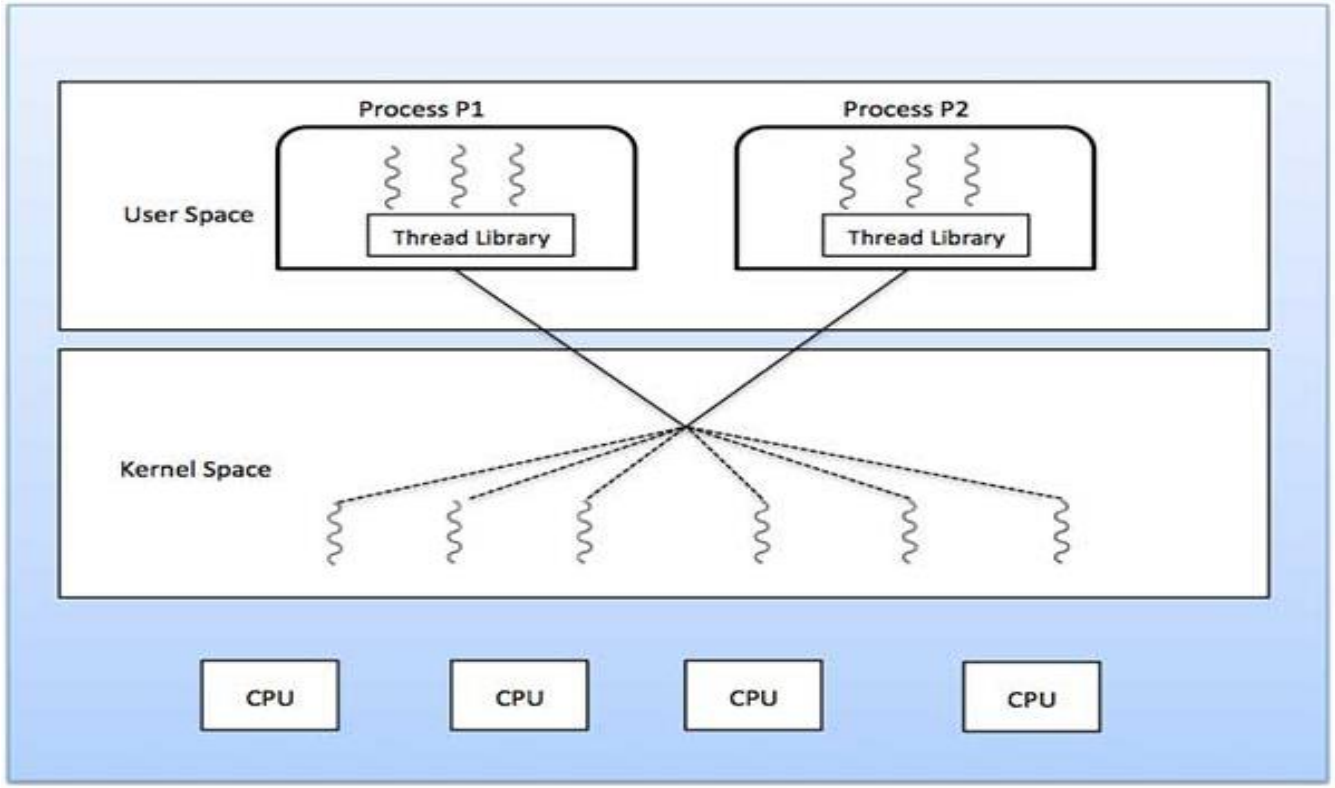
- Multithreading is similar to multitasking, but enables the processing of multiple threads at one time, rather than multiple processes.
  - Since threads are smaller, more basic instructions than processes, multithreading may occur within processes.

- By incorporating multithreading, programs can perform multiple operations at once.
  - For example, a multithreaded operating system may run several background tasks, such as logging file changes, indexing data, and managing windows at the same time.
  - Web browsers that support multithreading can have multiple windows open with JavaScript and Flash animations running simultaneously.
  - If a program is fully multithreaded, the different processes should not affect each other at all, as long as the CPU has enough power to handle them.

- Similar to multitasking, multithreading also improves the stability of programs.
  - However, instead of keeping the computer from crashing, multithreading may prevent a program from crashing.
  - Since each thread is handled separately, if one thread has an error, it should not affect the rest of the program.
  - Therefore, multithreading can lead to less crashes, which is something we can all be thankful for.

