

**Information security -IV**  
**Prof. Kamakoti V**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture -58**  
**Address Space Basics (Ref: IS-2 and IS-3)**

So welcome to the next session, as we just indicated towards the end of last session. We had team, three important steps that make the meltdown attack one is that every process has address space, in which it is own data code and stack herself, but then there is a space allocated for the kernel.

(Refer Slide Time: 00:26)



**Meltdown: A 3 step attack**

1. (OS) Address Space basics: How address is allocated to a process for its execution? How one process is protected from another? Is it fool-proof?
2. (HW) Out-of-Order Execution: Instructions are executed by the processor in an order different from the actual sequence. Why? How attackers can take advantage of this?
3. (Micro-Architecture) Cache Memory attacks: A timing side-channel attack on the cache which allows the attacker to read a memory address that he is not supposed to.

NPTEL

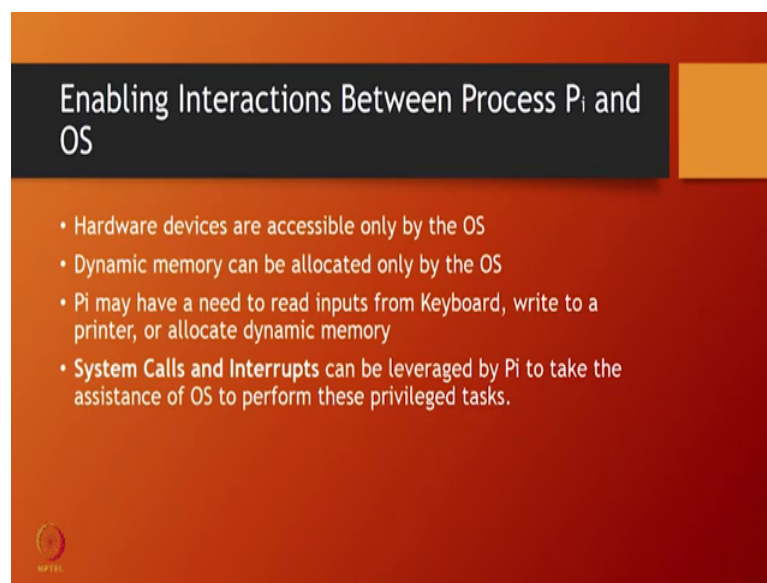
So, this is true for every process and what happens is the space which the kernel space the operating system space is not exclusive for every process. So, two processes can say, can basically, share the same kernel space. So, as process P1, if I write some secret into the kernel, which is stored in the kernel another process P2 will have, if it can have access to the kernel space, then it can read my secret. So, by this here process P ones, secret can leak to P2.

So, that is the first part and the second part is there is an hardware, out of order execution, we will also deal with that in some detail, I have already dealt with it in I S 2, the information security 2 course, but I will give a brief outline of that and that is also exploited and then there is a micro architecture level hack, that is done, where the, cache

is basically, utilized to leak information. So, these three steps, are forget that this is a it is a security vulnerability, which is causing a lot of concern, but from the, other angle it is a very brilliant way of breaking into the system. So, let us now deal each of this, in quick succession. We will just see, what, what is in store for each.

Now, first one let us start about the address space basics I have just talked about it in the previous sessions, in this course itself, but I will just summarize that very quickly and whatever I am summarizing has been part of the information, security two course and I information security three course. So, if you find something that you are not able to quickly follow I also suggest that you go back to those, videos and have a look at it, to have a much broader, deep understanding of the concepts, that I am, I am going to cover, in this part.

(Refer Slide Time: 02:37)



The slide features a dark red background with a lighter red header bar. The title 'Enabling Interactions Between Process P1 and OS' is written in white text on the header. Below the title, there is a bulleted list of four points in white text. In the bottom left corner, there is a small circular logo with the letters 'NPTEL' inside.

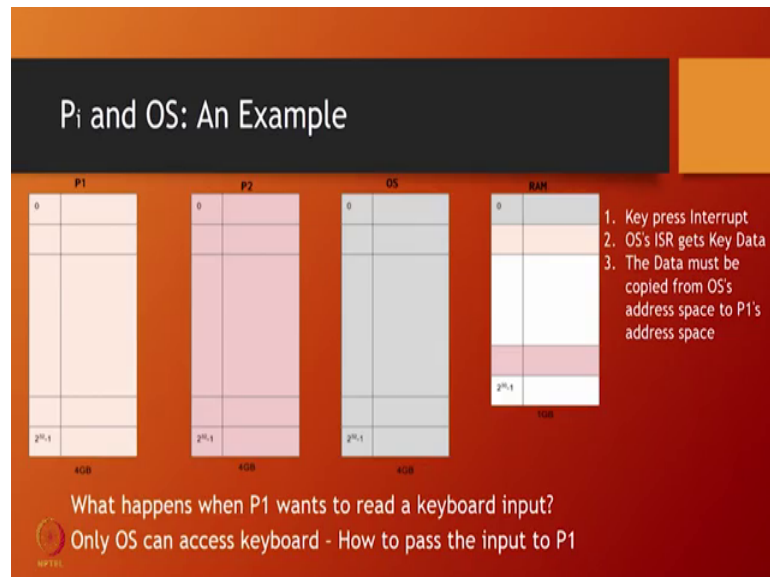
### Enabling Interactions Between Process P1 and OS

