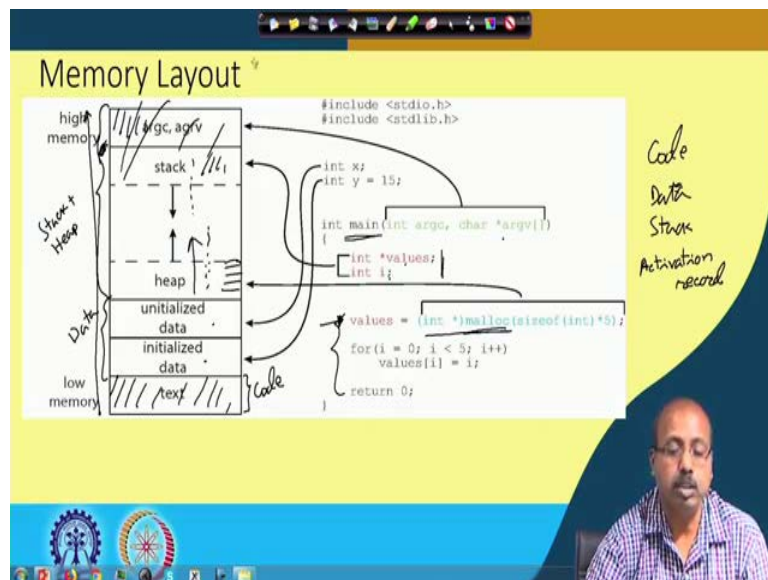


Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 19
Threads

So, in our last class we were discussing about the inter process communication primitives like message passing, then shared memory, then we have got this pipes and all. So, before proceeding further, so we would like to highlight like what you how the program will look like when it loaded into the main memory.

(Refer Slide Time: 00:47)



So, here we have got a sample C code where we see that there we have got some of the global variables like x and y and between x and Y, so x is uninitialized whereas, y is initialized to 15 and we have got a main program where some arguments are been passed argc and char star agrv, so argc and agrv, so these two parameters are passed.

And within the main program, so we have got this local variables values and i; out of these values is a integer is an integer pointer and i is an integer. And in the code we have got this where we first do malloc operations. So, it calls malloc and this in this malloc it returns size of integer into 5. So, it return space for 5 integers and then that point

returned pointer is stored into the values variable. And then in a for loop, so we are initializing the value of the we are initializing these values to i.

So, values i equal to i, so this is initializing the values variable to sum values and then return 0. Now, our objective is not to understand the meaning of this program rather we would like to see how this program will look like when it is loaded into the main memory. So, as we know that any program that is loaded, so, it can be considered to be consisting of three segments at least, the code segment, the data segment and the stack segment, so this we have discussed previously. Now, how do they look like?

So, if you look into this layout, so, if it is loaded, if the program is loaded from this address going towards the highest memory address like this, so, if it is the program is loaded in this region, then in the lower part of this memory locations the text of the program will be loaded. So, this program when it passes through a compiler, so it will generate some machine code and that machine code will be loaded will be copied into this text region. However, we need to create, so this is basically the portion corresponding to this part ok.

So, values equal to this, so what are the operations to be done when this program starts executing. As you know that when a C program starts executing the control is transferred to the first executable statement of the function main and this happens to be the first executable statement of main, so this is the control will be transferred here. Now, apart from that for the variables we need some space where the variables will be residing in the main memory and then we have got this global variables x and y, so they are kept in the data segment. So, this whole part is the data segment and whereas the previous portion that we have, so this parts; so this corresponds to the code segment.

So, we have got code segment and data segment and data segment is further divided into two parts. In certain place one part of it we keep uninitialized data and in the other part we keep initialized data. So, uninitialized data like x, it will go to uninitialized data because it is not initialized and the initialized data, so this will be y will be assigned some space here. So, what will be the size of this data segment that depends on the number of global variables you have and their types, size requirements etcetera and again the size of this initialized data and uninitialized data segment, so that will also depend on the variable that you have.