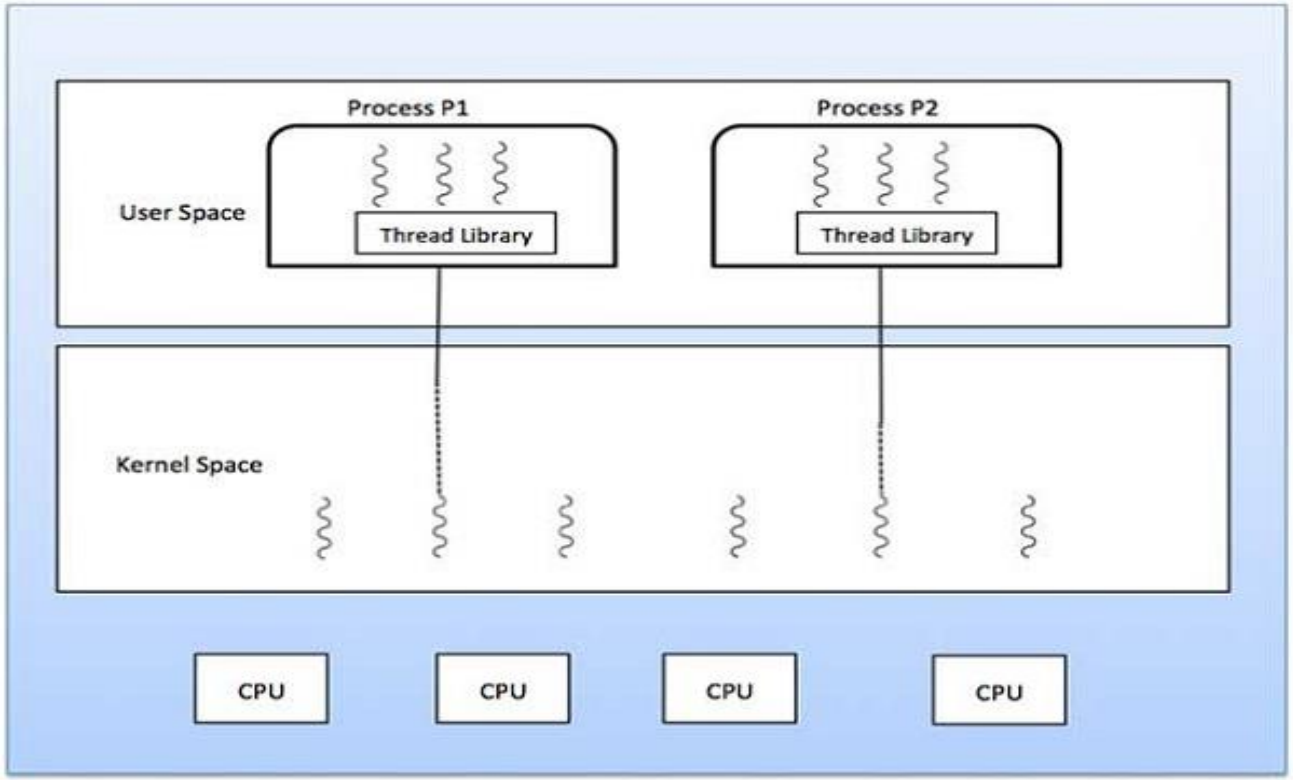
MULTITHREADING MODELS – MANY TO MANY MODEL

- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.
- In the diagram below, 6 user level threads are multiplexing with 6 kernel level threads.
  - In this model, developers can create as many user threads as necessary and the corresponding kernel threads can run in parallel on a multiprocessor machine.
  - This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.