- Hardware devices are accessible only by the OS
- Dynamic memory can be allocated only by the OS
- P1 may have a need to read inputs from Keyboard, write to a printer, or allocate dynamic memory
- System Calls and Interrupts can be leveraged by P1 to take the assistance of OS to perform these privileged tasks.

So, there is a need for a process P1 to interact with the operating system, whether it want to print some value, it wants to take some memory like malloc, etcetera. So, or it is wants to accept a value from a keyboard, everything. There is a need for the operating system for the process to basically look into the operating system for help and when there is, when, when the operating system executes a system call right, when the application executes is a system call or there is an interrupt, the operating system, the interrupt service routine system call is also like an interrupt to the operating system.

So, your system calls and the interrupt service routines are also part of the operating system. So, there is lot of interaction between the application program and the operating system. So, let us now go and look at very clearly.

(Refer Slide Time: 03:30)



So, when, when a process starts executing essentially from the process point of view, that is, there is some memory space, which is that 4 GB. Let us take, it is a 32 bit architecture. So, 4 GB address space. So, if I am a process P 1, I am given a 4 GB at the space. Similarly, P2, another process P2, it is also given 4 GB address space right and then OS, OS also will get a 4 GB address space, but then, then the actual RAM can be 1 GB right. So, this is a concept of virtual memory. So, every process will get the whole 4 GB and then, but the actual, HM thing would be only 1 GB.

Actual memory can be 1 GB right and this is how paging works. So, paging takes care of you know maintaining these individual, address space for processes OS is also a process pieces, please understand that OS is also a program in execution, it is also a process. Now, let us see what is going to happen. Suppose, this is the stage where I say P1, I cannot give the entire 4 GB for P1. So, I will give some space for P1. I will give some space for P2, I will give some space for OS ok.

So, this entire 4 GB address space will be, you know shared and used shared in the sense that some part, it can be exclude, it is exclusive. So, let us say, I do not want to, share a same, virtual address between three processes. So, suppose P1 requires some, some,

some amount of space. So, P1 will have its space. So, in the corresponding page table those pages alone will be enabled in the corresponding page table of P2. Those pages will be enabled in the corresponding page table of OS, those pages will be enabled right. So, if you have understood paging, you will understand what I am talking here. So, if, for example, P1 is say 16 K B, each page is 4 K B.

So, I will have 4 pages for P1 and I will have say, let it, this be 30 kb. I will have 8 pages for P2. Let the OS be some large one, then I can have 20 pages, 25 pages for OS, let us just talk. But then I will ensure that the pages are not the same, the same page is not, you know shared so; that means, with respect to the virtual address space, each person has got a separate isolated memory. Now, if that is the case, now let us just take P1 and the OS the P1 and OS. So, P1 will not have any access to the OS pages right so because; in the P1 space table there will be no entry corresponding to where the OS pages are right.

The OS pages also will be in that 4 GB address space, P1 HM pages also will be in the 4 GB address space, but the entries for P1, in the P1 space table will correspond to only those pages that are allocated to P1, the other pages it can never access and. So, the OS cannot be accessed by P1. So, this is an ideal scenario, if that is a scenario, now, let us take a very simple example like a keyboard, you press some, say some character in the keyboard, where it will go. It will go and basically, gets saved in the OS address space.

So, in some page corresponding to the OS, there will be a buffer, in which this whatever character you have entered will go and stay there. Now, who is asking for that character? I am an application program, I am doing a scan of. So, I need that value, I as a process. So, P2 needs that value, but that value, when somebody presses the keyboard then there is interrupts generated and what will that interrupt do? There will be interrupt service routine that will be actually the keyboard controller or whatever and then the data that is typed actually, comes and stays in the OS address space. So, then finally, the data is needed by the, process P1.

