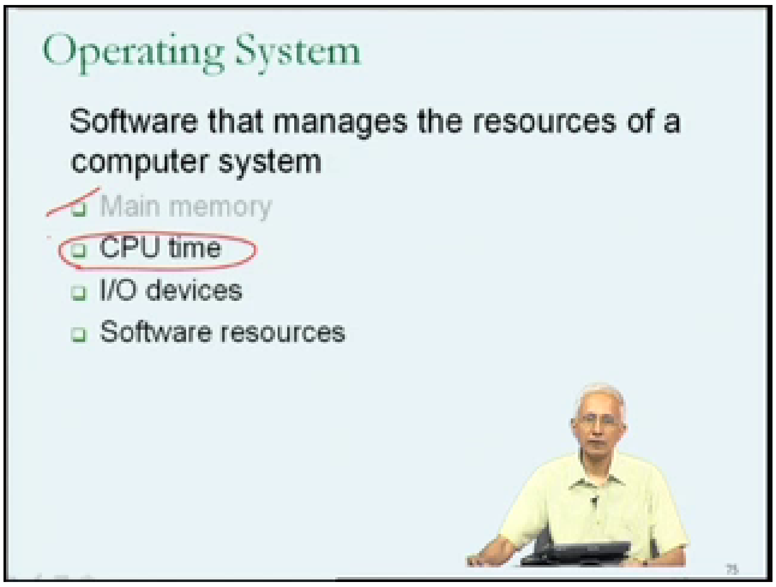
High Performance Computing
Prof. Matthew Jacob
Department of Computer Science and Automation
Indian Institute of Science, Bangalore

Lecture No. # 16

Welcome to lecture number 16, of the course on, High Performance Computing. For the past five lectures or so, we have been looking at the operating system, which is the key piece of software running on a piece of computer hardware, which heavily interacts with your program, in execution, making it possible for the program actually utilize the hardware.

(Refer Slide Time: 00:40)



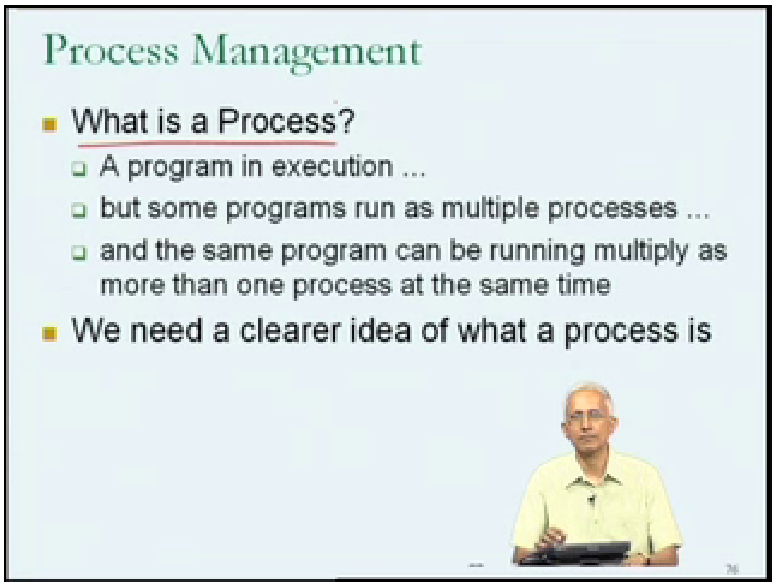
Operating System

Software that manages the resources of a computer system

- Main memory
- CPU time
- I/O devices
- Software resources

So, in the study of the operating system, you will recall from the end of the previous lecture that there are at least four components that are important for us to understand. We have already looked at main memory, in terms of the most common way that operating systems manage main memory today, in terms of paged virtual memory. And we are about to embark on discussion on how the operating system manages the CPU and the other critical resource, the most critical resource in the computer is how a single processor is shared among many programs in execution.

(Refer Slide Time: 01:24)



Process Management

- What is a Process?
  - A program in execution …
  - but some programs run as multiple processes …
  - and the same program can be running multiply as more than one process at the same time
- We need a clearer idea of what a process is

Now, in order to get a good feel for how the operating system does this, it turns out that we have to go back to our discussion, about the concept of the process, and in fact, rather than talking of this area of operating systems as CPU management or CPU time management, which was our perspective, from the hardware side, we could just as well think of it as, Process Management, if one thinks about the start from the perspective of the software side, the software organization of the system.

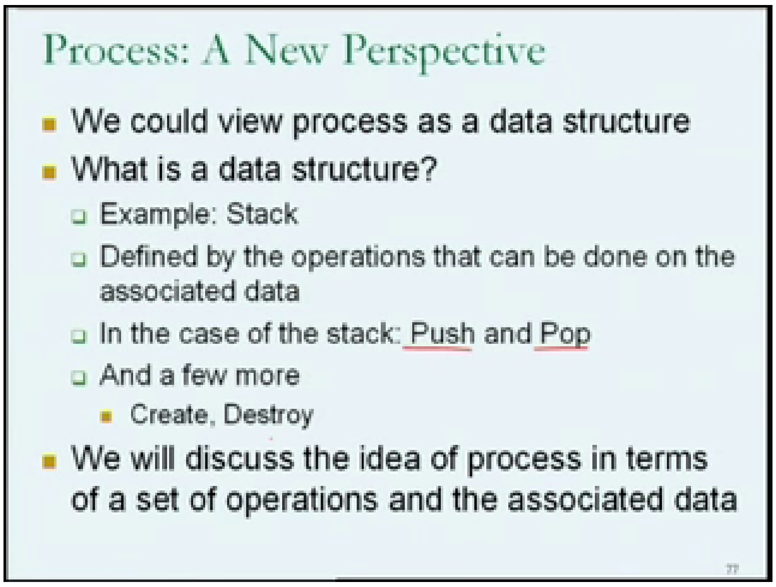
So, hence forth, I will actually refer to this as, Process Management, rather than CPU time management. Now, earlier when I had introduced the concept of process, I had mentioned that we will think of a process, as being a program in execution. So, when you asked for a program to be executed on a computer system, it exists as a process on the computer system. But, soon after having made that (( )) definition without explanation of the term process, I had indicated that would need a better understanding of what a process is, to understand the mechanics of what happens inside the operating system. So, we will actually spend a little bit of time addressing this question, what is a Process? Try to understand what the attributes of a process are, and from this we should be able to get a better picture of the process, from the perspective of the operating system, in other words, the run time existence of the process.

Now, we have started with the explanation that a process is a program in execution. And we understood that it is true that when a program starts executing, it will exist as one process. But, subsequently could through the fork system call, continue its execution is multiple processes. And so we were not satisfied with the explanation of process is a program in execution, and we extended it because there could be some programs, which run as multiple processes and therefore, we understand that a process is something different from the execution of a program. Because, the execution of a program involves multiple processes, further if we think about this little bit more, you will realize that they could be a particular program, which is running multiple times on the same processor and the best way to look or think about this is the example of the shell, which we had used earlier.

You will recall that when you login to Linux or Unix computer system, your interaction with the computer system happens through a shell and you prompted by the shell to type in a command. So, you have the shell prompt on the screen. Let us suppose, I am using the bash shell in Linux. Then, when I get the prompt on the screen, it is being printed by the bash program, running as a bash process a single process.

