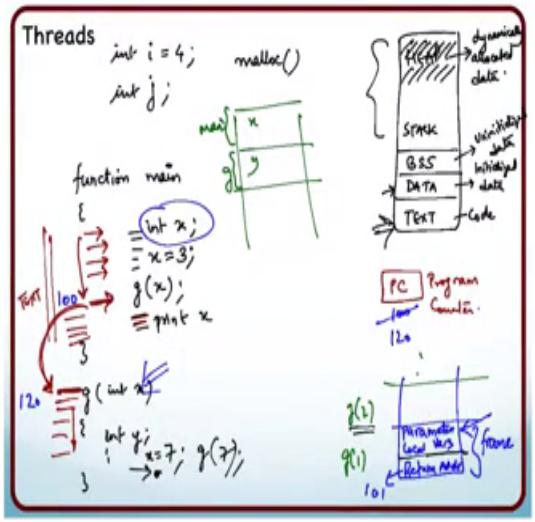
Introduction to Parallel Programming in OpenMp
Dr. Yogish Sabharwal
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

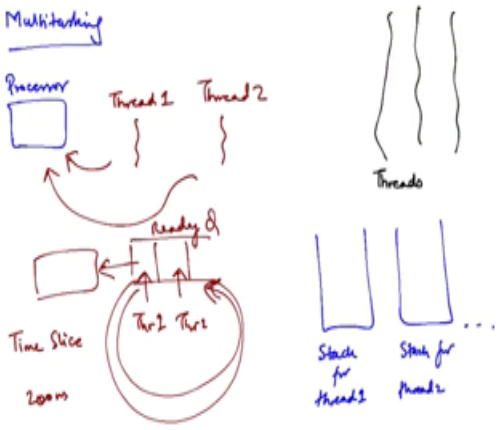
Lecture – 13
Program Memory with Multiple threads and Multi-tasking

(Refer Slide Time: 00:04)



Now what happened in a parallel program? So, I want to execute not one stream, but multiple streams I want to have multiple code flows. So, first of all one program counter does not do the job for me, if I have multiple execution flows.

(Refer Slide Time: 00:18)



So, I will call these execution flows as threads right if I have multiple threads, I want each thread to be able to execute it is own code right. So, let us come back here right. So, which of these all these segments do you think is required to be created separately for each thread, for each execution stream. So, different execution streams the different threads could be doing different things altogether. One could be computing a factorial the other could be computing a square root and so on, right.

First of all; the code segment it can be the same, the same code is loaded they might be pointing to different rotations their respective program counters might be pointing to different rotations right. The data segment that is global data shared by all the threads BSS that is also global data shared by other threads. Heap is dynamically allocated data, the allocation and release of that data all of that is controlled by calls to malloc and free.

So, they can also be shared by all the threads. So, what is the problem? The problem is the stack because each thread is executing it is own function right, it is its executing it is own code flow it is invoking it is own set of functions I cannot combine all of these into the same stack. I cannot put them all in the same stack what will happen if I jumble them up in the same stack. This frames will be interleaved right and then it does not make any sense, when I release one frame where do I go back where does the previous frame start from I have to go look it up.

I mean there is no sense in trying to maintain all these stack frames of different threads in the same stack right. So, each thread has it is own stack and so on. So, now, it can keep track of it is own frames, it can keep track of it is own code flow where is it supposed to come back and whatever function is executed by one thread it keeps track of the variables that I defined inside that function, within it is own stack right that is not shared with other threads does not make sense, right.

So, threads are essentially different execution streams which can execute independently; our threads scheduled by the operating system, how do they execute on the processor. So, most operating systems are multitasking; that means, that they can execute multiple tasks together. Let us say that there is program 1 program 2. So, there are multiple codes that are executing or these could be threads as well, suppose that there is only a single processor or let us just call it thread. So, there is thread 1 that is executing there is thread 2 that is executing and so on right. So, what it can do is, it can schedule thread 1 for some period of time with called a time slice, after the time slice it can throw outside 1 and schedule thread 2.

Student: (Refer Time: 03:30).

Not when it is completed. So, it might allocated for short periods of time, if you open up a browser window and you fire three different requests. So, all of them seem to be running in parallel right, but how are they running in parallel on a single processor architecture. Now on the processor they are being (Refer Time: 03:48) one at a time, there are several ways in which it figures out which thread should be scheduled. So, it maintains a ready Q, where it has let us say thread 1 thread 2. It keeps track of which threads are ready to be scheduled right and if thread 1 is ready to be scheduled it basically allocates the processor to thread 1 right whichever is at the head of the queue.

So, the thread gets to execute on the processor for some period of time right that is called a time slice that is determined by the operating system. And once that time slice is done it throws this thread out right, it puts it at the end of the queue and schedule the next thread there are even priorities and all that stuff. So, will not get into that the operating system determines which thread is suppose to be scheduled next, and one simplest way of maintaining it is just ready queue with round robin scheduling.

Another case where the thread may get scheduled out is it is not just purely based on time slice, if the thread does some blocking operations. So, what is the blocking operation? So, if it does some file system access if it tries to read a file. Now getting data from a file is very very expensive, it takes several milliseconds to actually get it. So, a processor does not set idle during that time. So, what does it do? It just picks up the tread and puts it somewhere else right. So, there are even queues maintained for all devices that who's trying to read this and so on.

So, will not get into all that stuff, but eventually it basically schedules it out of the CPU if it makes a call a blocking call. Similarly if it makes an http request to some website right, now it is performing an IO operation a network operation is being performed. So, the moment it performs at IO operation it schedules it out. So, what happens? The request is gone the http request has gone out to the server right it is going to take some time for it to come back, in the meanwhile it will pick up thread 2 and schedule it give the CPU resources to that.

Thread 2 is going to execute maybe it makes a request to another http server right. So, that request again and I will be performed again it will be scheduled out right. The third comes that makes energized http request it will be scheduled out. All operations which are very very expensive and the CPU does not have anything to do with it, because when you have made a server request now the server has to respond by god knows when it is going to come back. There is no point to waste your CPU resources waiting for that request to come back, similarly if you made a disk access there is no point waiting for this to send back the data it might take very long for it to come back.

So, the CPU it does not set idle it just schedules a next thread which is there allocates the CPU resources to the next step. And if a thread is just doing a lot of computation and not making any blocking calls to disk IO in that case, it will throw it out after it completes it is time slice. So, it has this notion of time slice and if the thread does not give up the CPU in that time, it will schedule, it out put it at the end of the queue all right. So, that is multitasking.