So, some system may put a limit on the size of these segments whereas, some segments some other operating systems they may not put any limit; so it varies from system to system. Then after that we have to see for the functions like within the main is a function and within that we have got this local variable values and i. So, they are assigned to the stack segment and as I said that data and this heap and stacks, so they are allocated on the same portion of memory, but they grow in the opposite direction. So, this entire chunk of memory space, so this is allocated to stack plus heap, they are allocated to stack plus heap.

So, stacks starts growing from the highest address, heap starts growing from the lowest address, so they grow in the opposite direction. So, that if your program makes use of more of stack space then it will be able to do that. On the other hand, if a program does less of function calls, but more of dynamic allocation, so, they will be done they will be allocated at the heap space. So, they are put on the opposite ends of the same memory block and they grow like that, the stack and heap. So, in this case you see that this main program it has got this local variables values and i.

So, they are assigned space into this stack segment, whereas, this when this particular function is called this malloc function is called, so, what the system does? It tries to assign some dynamic space to the program ok. So, here I am asking for space which is equal to 5 integers. So, if I have got space available in this stack plus heap segment what the system will do? It will allocate 5 memory locations. So, if I say that these are those 5 memory locations. So, this 5 memory locations will be allocated and that pointer will be given to this values variable and then, so that is allocated in the heap. So, this stack and heap they grow in the opposite direction. So, they are so it is located here.

Now, if a program is doing is a recursive program, then what will happen? So, it will have more lot of function called recursive calls, so this stack segment will grow significantly. On the other hand if a program has got more of these dynamics storage utilization like linked list type of program, so, there for them this heap segment will grow significantly.

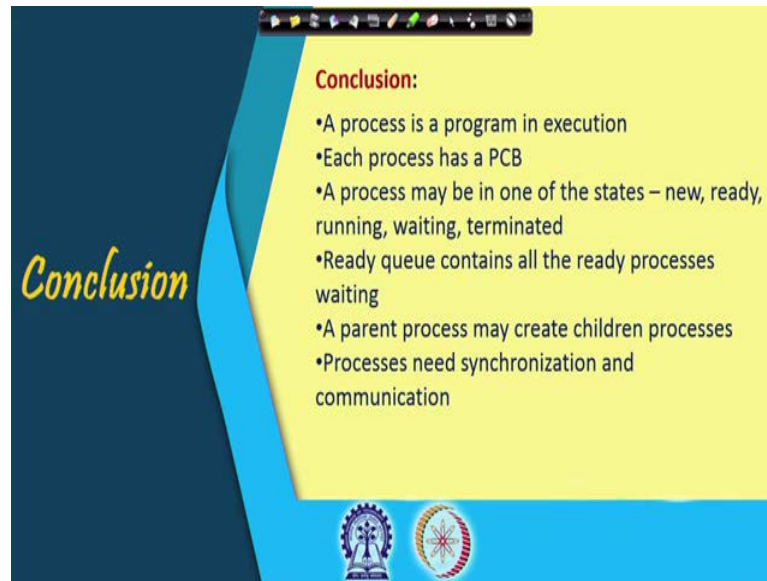
So, whatever it is ultimately it now we have got this variables implemented and then this argc agrv so, they are actually for some parameters for the whole program and they are allocated some space which is over and above the original space because it is coming

from some other programs, so whichever program is calling this so they will be giving this argc agrv. Sometimes, so this also this is also local variable of these parameters that is passed for the main routine. So, you can say that actually this is also a part of this stack for this for the function main.

So, I should draw start this stack plus this thing from here. I have to include that argc and agrv. So, they are parameters; so they are also created onto the stack. So, if you are familiar with compiler design courses, so, you will know that there is something called activation record. So, when a procedurals called, so it creates the system creates some activation record and this activation record, so this holds the information like the parameters that are passed, the local variables that are created and all that and whenever if call is made one such activation record is pushed into that stack. So, you can say that this argc agrv values and i. So, they will constitute one activation record for the function main.