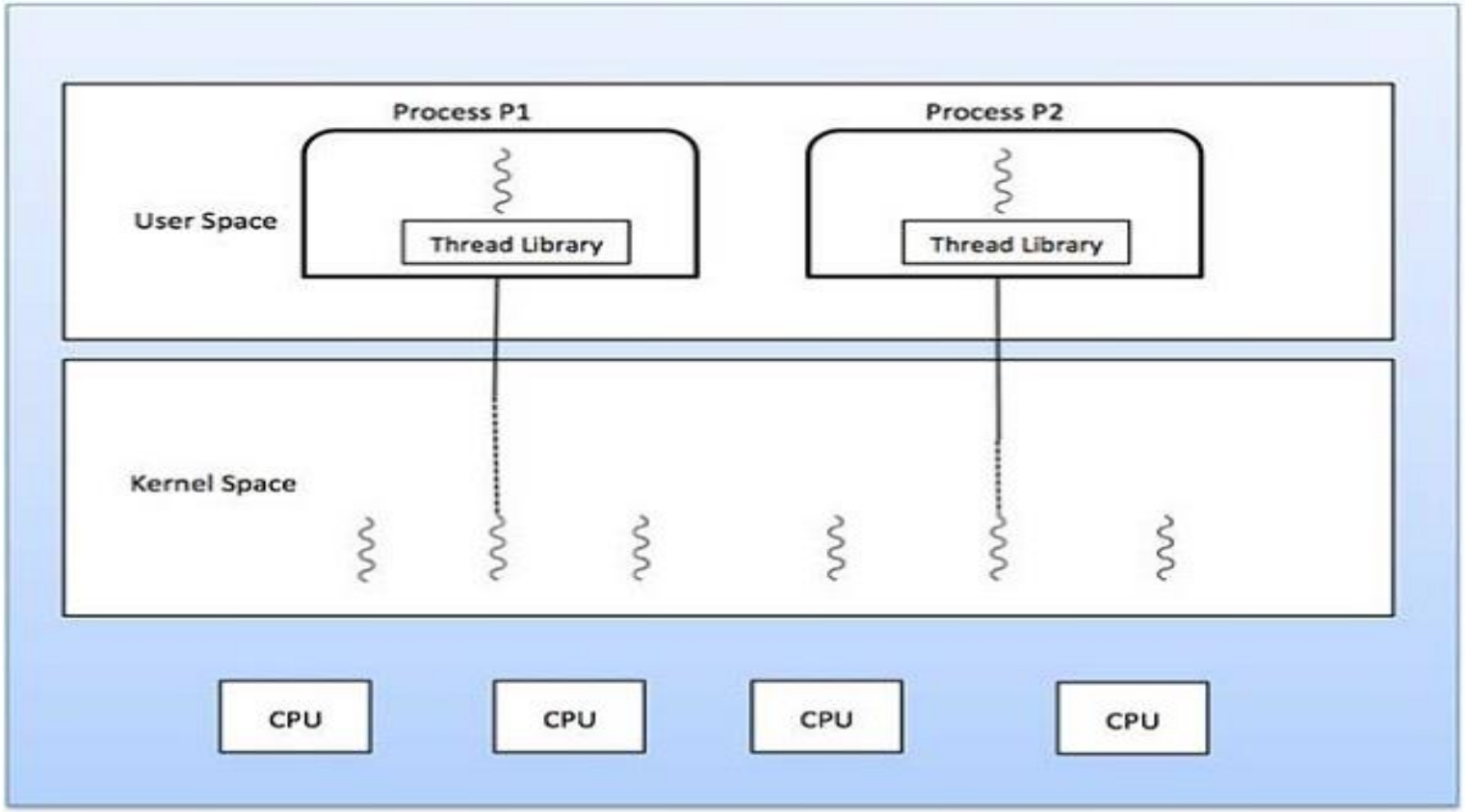
MULTITHREADING MODELS – MANY TO ONE MODEL

- Many to One models maps many user level threads to one kernel level thread.
  - Thread management is done in user space by the thread library.
  - When thread makes a blocking system call, the entire process will be blocked.
  - Only one thread can access the kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.
  - If the user-level thread libraries are implemented in the OS in such a way that the system does not support them, then the kernel threads use the many to one relationship modes.



MULTITHREADING MODELS – ONE TO ONE MODEL

- There is a one to one relationship of user level to the kernel level thread.
  - This model provides more concurrency than the many to one model.
  - It also allows another thread to run when a thread makes a blocking system call.
  - It supports multiple threads to execute in parallel on microprocessors.
  - The disadvantage of this model is that creating user thread requires the corresponding kernel thread.
  - OS/2, Windows NT and Windows 2000 use such multithreading model.



COMPARISON OF THREAD LEVELS

S/N	User-Level Threads	Kernel-Level Threads
1	Faster for creating and managing threads.	Slower for creating and managing threads
2	Implementation is by thread library at the user level.	OS supports creation of kernel level threads
3	They are generic and can run on any OS	They can run on specific OS.
4	Multi-threaded applications cannot take advantage of multiprocessing	Kernel routines themselves can be multithreaded.

CONCURRENCY

- **Concurrency** is the execution of the multiple instruction sequences at the same time.
  - It happens in the operating system when there are several process threads running in parallel.
  - The running process threads always communicate with each other through shared memory or message passing.
  - It results in sharing of resources result in problems like deadlocks and resources starvation.
  - It helps in techniques like coordinating execution of processes, memory allocation and execution scheduling for maximizing throughput.