Now, one thing that I could type in response to this command is bash, in other words, I could cause another shell, another instance of the program bash, to be running as a sub processor or child process of the shell which was or which just now printed the prompt on the screen. So, if this is the case, and after typing bash and entering pressing enter, I will have two processes running bash on the computer system. This is two programs in execution, running as two processes. So, in order to avoid any confusion about process and program, hence forth, we need and we will discuss the concept of process and differentiate, distinguish from the concept of program and then look into the operating system in details.

(Refer Slide Time: 04:45)



Process: A New Perspective

- We could view process as a data structure
- What is a data structure?
  - Example: Stack
  - Defined by the operations that can be done on the associated data
  - In the case of the stack: Push and Pop
  - And a few more
    - Create, Destroy
- We will discuss the idea of process in terms of a set of operations and the associated data

So, we need a perspective. We do not have good perspective at this point. We need, let us say a new perspective on what the process is, and I am going to start by trying to view a process, as a data structure. Now, this may come as a surprise. But, if you think about it little bit, we have already seen data structures. In fact, we saw the example of a stack and we know, therefore, what a data structure is and we could try to apply our understanding about the data structure is to everything that we know about a process, and through that exercise may refine our understanding of a process and improve our understanding of what data structure is.

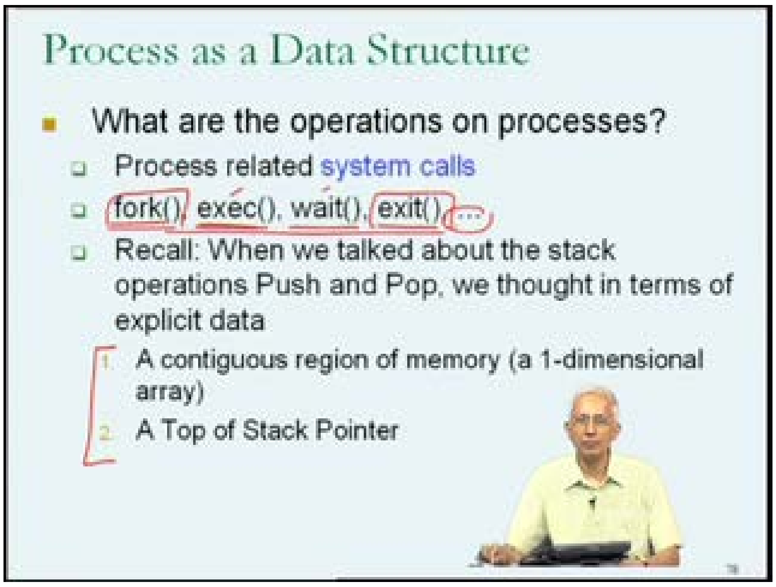
Let me just remind you what we had talked about the stack. Now, I had mentioned that the stack is an example of a data structure, and the way that we describe the stack data structure, was to start by describing the functionality of the stack. So, we talked about the operations that can be done on the data associated with the stack. So, in talking about the stack, we came up with two I described two operations. Remember, the picture you should have in mind with regard with the stack data structure is functionally, in terms of its operations, something like a stack of books on a table, so you could put a new book onto the top of the stack or you could remove the book, which is currently on the top of the stack and these two operations were what we know by the terms, Push and Pop. You push a new book or a new element on to a stack, and you pop the element from the top of the stack and these were the only two operations we talked about in connection with the stack.

But, at this point, we will add a few more, just to make sure our understanding of the stack is complete. Now, the additional operations, which I am going to add, relate not new functionalities of the stack, but of the fact that at some point, during the execution of a program when it wants to use a new stack, it will actually have to create a stack. Henceforth, when we think of a data structure, we will possibly have to add operations for the creation and the deletion of the data structure, and I will do that for the stack right now. So, therefore, now on, we will not talk about just two functional operations of pushing and popping data to or from a stack, but rather also the operation of creating a new stack and of destroying a stack.

And hence we could actually talk about four operations on the stack. Now, in talking about the stack, the operations push and pop are quite easy to understand. In the sense that we knew that when you push a value on to the stack, the stages of the stack changes and the new values on top of the stack, the stack pointer changes, to point at this new value and so on and similarly, for pop.

Thinking about the functionality of create and destroy similarly, one could think about create as being an operation, which causes space to be allocated for the stack, in memory, and initializes the stack pointer, in some way. Whereas, destroy would reclaim the space, which had been associated with the stack and also cause the end of the use of the stack pointer. Hence, the stack no longer exists after this. So, all four operations are functionally quite clear to us at this point, from our use of the stack in few contexts. Now, we need to do the same thing for the idea of the process. We have to try to understand the idea of the process, in terms of a set of operations, on the data that is associated with the process.

(Refer Slide Time: 08:02)



Process as a Data Structure

- What are the operations on processes?
  - Process related system calls
  - fork(), exec(), wait(), exit(), ...
  - Recall: When we talked about the stack operations Push and Pop, we thought in terms of explicit data
    1. A contiguous region of memory (a 1-dimensional array)
    2. A Top of Stack Pointer

So, this raises the question of if I want to view the process as a data structure, have to understand what the operations on the process are. What are the operations on processes? After this point in time, we have seen a few operations on processes, which came in the guise of system calls. They were the process related system calls and I had mentioned I think, three or four such system calls related to processes. For example, there was a system called using which you can create a new process. There is the system called using which you can change the memory image of the process. A system called using, which you could cause a parent process to wait for its child process and a system call using, which you could cause a graceful termination of a process, in some sense.

So, clearly the fork is the create, in the case of the stack, we have the create operation, the fork system call serves a similar role here, where as the exit system call, which causes the graceful termination of the process, plays a role of destroy of the stack, destroy operation. And exec and wait cause some change to the state of the process in ways that are defined by the system call, as we saw exec changes the memory image, wait causes the process, the parent process to wait for some behavior from its child or to sleep, until its child had satisfied certain requirements.

So, very clearly, we can think about operations on processes. This is not the problem as far as giving the process as a data structure, and I put some dots here (Refer Slide Time:

09:30), because as I have mentioned in talking about system calls, they are additional operations on processes as and when the need arises, we will learn about them.

So, this is much more sophisticated data structure than the stack where there were only four operations, here we have a list of several operations. Now, in talking about the stack you will remember, we talked about the functional operations, but understood them in terms of how they manipulated the data associated with the stack and we need to do something similar to that for our understanding of the operations on the process.

In other words, we need to understand how fork manipulates the data associated with the process, how exact manipulates the data associate with the process and so on. So, in talking about the data structure, in general, we need to understand what the operations are and also what the associated data is and how the operations relate to the associated data.