So, that will be put into the stack and the system will and the execution will start like that and when this main is over this whole thing will be popped out from this stack, so it will go out and then the stack content will become similar as the in the previous one from which point it was called. So, I would suggest that you look into some compiler design books for details about this activation record and all and you will get a better idea like how these variables are assigned. So, an operating system designer we have to provide a mechanism by which this compiler designer can create this activation record put them onto stack etcetera. So, that is the duty of the OS designer.

(Refer Slide Time: 09:27)



Conclusion:

- A process is a program in execution
- Each process has a PCB
- A process may be in one of the states – new, ready, running, waiting, terminated
- Ready queue contains all the ready processes waiting
- A parent process may create children processes
- Processes need synchronization and communication

So, with this we come to a conclusion of this discussion on process. A process is a program under execution, so a program is I should say that it is a static item that is residing in the disc and then when it goes to the when it goes into execution, so it becomes a process. Each process has got a process control block that holds all the information regarding the process starting with identifier or identification of the process, its ownership and all up to the context that is required like the CPU register values, then this program counter, then this memory limits, then this scheduling related parameters like your timing, the CPU usage, disc usage, memory usage, so like that.

Now, process we have seen that process goes through various states in its execution when it is when the user has just given the comment executive program. So, that is creates a new process, so the state of the process is new. After the program has been loaded into main memory or the process has been loaded into main memory. So, it is ready; so it is ready is in the sense that if it can if it gets CPU, now it can starts executing.

So, after sometime the scheduler will pick up the process for execution, so it goes to the running state and of during running so, there can be two situation after sometime may be the process wants to need some IO operation or the process maybe may have to wait for some event like a process might have created a child process and this process now wants to wait for the child to be over. So, that way the process may be put into a blocked state

waiting for some input output operation or some event or maybe that is the process is given CPU for a small time quantum. So, after the time quantum expires the process is taken back from the running state and it is put it put back into the ready state, so another process gets chance execution.

So, this way process from the running state it can go to a waiting state or it can go to a blocked state and also a process while executing in the CPU it may terminate because its execution is over. So, it goes to a state which is called terminated and we remember this because the parent process you created this one may need to know what was the outcome of the previous process that it created. So, it may be interested about the successful termination of the process and all. So, this exit code of the process becomes important. So, we have seen that the ready queue it contains all ready processes waiting for getting the CPU.

A parent process may create several children processes, so that way there is a parent child relationship between the processes. And of course, processes need synchronization and communication this is very important because a process a number of processes they may constitute a whole system and as a result we will be requiring them to communicate between themselves and sometimes a different parts of computation is done by different processes. So, we need to have proper synchronization between them. So, in our successive classes, so we will be looking into this synchronization; so then this scheduling, so etcetera in more detail. So, with that we end this portion.

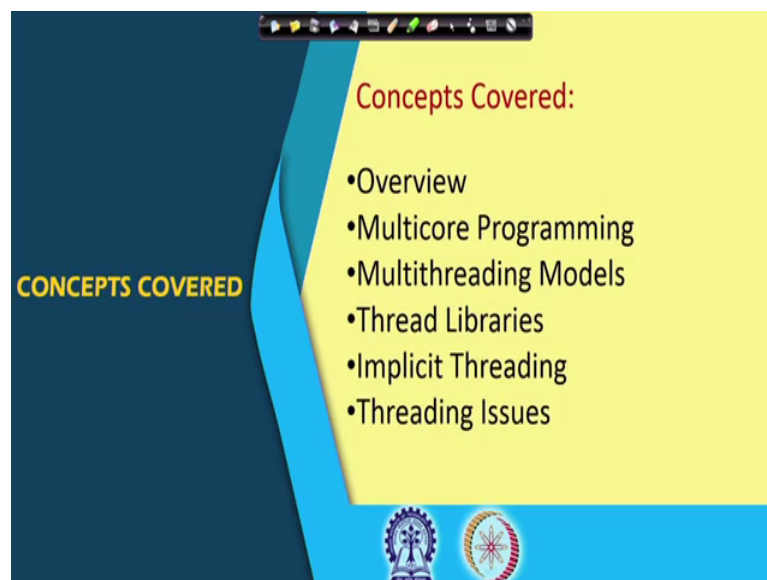
And so, next we will be going to another topic where we will be discussing about the something called threads ok. So, this threads is basically a thread is. So, next we will be going to this discussion on threads, so basically it is a variant of process, we will see that it is not as much detailed as process. So, what happens is that many a times the computation that we are doing across different processes they are quiet similar.