- **Principles of Concurrency:** Both interleaved and overlapped processes can be viewed as examples of concurrent processes, they both present the same problems.
- The relative speed of execution cannot be predicted.
- It depends on the following:
  - The activities of other processes
  - The way operating system handles interrupts
  - The scheduling policies of the operating system

PROBLEMS IN CONCURRENCY

- **Sharing global resources**
  - Sharing of global resources safely is difficult.
  - If two processes both make use of a global variable and both perform read and write on that variable, then the order in which various read and write are executed is critical.
- **Optimal allocation of resources**
  - It is difficult for the operating system to manage the allocation of resources optimally.
- **Locating programming errors**
  - It is very difficult to locate a programming error because reports are usually not reproducible.
- **Locking the channel**
  - It may be inefficient for the operating system to simply lock the channel and prevents its use by other processes.

ADVANTAGES OF CONCURRENCY

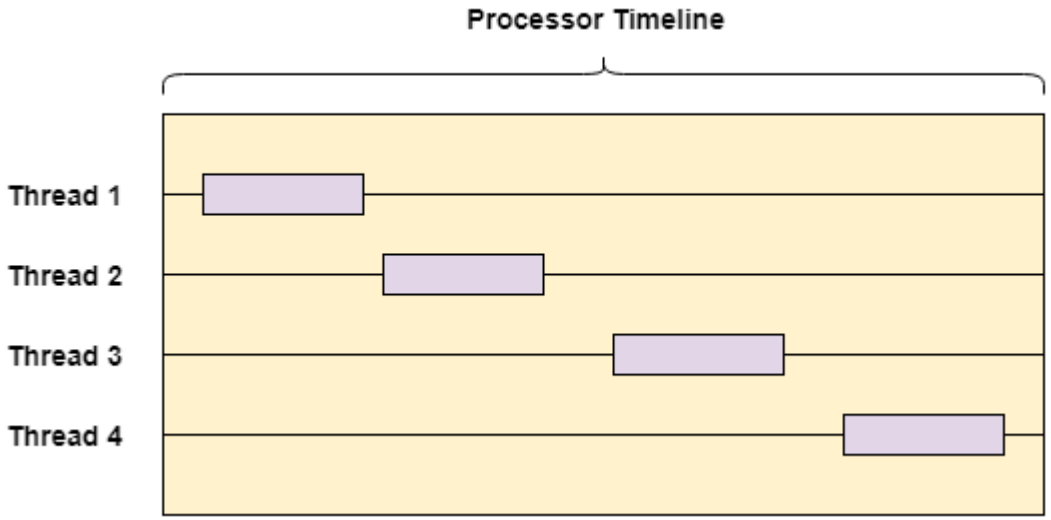
- **Running of multiple applications**
  - It enable to run multiple applications at the same time.
- **Better resource utilization**
  - It enables that the resources that are unused by one application can be used for other applications.
- **Better average response time**
  - Without concurrency, each application has to be run to completion before the next one can be run.
- **Better performance**
  - It enables the better performance by the operating system.
  - When one application uses only the processor and another application uses only the disk drive then the time to run both applications concurrently to completion will be shorter than the time to run each application consecutively.