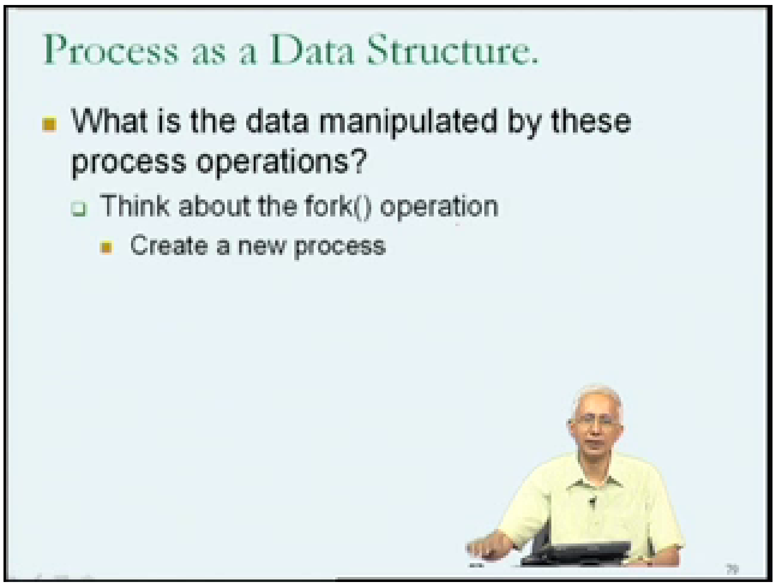
In the case of the stack, you will remember our picture of the data associated with the stack rose, when we talked about trying to implement the stack. Until, the time came to implement the stack, it was enough for me to talk about the example of a stack of books in order of you to visualize the stack. Conceptually, that was adequate. Subsequently, you could think about a stack of integers, as a stack of integers on the table in front of you, and that was adequate for the purpose of understanding the functionality of the stack.

But, when the time comes to implement the stack, we have to think in terms of variables or regions in memory, which will be associated with the stack and that is where when we talked about the data associated with the stack data structure, we talked about a region of memory, which was in a sense one-dimensional array and we talked about whether the stack would grow to lower and lower memory addresses or to higher and higher addresses, within that region.

So, we were actually talking about the data in terms of an implementation, not conceptual understanding of the data from the perspective of the concept of the stack of books on a table. What else was there associated with the stack data structure? Was any other a piece of data? The answer is yes, there was one other very important piece of data associated with the stack data structure, in addition to the region of memory, which was going to hold the different pieces of data, which were pushed into the stack. The second, an important piece of data associated with the stack is the top of stack pointer.

Without the top of stack pointer, the region of memory, would be useless, in terms of implementing a stack, because there will be no way to figure out, which was the current top of stack element for the push or the pop operations to operate from. So, we had a very explicit notion of what the data associated with the stack was from the perspective of implementing the stack and we need something similar from the perspective of viewing the process as a data structure. In other words, explicitly what are the pieces of data associated with the process data manipulated by the process operations, and that will complete our picture, our perspective on viewing the process as a data structure.

(Refer Slide Time: 12:32)

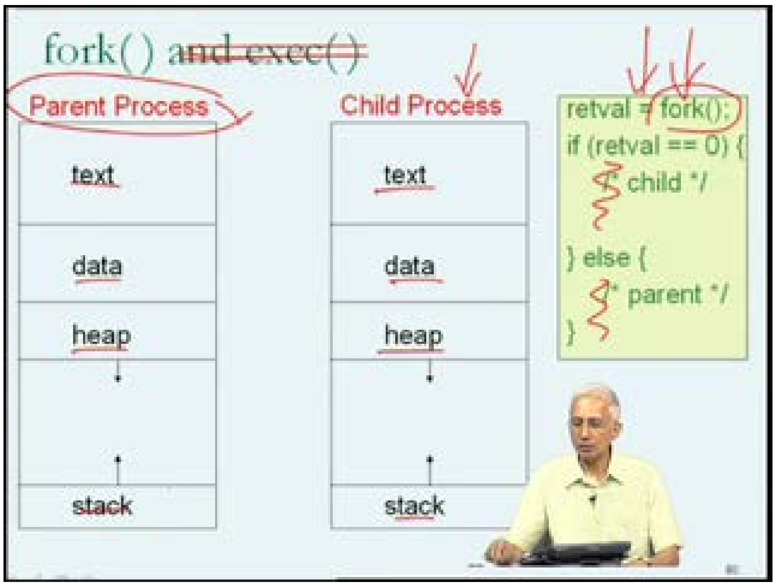


Process as a Data Structure.

- What is the data manipulated by these process operations?
  - Think about the fork() operation
    - Create a new process

Now, this then is the question we have to deal with. We understand the process operations; we understand that they must be manipulating data. So, what is this specifically the data manipulated by the process operations? Let us think about fork to start with. Now, we understood that when you do the fork operation it creates a new process, and that is the conceptual idea at a level of talking about a stack of books on a table. At this point, we have to try to think about explicitly what is the data that has to be associated with the fork operation.

(Refer Slide Time: 13:06)

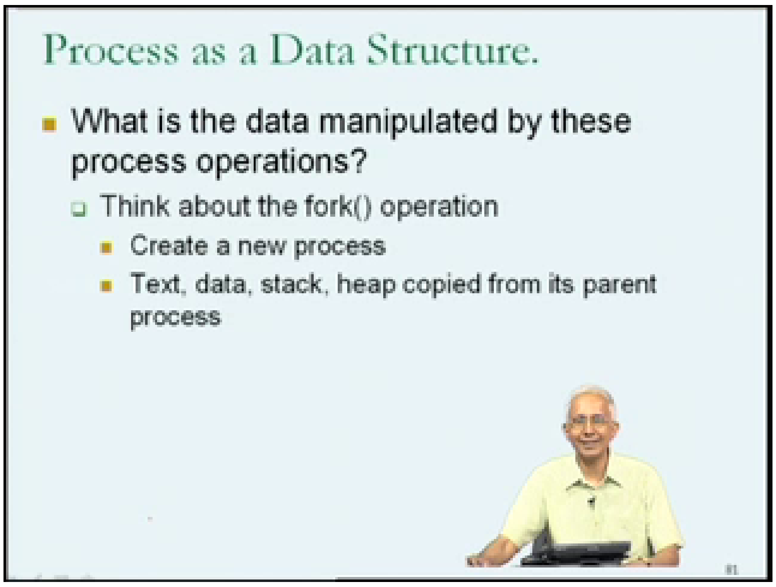


```
retval = fork();
if (retval == 0) {
  /* child */
} else {
  /* parent */
}
```

So, we will go back to our picture of what happens when a fork is executed. This is the same slide that we had. It also dealt with exec that would not be necessary for this discussion. So, I will strike it out.

So, you will recall that if there is a process, which is executing the code shown on the right side. So, the child process is the process that is going to come into existence. So, initially, there is one process, which is the parent process. It is executing the program that you see on the right, and over here it executes the fork system call, as a result of which a new process comes into existence. The new process is the child process, which executes this piece of code, when it next gets a chance to execute. Whereas, the parent process executes this piece of code, both are running the same program. So, the child process was initialized with its own, with the text, data, stack and heap, just like the parent process had.

(Refer Slide Time: 14:09)

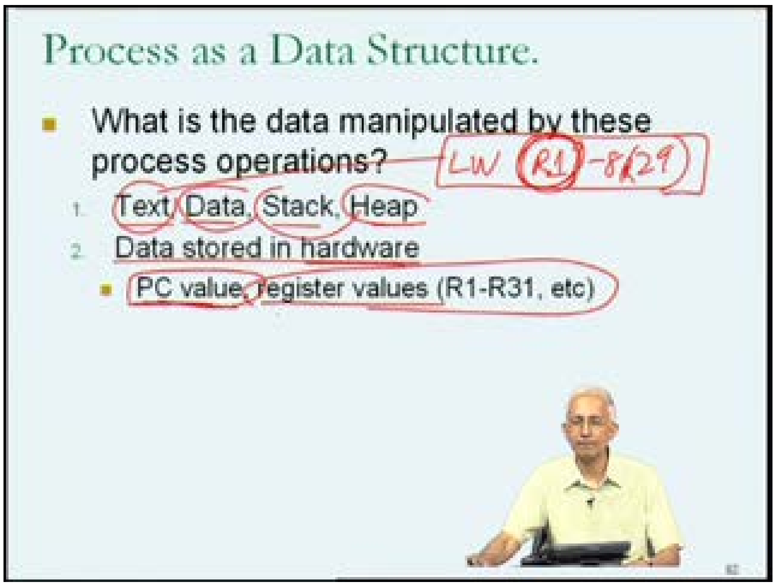


Process as a Data Structure.