For example, like if we are doing say web browsing. So, web browsing ultimately it is contacting some server and maybe there are 10 different people who are accessing from different places and all of them are trying to access a server. So, as far as the server is concerned, so it is getting request from 10 different clients and it is trying to process them.

So, one way for handling this type of situation is that you create you solve this client requests one by one, but that way somebody may feel that ok I am not getting enough care or enough attention. So, response to some of the client requests maybe slow, so you want to make it fast and for making it fast what you have to do is that we have to create multiple processes. So, each the client request come, so we create a process and that process takes care of that particular request.

Now, the difficulty with this type of situation is that whenever we have got a process means we have got this code, data and stack segment, but that often becomes too heavy and we will see that in different application, so that much distinction between these processes is not necessary.

(Refer Slide Time: 14:47)



So, in this part of the lectures, so we would be looking into a new concept called threads which will make this execution much more simple and that will that will help us in developing programs particularly for multi core environment, where we where would there several quotes are there and they can take up that different jobs. So, overall what to that topics that we are going to cover is first we will have an overview of what is threads etcetera. We will be talking about multi core programming then we will talk about multithreading models. So, multi core I hope you are now familiar like most of this CPU chips that you get, so they have got multiple cores in them.

Now, it is not sufficient that processor chip it has got say 8 different cores, what is required is that as an OS designer I should be able to put 8 different operations on to those 8 different codes, if I am not able to do that. So, if I if my OS cannot create 8 different parallel tasks and put the put them on to these 8 core, then ultimately it becomes a sequential one. So, performance wise we do not see any difference. So, it is not only the availability of the hardware, so it is the operating systems responsibility to exploit that hardware in a proper fashion. So, this threading concept will help us in doing that exploitation.

So, we will be talking about multithreading models, so there can be different types of models for this multiple threads like how they are related to each other; so we will be looking into that. Now, for making this thread based programming easy, so there are several thread libraries that are available. So, we will be looking into some of them though not in much detail, but will see we will have a glimpse of like how they are the thread libraries will look like and what are the essential features that we get from a thread library.

So, that if you are exploring a particular library you can look into those features and start using them. There are some implicit trading like automatically the threading occurs and then we have got some threading issues, so that that we will discuss one after the other.

(Refer Slide Time: 16:51)

Motivation

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
 - Update display
 - Fetch data
 - Spell checking
 - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

The slide features two hand-drawn diagrams of servers. The top diagram is labeled 'web server' and has four lines extending downwards, each ending in a small circle. The bottom diagram is labeled 'database server' and has three lines extending downwards, each ending in a small circle. The slide is set against a yellow background with a blue gradient at the bottom. At the bottom of the slide, there is a Windows taskbar with various icons and a small inset video of a man in a checkered shirt speaking.

So, why should we go for this threading? Most modern applications are multithreaded. So, a typical exam that I have that I said previously is like this, so I have got a web server. So, this is a web server and to this web server, so we have got several client requests that are coming. So, this client; so that are coming, so they are similar in the sense that they are trying to access some information from this web server and get something done ok. So, in that they are more or less similar, but maybe this fellow this client request is they are trying to access some part of information, where as this fellow is trying to access some other part of information, but it is accessing all of them they are using some services which are provided by the web server.