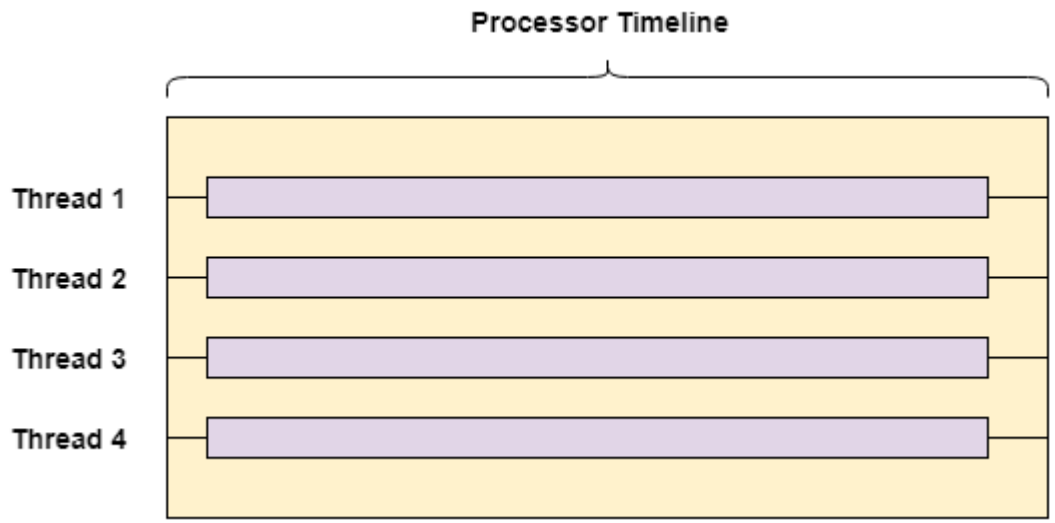
ISSUES OF CONCURRENCY

- **Non-atomic**
  - Operations that are non-atomic but interruptible by multiple processes can cause problems.
- **Race conditions**
  - A race condition occurs of the outcome depends on which of several processes gets to a point first.
- **Blocking**
  - Processes can block waiting for resources. A process could be blocked for long period of time waiting for input from a terminal. If the process is required to periodically update some data, this would be very undesirable.
- **Starvation**
  - It occurs when a process does not obtain service to progress.
- **Deadlock**
  - It occurs when two processes are blocked and hence neither can proceed to execute.

CONCURRENCY AND PARALLELISM

- **Concurrency**
  - Concurrency means that multiple processes or threads are making progress concurrently.
  - While only one thread is executed at a time by the CPU, these threads can be switched in and out as required.
  - This means that no thread is actually completed totally before another is scheduled. So all the threads are executing concurrently.
- **Parallelism**
  - Parallelism means that multiple processes or threads are making progress in parallel.
  - This means that the threads are executing at the same time.
  - This can happen if all the threads are scheduled on parallel processors.





CONCURRENT PROGRAMMING APPLICATIONS

- Multiprocessing can refer to one job using several processors
- This requires a programming language and a computer system that can support it, called **concurrent processing system**
- Most programming languages are serial – instructions are executed one at a time.
- In order to solve an arithmetic expression, every operation is performed in sequence.

CONCURRENT PROGRAMMING APPLICATIONS...

- For example, consider the expression
  - A = 3 * B * C + 4/(D + E) * * (F – G)

- The Step, Operation and Results are presented as follows.
  1. (F – G) – Store difference in *T1*
  2. (D + E) – Store sum in *T2*
  3. (T1) * * (T2) – Store power in *T1*
  4. 4/(T1) – Store quotient in *T2*
  5. 3 * B – Store product in *T1*
  6. (T1) * C – Store product in *T1*
  7. (T1) + (T2) – Store sum in A

CONCURRENT PROGRAMMING APPLICATIONS...

- The arithmetic expression can be processed differently if we use a language that allows concurrent processing.

- Define COBEGIN and COEND to indicate to the compiler which instructions can be processed concurrently:

COBEGIN

T1 = 3 * B

T2 = D + E

T3 = F - G

COEND

COBEGIN

T4 = T1 * C

T5 = T2 * * T3

COEND

A = T4 + 4/T5

STEP	PROCESS	OPERATION	RESULT
1	1	3 * B	Store difference in T1
	2	(D + E)	Store sum in T2
	3	(F – G)	Store difference in T3
2	1	(T1) * C	Store product in T4
	2	(T2) * * (T3)	Store power in T5
3	1	4/(T5)	Store quotient in T1
4	1	(T4) + (T1)	Store sum in A

CONCURRENT PROGRAMMING APPLICATIONS...

- Increased computational speed
  - Increased complexity of programming language
  - Increased complexity of hardware (machinery and communication using machines)
  - Programmer must explicitly state which instructions are to be executed in parallel, called explicit parallelism
    - Solution: Compiler can automatically detect instructions that can be performed in parallel, called implicit parallelism

CONCURRENT PROGRAMMING APPLICATIONS...

- In order to perform an array operation within a loop in three steps, for instructions might say:

```
for (j=1; j<=3; j++)
a(j) = b(j) + c(j)
```

- If we use three processors, the instructions can be performed in a single step:
  - Processor#1 performs: a(1) = b(1) + c(1)
  - Processor#2 performs: a(2) = b(2) + c(2)
  - Processor#1 performs: a(3) = b(3) + c(3)

CONCURRENT PROGRAMMING APPLICATIONS...

- To perform C = A * B where A and B represent two 3 by 3 matrices
  - $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, and $B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$
  - Several elements of a row in A are multiplied by the corresponding elements of the column in B

- Serially, the answer can be computed in 45 steps (5 x 9)

- However, with 3 processors the answer takes only 27 steps, by multiplying in parallel (3 x 9).