- What is the data manipulated by these process operations?
  - Think about the fork() operation
    - Create a new process
    - Text, data, stack, heap copied from its parent process

So, it looks to us like this text, data, stack and heap, which were created for the child process, as part of its birth, its forking, are in fact data associated with the process. They are part of the data associated with the process and in fact, in the case, its specific case of fork, the text, data, stack and heap are almost copied from the parent process into the new child process. But we can certainly view text the contents of the text, data, stack and heap, regions of memory, as being data associated, with the process. They are the data of program in execution and hence they are clearly the data associated with the process.

(Refer Slide Time: 14:46)



Process as a Data Structure.

- What is the data manipulated by these process operations? LW R1 -8(R29)
  1. Text, Data, Stack, Heap
  2. Data stored in hardware
     - PC value, register values (R1-R31, etc)

So, in talking about what data is manipulated by the process operations, we have seen that fork manipulates text, data, stack and heap. In a discussion of exec, we also saw that exec manipulates certain. Does it manipulate text? Yes, because it causes a new program to be overlaid over the text segment.

So, it manipulates text, data, stack and heap, so when a child process executes exec, once again text, data, stack and heap are changed or modified. In the case of fork they are copied, which is an operation on that data. In the case of exec they are modified, they are changed.

Text, data, stack and heap are certainly part of the data associated with the process. They are manipulated by some of the process operations. You can see that I have numbered this one (Refer Slide Time: 15:31) suggesting that they are several more to come. This was the piece of the parts of These were the pieces of data, text data, stack and heap associated with the process, which we readily understood since the text related to the program that we had written, the data relates to the global, the statically allocated variables in our program and since I have written the program, I relate easily to the data.

The stack relates to the local variables and parameters of the program, and if I have written the program, I relate closely to the contents of the stack. I can visualize that are now function called new elements are added to the stack and so on and similarly, with the heap. So, this was an easy step for us to understand the data associated with the process. These are very close to the program that we wrote in order to achieve whatever the objective of at hand was. Now, the rest of the data associated with process when we viewed as a data structures slightly to be less obvious. Let me just give you one example. Now, we view the process as being in the simplest form and data in execution, which means that I am sorry a program in execution, which means that when a process exists, it is manipulating the data by executing instructions and therefore, at any given point in time, they could be data inside the hardware, in other words, inside the program counter, inside the instruction register, inside the general purpose registers, and various other places, inside the hardware associated with a process.

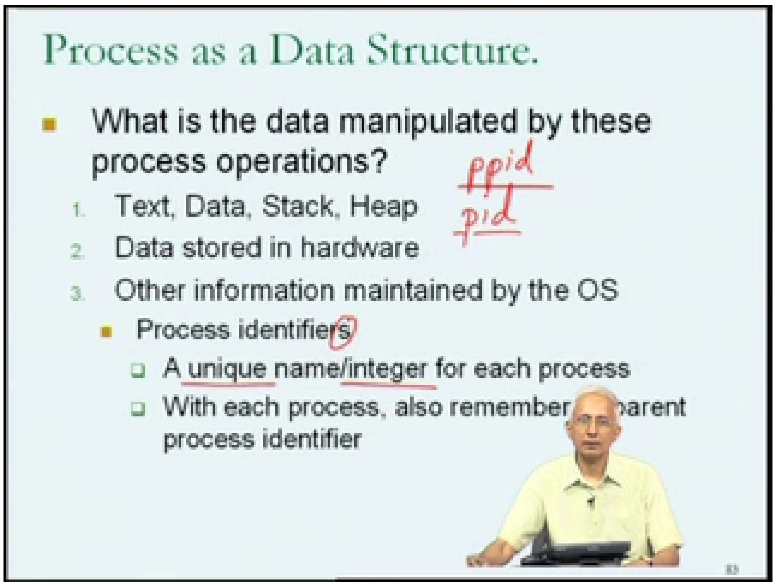
And this we readily understand, because if I have a program and if one of the instructions of the program is a load instruction, then the effect of this load instruction, which is one of the instructions of the program. This instruction could be executed when the program is… This particular instruction could be executed and this is going to have an impact on register R 1. In other words, the data stored in the hardware or the state of the hardware.

So, in thinking about a process, we realize that the contents of the PC register, the contents of the I/R register, the contents of the general purpose registers, if they are floating point registers, their contents too, are all actually data associated with the process. It is this particular process in execution, executing L W- load word R 1, minus 8 R 29, which cause the value inside hardware register R 1 to change.

And therefore, the value inside hardware register R 1 is to be associated with the process, and hence I included in this list of data associated with the process, as a second item that is sort of easy for us to understand. And once again, it is easy for us to understand, because typically many of these pieces of data in the hardware, such as a general purpose registers are directly, because of the instructions are executed by our program like the load word instruction.

Similarly, now that we understand the different special purpose registers in the hardware, we realize that the current value in the program counter is associated with one process, cannot be associated with other processes. It has meaning only for that process, which caused that particular program counter value to have meaning through the fact that it was currently executing a particular instruction, which had as it address, the value inside the program counter value, so the first two items again are easy for the programmer to understand.

(Refer Slide Time: 18:54)



Process as a Data Structure.

- What is the data manipulated by these process operations?
  1. Text, Data, Stack, Heap
  2. Data stored in hardware
  3. Other information maintained by the OS
     - Process identifiers
       - A unique name/integer for each process
       - With each process, also remember parent process identifier

Now, we now have to switch to things which are not obvious from the program, but might become obvious from the perspective of the fact that we now know there is a important piece of software called the Operating System, which is allowing the different pieces of hardware present in a computer system to be shared among multiple processes. It stands to reason therefore, that at any given point in time the operating system must be maintaining information, regarding the sharing of resources among many processes and for each process, it will have to maintain some information, about the sharing of resources, in connection with that process.

So, for example, we expect that it will be necessary for the operating system to actually distinguish between processes, and in order to distinguish between processes, it may have to have names or identifies for the individual processes. Now, since we are talking about software here, it is quite likely that this process identifies will be small integers. But, the actual value of the process identifier, process identifier we will probably abbreviated as, p i d standing for process i d or process identifier, and once again we will think of this as being an unsigned integer. But, for a process to, for an operating system to manage hundreds of processes, it must really distinguish between the different processes, and the easiest way to do that is by associating a small integer, unsigned integer value with each of the processes.

Very clearly, this is for any one process, its process i d is a constant, but it is the value associated with that process, and from the operating system side, it is an important data value associated with that process and therefore, we clearly have to list it here. Now, what other kinds of identifiers associated with the… You have noticed that I have put this down in the plural, suggesting that not only what the process have to have information about its i d, but also possibly about some other identifiers..