So, if you look into the code that you have here and the code that you have here; and the code that you have here. So, there is there are lots of similarities between them because ultimately what they are doing is requesting from some service from the web server. Or if you have got a database server, then in the case of database server the type of operations that we do is to access some of the records that we have in the database and possibly modified them.

Now, all this queries that have coming, so all this queries they are again of similar structure in the sense that what they do is that they are trying to access some records in the database and then trying to do some modification to the database and all. So, that way this structure of all these are quite similar. Now; so that is that is why it is said that most modern applications are multi threaded. So, we have got; so we have got this another example here after sometime we will see and threads they will run with an application. So, application has got multiple thread and this thread will done in parallel. So, one (Refer Time: 18:55) a typical example maybe like this that sorry; so one typical example maybe like this.

So, we have got multiple task with applications can be implemented by separate threads like we may have some application that that perform this thing. So, it updates display, fetches data may be from file maybe from internet wherever it is, then it perform some spell checking and answer a network request, may be in one application we are doing this 4 services.

Now, these 4 services you see they can go in parallel, so if you write a single piece of code which is doing all these things, the code becomes very clumsy. So, instead of that

we design different different codes for different different application, so that is for different different tasks. So, that is one way, but all of them are part of same application ok, but their tasks are different.

(Refer Slide Time: 19:57)

Motivation

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
 - Update display → P₁
 - Fetch data → P₁
 - Spell checking → P₂
 - Answer a network request → P₂
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

So, as a result what will happen is that, if you look into the individual operation that this fellow is doing. So, it is different from this, but many, so major part of it major part of them may be similar also because they are accessing the same data elements and they are modifying the same data elements, so like that. So, that way; so whatever the data this fellow has faced, so this spell checking module, so it is working on the data sets.

So, if you make it separate, so if you create two different processes suppose this fetch data is done by P 1 and spell check is done by P 2. Now, the difficulty that we have seen previously between P 1 and P 2 even if they are even if there is a parent child relationship between P 1 and P 2 say they do not share the data segment between them. So, whatever has been faced by P 1, so for P 2 that is not visible?

So, for P 2 for making it visible to P 2 I have to put them on the shared memory and from there only this P 1 and P 2 can see them and the shared memory is limited in size as you know that any operating system there will be limited amount of shared memory. So, design in such a system becomes quiet complex. So, if you can do something, so that whatever the data segment P 1 has; so P 2 has it has it common with P 1 and whatever

modification P 1 is doing, so it is visible to P 2 and whatever modification P 1 is doing is visible to P 1, so like that if it can be designed.

So, this is essentially the idea behind this threading. So, we will have separate threads for doing different for separate operations, but there will be lots of sharing between them. So, this is the this actually summarizes the statement that I was trying to make. So, this process creation is heavyweight because if you are why do you say so, like if you want to create a process, so; that means, you have to create the code data and stack segment for that process; code data and stack and heap. So, they are to be created for this process.

And then possibly you have to tell like which code should execute after. So, after so you have to use some fork system call after that there should be some exact system call for doing this operation and this is heavy because you have to update the process table, you have to create the PCB and all those things are to be done.

So, that way it becomes a heavy thing and so difficulty with being heavy is that if you want to switch between the processes, it becomes time consuming. So, over all throughput of the system will be low. On the other hand this thread will see that, they are defined in such a fashion that you do not have much work to do to create a new thread ok, so this is called lightweight; so thread creation is lightweight ok. So we will see how is it lightweight. Now, it can simplify code and thus increase the efficiency, kernels are generally multithreaded.