Now, this is an interesting point. We will look back and we wonder, is there any situation, where one process has to interact with another process and may therefore, have to have associated with its data, some idea about the identifier of some other process and we saw for example, that in the wait system call the parent process waits for something to do with its children, which means they are associated with the parent, is possible that its child process identifiers may have to be remembered.

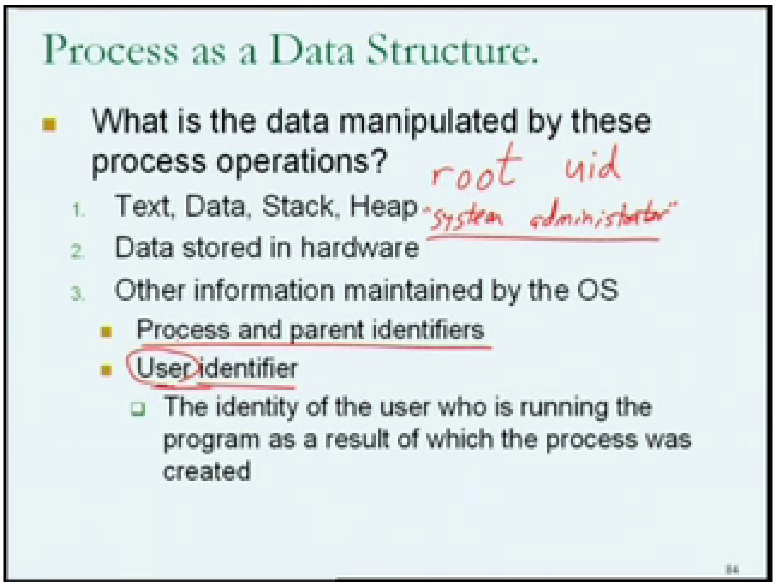
And similarly, associated with the child process, its parent process identifier may have to be remembered. In fact associated with any processes, we might expect to find its parent process i d, as an important constant value, associated with it and parent process i d is often abbreviated as, p p i d.

Now, I mentioned p i d and p p i d, because as you work on a Linux or Unix system, you will find out that it is in fact possible using operating system provided mechanisms to find out what the p i d of your program in execution is, or to find out what the parent p i d of your program in execution is, and it is quite easy to find out how to do this. But, the very fact that the operating system provides mechanisms, like system calls, for a program in execution to find out its p i d, for its parent p i d, means that this knowledge is sometimes useful to a programmer and may use this in the writing of the program.

So, it is possible that more than one process identifier and this would be constant values for any one process, its process id, its p i d is a constant, it is parent p i d is a constant, and these have to be viewed as pieces of data associated with a process, possibly manipulated or read by some of the operations on the process, such as the system calls. Now, one other kind of information, which is… Let me just note that process identifier is a unique name for each process as I mentioned the operating system would have to distinguish between all the programs in execution or processes.

Therefore, would have to associate with each unique identifier and this is typically in the form of unique unsigned integer. So, by unique, I mean that each process running on this, each process on the system, at a given point in time, would have a different p i d. Now, another possible kind of identifier that may be necessary for the operating system to manage the operations associated with a process is identity of the user who cause this program this process to come into existence.

(Refer Slide Time: 23:27)



Process as a Data Structure.

- What is the data manipulated by these process operations?
  1. Text, Data, Stack, Heap
  2. Data stored in hardware
  3. Other information maintained by the OS
     - Process and parent identifiers
     - User identifier
       - The identity of the user who is running the program as a result of which the process was created

root uid

"system administrator"

Now, this is a new concept, which I will just dwell on little bit. It was possible for us to understand process id, from the perspective of, ok, the process is an operating system entity, and therefore, the operating system would obviously, have to distinguish between the different processes.

But now I mentioned using the term, user and you or I when we run, when we log into a system or as users on the system, and this suggest that for every user on the system, there is a unique identifier, and that associated with, when I run a program associated with the processes, which come into existence because of the execution of that program, my identity may also be associated.

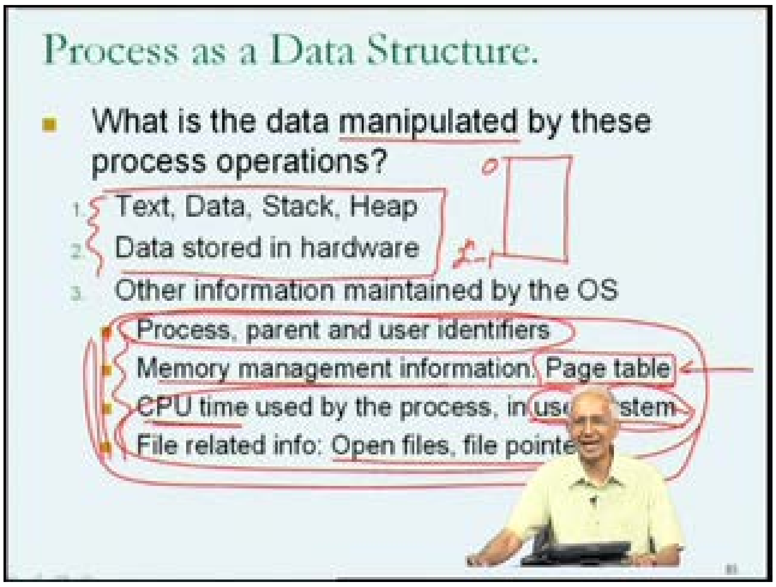
Now, the fact that every user in a system may have a unique identifier, may not come as a surprise, because you know that many of these Unix or Linux systems, you login using login id and a password, and the purpose of this authentication, to make sure that no unauthorized people try to use the system, but also to clearly identify, which user among all the authentic users, which particular user is, has just logged in, and this is often necessary, because depending on the user, the level of privileges may differ.

Among the bulk of users, the level of privileges may be the same. But many of you will be aware that there is particular very privileged user on Linux or Unix systems known as root, and you may have often thought that root is the system administrator, and it is possible or the person who manages the system that is what we mean by system administrator, and it is possible that the system administrator logs into the system using the login id root. But very clearly the root login id, the root user identity has markedly greater privileges on the system than an ordinary user like you or me, because the root user id may be used by the system administrator managing the system, making sure the things are going ok, and that will certainly require more privileges than what you or I would have on a system.

Now, when a system administrator logged in as root, execute a program, it is conceivable that the process, which is execution of that program would have therefore, different privileges, capability of doing different things, from what that of an ordinary user may do. And therefore, associated with each process, it may be beneficial for the operating system in its management of the resources of the system to keep track of what the user id of particular processes. In other words, the identity of the user who cause that process to come in to existence, and once again, this could be a small unsigned integer and every user of the system would have to be assigned a unique unsigned integer, and user id is often abbreviated as, u i d as suppose to p i d for process id.

So, these process and user identifiers seem to be useful from the operating system perspective and it is conceivable, you should know that we could actually write programs, which run on a computer system, and we may not even be aware of the fact that we have a user id or that the programs that we execute run with process ids. But, as I said, may it is conceivable that once we have this knowledge, we may use it, to utilize the information inside these id in some way. And the operating systems certainly maintain this information for certain system calls and therefore, we must list these types of ids as pieces of data, maintain by the operating system associated with a process.