(Refer Slide Time: 23:03)

Motivation

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
 - Update display
 - Fetch data
 - Spell checking
 - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

Handwritten notes on the slide:

- $x = x + 2$
`print("...")`
- `P2 fork()`
- `P1 fork()`
- `print()`
- Diagram showing `User Space` and `Kernel Space` with an arrow pointing from User Space to Kernel Space.
- `Single threaded kernel`
- `Process table` (represented by a grid)
- `File table` (represented by a circle)

So, this statement I would like to emphasize like so as you know that in any operating system. So, we can say that it can be divided into two origin; one is called the user space; one is called the user space and other is called the kernel space. Now, in the kernel space we have got the important data structures and routines that that are useful by the operating system.

So, whenever you need some system service, a program it has to make a system call and by this system call the program goes from user space to kernel space and starts executing in the kernel mode. For example, when you are doing the computation like $x = y + z$ etcetera, the simple arithmetic logic of computation, so this is fine but as soon as you are telling that I want to do a print f, I want to do a print f something, then you need to access the IO device and as a result that goes into the kernel space of execution or whenever you are trying to create a new process that it goes into the kernel mode of execution.

Now, suppose in this kernel I have got only one thread ok, so only one thread is there. So, whichever process wants to get a system service it will be using this particular thread for doing the operation.

So, as a result if there are multiple processes or multiple users in the system. So, whenever they make a system call, so they, so only one of them will be allowed and others will be blocked. So, this is a very serious problem because it was so happened that while executing the system call corresponding to one particular users. So, there was an error maybe because of OS or maybe because of this user the way this call is made it etcetera, the call may fail and many things can happen. So, we just take this is this is just take into consideration the situation that we have got a single thread here in the kernel space and all the users user processes that wants to make system call will be utilizing this particular thread.

So, that type of design is known as single threaded kernel; so this type of design is known as single threaded kernel. Now, initial operating systems that were designed, so they were of this nature single threaded kernel, the advantage that you get is that. So, since there is only one thread inside the kernel, so many problems of this is synchronization and inconsistency they get resolved.

So, but, so that, but the difficulty that we face with the single threaded kernel is that, we cannot get multiple jobs going on simultaneously. So, it may so happened that one user is trying to modify say one user is making a fork system call by that user wants to create a new process. Another program, so it is trying to do a print f by accessing the printer data, the printer source.

So, that way these two as such there is no harm in making them to proceed simultaneously, but since only one thread is available only one of the processes will be allowed will be allowed to proceed. So, other one will be blocked till this fellow as finished. So, that way a performance the system throughput will be low, but the point is that if inside, if we allow multiple threads then the difficulty that we get if suppose I have got a process P 1 which is executing a fork, I have got a process P 2 which is also executing a fork.

Now, whenever this is done, so if so if I allow both P 1 and P 2 to come into kernel mode and start executing, then both of them as a as a part of executing this for. So, they will try to access something called the process table ok. So, they will try to access the process table and this process table updation it may so happened that this updation becomes inconsistent maybe something is written by process 1 and that is over written by process 2. This will be more clear when we go to this concurrency control chapter, but whenever you have got any shared resource and multiple processes are trying to modify them that creates a difficulty ok.

So, we allow only we should allow only one process at a time, but the single threaded kernel. So, it has got the advantage because only one process will be allowed or only one thread will be allowed inside the kernel. So, the synchronization problems are solved, but this there was a throughput a performance will be poor, the multithreaded kernel multiple threads can be there inside the kernel mode. So, your throughput or performance will be better, but the this design will become more complex because you have to have some protection. So, somehow if there are say many such shared resources. So, I should somehow ensure that this process table is not accessed by more than one process simultaneously or more than one third simultaneously.

Similarly, if there is another table which is say the file table that remembers all the open files that we have in the system. So, if two processes they are trying to open files, so they

should not be allowed simultaneously to modify this file table, so that you should have protection there. So, you should have protection around each of these individual data structures or individual table. So, that create that makes it difficult and the, but with the multithreaded kernel designs. So, this is taken care of and most of the operating systems that we have today, so they are having multithreaded kernel.

So, we will look into this multithreaded kernel situation in more detail and how the mapping is done between user level and kernel level in case of threading. So, in the successive classes we will see.