(Refer Slide Time: 27:03)


Process as a Data Structure.

- What is the data manipulated by these process operations?
  1. Text, Data, Stack, Heap
  2. Data stored in hardware
  3. Other information maintained by the OS
     - Process, parent and user identifiers
     - Memory management information: Page table
     - CPU time used by the process, in user, system
     - File related info: Open files, file pointe

So, I will summarize that the operating system will maintain some id, such as a process id, which is a unique identifiers, associated with that particular process. Parent id, which is unique identifier associated with that parent of that process, and the user id, which is a unique identifier associated with the user, the human being, who cause the process to come into existence. So, that some of the information maintain by the operating system.

Now, let us think further. What do we know about the operating system after this point? We know that it is very important piece of software, which manages the sharing of resources, among the programs in execution on a computer system. Secondly, we learned quite a bit, about the way that the operating system manages main memory.

Now, was there any piece of data, was there any data, which was used by the operating system in connection with memory management, virtual page, virtual memory for example, which was specific to a process. If so, we would have list it here and by little bit of recall, we realize that the operating system maintains for each process, a lot of address translation information. You will recall that each process has its own virtual address page, which starts from the address 0 and goes up to some maximum possible address.

This was necessary, if the processes do not actually use physical main memory addresses, because that would make a difficult to protect one process from another and therefore, each process is compiled, so that it runs assuming that it can use addresses from 0 up to 2 to the power n minus 1. Subsequently, these addresses are translated before memory is involved, main memory is involved, and the translation information is contained inside the page table.

There is one page table for each process. Therefore, the contents of the page table at any point in time constitute information or data associated with that process, and as the process, as a program continues execution, the contents of the page table change. For example, when there is a page fault, we saw that the corresponding page table entry may have to be changed, when a page get fetched from the disk into main memory the corresponding page table entry will have to be changed.

We also saw that it is possible that in the act of handling a page fault, one of the pages which is currently in memory, may have to be replaced, may have to be evicted from memory, in which case that page table entry also will have to be changed. Therefore, the page table entries of a process get manipulated due to the memory accesses made by that process and all of this happens due to the activity of the operating system in the task of address translation, and managing the memory.

Therefore, we must view the page table of a process as being information maintained by the OS and relevant to the process and therefore, part of the data of the process. So, we suspect that as we learn more about the different functionalities of the operating system, we will come across more and more such pieces of data, for example, when we studied about memory management, we learn that there is a per processes page table or one Page table per processes. Similarly, when we learn about how the CPU time is managed by the operating system, we may well learn that there is some information maintained in by the operating system associated with that processes, in connection with the management of CPU time. But, one thing, which I have already mentioned to you, is that the amount of CPU time used by a process is actually a piece of information that the operating system keeps track of and that the process can find out. Further, we talked about how when a process is running, it could be running either in user mode if it just running ordinary, the ordinary parts of its program.

But it could be running in system mode, which would be the case if it was executing for example, inside a system call. So, part of the time, your process when it running, may be running in user mode, and part of the time it may be running in system mode and therefore, if you were to ask how much CPU time had been used by a processes, you may be able to determined how much CPU time was spent in user mode, and how much CPU time was spent in system mode. Then you could add those two times together to find out how much CPU time your processes had used and this information is actually available. I had mentioned that it is possible for your program to determine how much CPU time it has receives so far.

And therefore, this information is, obviously, available in the data structures or in within the data kept track of by the operating system in associated with your process, and when you ask for it, it merely gives you the information out of those regions of memory, which where keeps this information. So, in connection with CPU time management, we know that there are going to be some elements of data which the operating system keeps track of in connection with each process.

Similarly, when we talk about files, it is conceivable that we will talk about files later, as far as the input in output management task of the operating system is possible that we will find out that the operating system keeps track of many pieces of information in connection with each process relating to the files that it is manipulating.

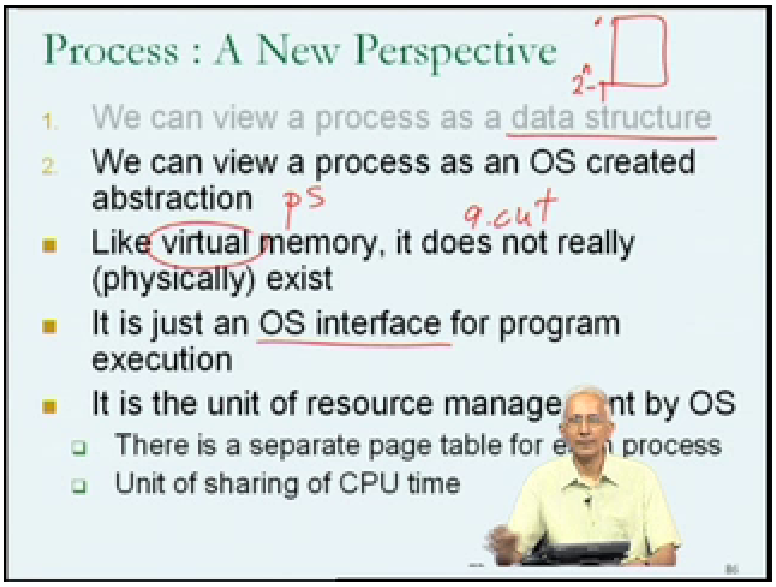
In short, there is going to be a fair amount of information, maintained by the operating system about each process, in addition to the text, data, stack and heap or the contents of the hardware registers due to that processes and putting all this together, we see that there is actually a lot of data associated with any one process and saw for each of the operations on processes some of this data would actually be in manipulated. By manipulated, I may mean just reading the value, for example, the process and parent ids and user ids are not going to be modified by a program that you would write, they are just going to get may be read. So, manipulation could mean either reading writing or both.

So, at this point we have three main items and in the third item of data associated with the process, we have various kinds of data objects, which the operating systems will keep track of in connection with that processes, for the various tasks with the operating system has to do. In short then, it does not seem unfair to think of a process as a data structure. There are well defined operations, in other words, the system calls associated with a process and there is well defined data, text, data, stack, heap, the data store in hardware and the various pieces of information maintain by the operating system in connection with that process.

So, this was not a bad way to look, think about a processes. What is the advantage of viewing the process as a data structure? Now, in going through this exercise of trying to think about the various pieces of data that have to be or that might be associated with the process, as I said, we used our knowledge of our program, but we also began to understand that knowledge of the operating system that we have so far, is enough to give us some idea for example, the idea that page table might be available.

So, this gives us little bit of a feel into what is happening inside the operating system, before we actually go there. We are already seen about page tables, but for the others, we expect, we can anticipate that we are going to see pieces of data that the operating system maintain some information about your process, about each process, in order to manage all the processes. So, in short, it seems to be fair to view the process as a data structure, just like a stack as a data structure in your program a process is the data structure from the perspective of the operating system.

(Refer Slide Time: 34:31)



Process : A New Perspective

1. We can view a process as a data structure
2. We can view a process as an OS created abstraction

- Like virtual memory, it does not really (physically) exist
- It is just an OS interface for program execution
- It is the unit of resource management by OS
  - There is a separate page table for each process
  - Unit of sharing of CPU time

So, in short, let me say that we can view a process as a data structure, but again we want to get a better perspective on processes. So, let me just add another possible view of a process. When I talk about the processes being a data structure, I am talking about something, which actually exists because in thinking about the data structure, we talked about operations on it. In other words, the ways that it is manipulated, and we talked about the data associated with its implementation.

If you look back, (Refer Slide Time: 35:02) you notice that all of these items that we have listed here are associated with the implementation of a process, until we talked about the operations on a process, we thought of a process as a program in execution, which was in some sense an abstract concept. It is only when just like in the case of the stack, when we talked about the stack as a stack of books on a table, it was an abstract concept from perspective of our programs. Our programs really deal with stacks of books on tables. So, it is just an abstraction or an abstract perspective on what what the stack is.

Similarly, when we looked at the process, it was useful to view it as data structure. In order to understand what may be happening inside the implementation of a process, inside the operating system. But if we choose not to take that implementation perspective of the process, then, what we are left with is that we could view the process as an operating system created abstraction. It something, which we need not even know about how it is implemented. Just as the perspective of the stack, as stack of books, on a table where we did not worry about the fact that memory had to be allocated for it or that there had to be a stack pointer. We did not have do this, because we were not worrying about the implementation. We just wanted to use the stack in our understanding of programs.

And you should note that you could write program, which use a stack and your programs could use somebody else is implementation of a stack. So, you do not really have to know how to implement a stack. So, this perspective of the process as an operating system created abstraction, may be relevant to us, in the sense that we are not ever we will… it is unlikely that we will ever be in a situation where we have to create an operating system and therefore, have to create the core to implement a process or decide what data structures we would have to declare in order to have a process.

So, this is a problem that operating system designers would have to handle. For us ordinary programmers, this may not be the case, and it may be adequate, if we just view the process as an abstraction. The way that we view the stack as an abstraction in the sense of a stack of books on a table, so in that sense, if I look back at virtual memory, I could think of the virtual memory prior to our discussion of virtual memory, where we learned about virtual memory, physical memory, the fact that the disk is involved, the fact that page tables are involved, the fact that MMU, a piece of hardware is involved, that inside the MMU there is a TLB, a translation look aside buffer to store some of the page table entries, all of which physically existed. But prior to that we were actually viewing virtual memory as something, which did not physically exist; we have viewed that virtual address space of a process as being an imaginary concept that each process could have the perspective of addressing instructions or data, which started at address 0 and went all the way up to address 2 to the power n minus 1.

This did not physically… From our perspective, it is not necessary to understand how this physically exists; we could just view it as a concept, something which does not exist. Hence, the name virtual, virtual in this context means imaginary, because to the programmer who does not want to know about it, it does not have to physically exist, we do not have to know about the implementation of the paging system.

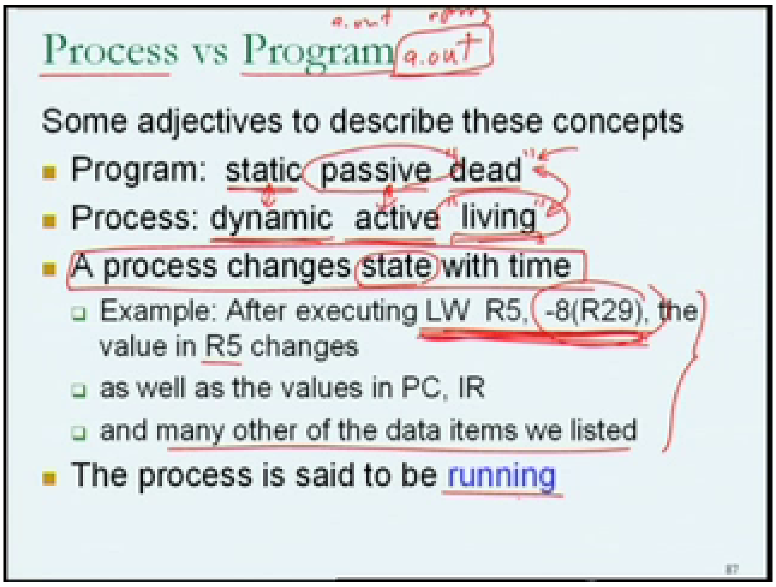
As we are going to see a little later, the knowledge that there is page virtual memory may be useful to us, in improving the quality of our programs. In other words, may be making them faster or more energy efficient or smaller, but it is not necessary for the functional correctness of our programs, which is why this is not a bad perspective to view the virtual memory or the operating of the process as imaginary, virtual things. From this perspective, they are going back to the idea of the system call interface, we could just give the process as this interface with the operating system for program execution. So, I have a program say a dot out and I want to execute this program, and I know that when this program executes, I may have to interact with the operating system through an interface and basically, I can view the process as being one of the terms within that interface, and that may be about it. I just have to know about the process in the context of the interaction paradigm that will be used by the operating system for me to do things to my program in execution.

So, from this perspective, we would view the process nearly as from the perspective of myself as a user, whose running a program on a computer system. I know that the process is a unit of resource management by the operating system. I know that for example, when I type p s, I can get information about my program in execution on the line, where the process with the particular process i d, associated with my program in execution is listed.

And we saw that there was a lot of information, which could be available from p s, in terms of the resources, used by my program in execution and that the operating system is basically managing, for example, virtual memory in units of processes, each process has a page table, is managing the CPU time in units of processes and so on. So, the different resources of the computer system are managed by the operating system on a process basis. So, in that sense, we can view the process as being in this interface with the operating system from the person running programs, just as unit of resource management as for example, there is a separate page table for each process.

So, this is a slightly more, this is a higher level, clearly a more abstract perspective on all the processes and would be adequate for us, for the normal use of the computer system, but may not be adequate for us from the perspective of improving the execution of our programs, because we know now that our programs do have to share resources, with other programs, in execution and the deeper knowledge, may help us to improve the characteristics of our program in that sharing, as we are going to see the process is also the unit of sharing of CPU time.

(Refer Slide Time: 41:03)



Process vs Program a.out

Some adjectives to describe these concepts
- Program: static passive dead
- Process: dynamic active living
- A process changes state with time
  - Example: After executing LW R5, -8(R29), the value in R5 changes
  - as well as the values in PC, IR
  - and many other of the data items we listed
- The process is said to be running

So, with this two perspectives on what a processes is, let us try to conceptualize what we have learnt in todays lecture from the perspective of distinguishing between what a process is and what a program is. Now, this may seem like a trivial exercise, but may be useful from the perspective of understanding. So, we know that you or I write a program. We may write the program in C or java or whatever it is. Let us suppose if you write the program in C, you know that subsequently you compile the program, so that is the program is present in machine executable form. For example, in a file called a dot out.

So, in this slide, when I talk about program, let us assume that I am talking about some abstract version of the program, but I am talking about a dot out, file associated with a program. This is an important distinction. Just note that your program, prior to, before you type in the program into a C file, your program does exist. First your program exists in your mind, a little later, if you are in the habit of writing your programs on paper before typing them in, your program exists on a piece of paper, and that too is your program. Subsequently, you type it into a C file stored in your computer system and that name of that file may be prog dot C and you compile that into an a dot out.

So, all of these, the program in your mind, the program on the piece of paper, program in prong dot C, all of these represent the program, but in this discussion when I say program, let us just assume that I am talking about the a dot out and we have better understanding what I mean by process. Now, that the first thing, which will try to do is to distinguish between a program, and let us say the process associated with this execution. Let us try to come up with some adjectives to describe the difference between being a program and being a process, and these adjectives will help us to understand some important properties of these two. Now, if I was ask to come up with adjectives to describe a program I could use adjectives like good bad and so on, but those do not help us in this context. We will assume that when I say program, I am referring to a correct program, a functionally correct program, and therefore, I mean see all programs are good if they are functionally correct, and the bad programs do not exists, because they have to be corrected, before they become functionally correct. But, I could describe a program as being static, because while the program in a C file, I may modify. I could change, I may run it, and then I may find out there is not good. So, therefore, I modify the program in the C file, but the program in the a dot out file does not change. It is static, in the sense that is fixed.

If I change the C program, I will have to recompile it, and that will generate a new a dot out file, executable file. Therefore, it is fair to think of a program is being static. It does not change, it exists in fact, on the disk and if I want to run it, I have to move it out of the disk into main memory, and that is when this becomes a process. But, the program itself is static. Another term, I might use to describe a program is that it is passive. What I mean by passive here is that it does not initiate anything. You can do things to a program, but the program does not do anything, it is passive.

So, for example, if I have my program in the C program file, prog dot C. I can do things to it, I can add comments or I can change the name of a variable, but the program will never do anything for me in that prog dot C form. Similarly, the a dot out file like in regenerate in a dot out file, but until I run it and change it into something other than a program, it is passive, it does not do anything. I can do things to it; I can change its name to let us say prog. I can do something to it, but it does not do anything on my behalf and therefore, I call it passive.

Now, given that I could call it static or passive, static means that is unchanging, passive means that it does not initiate anything. Things can be done to it, but it does not do things on its own. And might if you go so far as to describe a program is being dead, so this is not meant be technical in any form, but something, which does not change and does not do anything. So, it does not change and it does not initiate anything, you might describe it as being dead, is just another way of looking at the program. Now, as you would suspect, I am going to use terms, which are quite different from static, passive and dead to describe the process, because unlike the program we understand that the process comes into existence, when a request is made to execute a program, and subsequently, the program may run as many processes, and each of those processes are going to have the kinds of properties that were going to try to list through as it is in the next line.

So, as suppose to static, I might describe the process is being dynamic. What do we mean by dynamic? I mean something that changes, it is not in a static state; it is in a dynamic state, it is constantly changing. In what way does a process change? We know that if you look at the list of the different pieces of data associate with a process, text, data, stack heap, contents of program counter, contents of instruction register, contents of general purpose registers, page tables, all of these things as the process keeps running, and the instructions of the program, of which it is an instance execute, as those instructions execute, things happen to the process. So, it is a dynamic, it is a constantly changing.

So, as the instructions of the program of which it associated execute; the process changes. As we saw, its text does not change, but because the text is the instructions of the program, typically the text of program does not change, unless it is exact, using the exec system call, but the data, stack, heap, program counter value, instruction register value, the general purpose register values, the page tables, all of these things are changing as the program runs.

Therefore, it is fair to call the process dynamic, as suppose to the program it is fair to call the process dynamic as suppose to the process which I will still say static program. So, it just on the other hand, if you look at the word passive in connection with process I might use the word active. This process is actually doing things, in the sense that I know that the process is the program in execution. Therefore, the execution of the instructions of the program, constitute the activity of the process and the execution of the instructions of the program cause things to happen, they are its initiating activity. It is not just letting things happen, but it is initiating activity, it is causing the execution of load word R 1, 0 R29, is causing the value of R 1 to change. Trying to load from an address which is currently not in main memory, is going to cause a page fault, it is causing a page fault, which is going to cause the page table entries to change.

All of this is caused by the process and therefore, we could talk about the process has being active. It initiates things, it causes things to happen, as suppose to the program. In short, giving these two meaningful adjectives, so these are like opposite here, I might think of the process just as I thought of the program is being a dead entity. I could think of the process as being a living entity, living in the sense that during its life time, it is constantly doing things, it is constantly changing.

These two terms are not entirely satisfactory, because whether something is dead or living would depend on the time scale that one is viewing the object, for example, if you go to a forest and you look at a tree. You may assume that the tree is dead, but within little knowledge about the tree, you realize that the tree is living. It is respiring; it is doing things relating to nutrition and so on.

And it is timescale may be a little bit different from what we may observe of the tree during our observation period. But in some sense, that is the reason that I put these words between quotes, because in some sense, the distinction between dead and living may be in the eyes of the beholder, but very clearly, now we would think about the process from now on, as being this active, dynamic, entity for which the word living may be appropriate.

So, the key property that we have figured out at this time is we will view the process, as being something which changes state with time, and that is what the word dynamic meant. It was inferred, by the fact that it does things. In order to do things, it had to change state, and is in assents what we meant by the word living. The idea that it is changing state with time, as suppose to your program, so the right way to think about dead versus living, might be to think of from our perspective, we often think of a stone or a rock as being dead because it is unchanging. Again, you can take a stone and throw it, but that is something which you do to this stone. It is now something, which it did itself as suppose to the tree, which is living.

So, this is a very key concept. Process is constantly changing state with time, and simple example, we know that if a process executes, process causes the execution of load word R 5, minus 8 R 29, there is change in the information associated with the process. The value inside R 5 possibly changes. This is simple example of how it changes state. So, am using the word state here, but, it changes its condition, the values of the data associated with it. That is what I mean by state and here the value of the data in register R 5, is a value associated with that process.

Now, in addition to changing the value R 5, you will remember from our execution of the hardware that this particular instruction, the execution of this particular instruction, also causes the program counter value to change, and the instruction register value to change, because prior to the fetching of this instruction, you know with the first step in executing this instruction was to fetch it, and prior to that the program counter value was, after the execution of this instruction, the program counter value will be different, would be incremented and the contents of the instruction register too will be different.

So, this single instruction causes a fare amount of change to the state or the data associated with the process, from the perspective of our data structure view point of the process. So, not only the register R 5, but the program counter, the instruction register and possibly many other data items. Whatever, the data items associated with the process might get changed by the execution of this load word instruction, well we clearly saw that this load word instruction, could well cause a page fault. This load word instruction, which is generating a memory address and in connection with the translation of the memory address, it is conceivable that they would be a page fault and therefore, a page table entry could get modified and so on.

So, in short, to describe this idea of a process, as being this living entity that is changing with time, we could actually use the word running. So, when a process is in this kind of a situation, where it is executing instructions and causing its different data items to change, we could refer to that process as being running. I am putting this word in blue (Refer Slide Time: 52:21), because this is actually a technical term, in subsequent lectures, when we talking about the operating system, we will use the word running in the technical sense.

So, with this then, let me just sum up by pointing out that in today's lecture, we have talked about the process as the fundamental unit of the operating systems management of the resources of the computer system, in sharing them among many programs in execution, and we understand that we can view the process as being this active entity, which is in some sense living. The key property is that it is constantly changing state. So, if it is currently running, in other words, it is causing instructions to be executed, then it is causing state, it is causing changes to the pieces of data associate with process.

And in trying to look at the process as data structure, we had gone through the exercise of listing, the way just kinds of data, which either the program, the programmer or the operating system, associates with any one process. In the coming lecture, we will have a little bit more deeper look, in what is involved in, the way that the operating system shares the CPU time, among the different processes which are active on a computer system.

Thank you.