So, we have to copy it, exclusively from the OS address space into the, , into the user address space, or the process address space and then P1 has to read. So, there is an extra memory copy that needs to happen and it needs to happen for every character that you are entering and that thing is actually from one page table to another page table. Since,

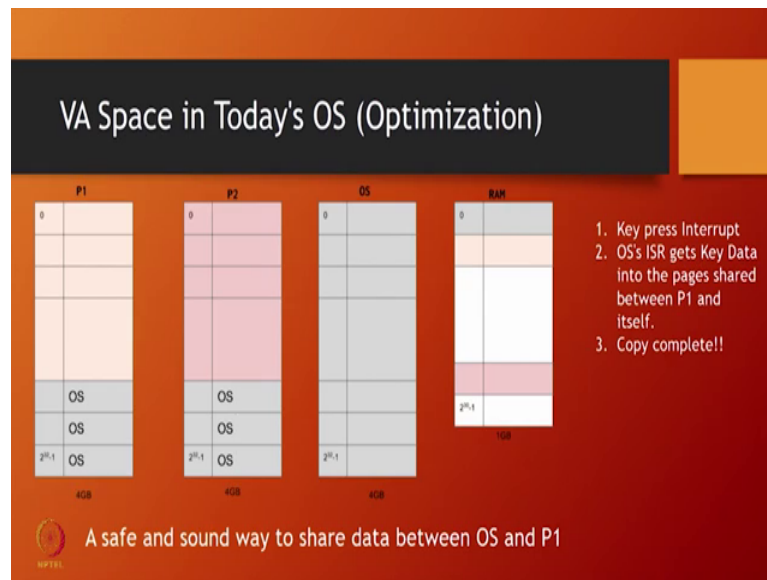
this page table will not have any entry about the OS pages and your, your values actually stored in some place, which P1 cannot see and so, now, OS has to transfer it back there.

So, this extra copy is needed and all this, because P1 does not have the page table for P1, does not have an, entry for the operating system right. So, this will really have a big, performance hit and the solution, which you know the different advisories are basically suggesting, to handle this meltdown. Attack is very close to this saying that you have it is separate address space and then you copy from that to this, it is very close to it, not exactly, but we will say what exactly their solution that they are presenting, in the later set, but this is very close to this.

So, if I so, essentially it is some of this slide, if the process P1. So, every process has its own page table that we have seen in the earlier class, earlier session, where we add this task stage segment that is a C R 3 entry their the C R, three entry points to start off a page table. So, every process including the OS, which is also a process can have a page table of its own and in the page table of process P1, I have entries only, corresponding to pay process P 1s data codes and stack in, in the, in the OS, I cannot see the OS page.

If that is the scenario as we are seen here, which is very ideal, a meltdown cannot happen in this type of a context, then every, every is for example, a simple thing like a keyboard I O, keyboard input should cross at least two stages before it reaches the application and that is going to have a serious, performance hit. So, this is a costly operation as you see in the screen right. Now, let us say.

(Refer Slide Time: 09:42)



So, what has happened is today, what they have done is to optimize this that is the reason, we have been saying the process, in this page table for the pro, each of the process, there is an OS, pages also loaded there. So, I need not go to a separate page table, look for where the OSs and then collect the data and then come back to this page table and then store it in some location.

Now, page table of process P1 has something for the process, which is on the top and whatever is shared at in the down is completely this OS and when I switch to process P2, again the process P2 pages will be there and then the same OS pages the same link will be there. So, the same page table can be at the same page entries are replicated here so; that means, P1 and P2 will share the same set of OS pages.

So, as you see here on the screen, the entries for the OS, the last three entries in the P1 page table, will be the same as the last three entries in the P2 page table. They will essentially, they are pointing to the same page. So, though I have two different page tables, one for P1 and another for P2, they, that the pages, that they use for the kernel, the pages that they use for themselves, they are isolated, but the pages that they use for the kernel are basically shared.

Now; that means, if P2 write something into the kernel P1, space table will have a pointer to access that right. So, if I start accessing it, whether I will get it or not is

another question, but there is a, there is and there is an access to the page table of P 1, I can go and access what P 2 has written into the kernel right.

Now, so, far what we all of us believed was that, as a process, I cannot go and access the OS region P1 cannot reach the OS region, simply P2 cannot and that is precisely, what meltdown says that, it is now, possible and that we will see how it is going to be possible and how the information can be leaked ok.

So, this is the scenario, but now, every operating system does this type of a thing, because of what, because of performance reasons right. So, if I had isolated O S, is isolated from P1 and P2 address spaces then I O becomes extremely complex, just to get rid of this, this, now becomes a safe and sound way to share data between OS and P1 as of now, but with meltdown that safeness, and safety and soundness has literally melted out ok.

Now, let us see, how that is going to happen? Now, in this, virtual address space, if I have a keep, keyboard interrupt basically, the data now goes and stores in the OS address space, but the OS address space is, accessible by both P1 and P2. So, the so, now, the copy there is no extra copy needed, across different page tables, it is not just copying from one memory location to another memory location.

Now, this whole memory access is governed by paging and. So, there is one page table and there is another page table, from one page table entry to another page table entry. I have to need, do a copy. So, that, that, that really makes the whole thing much complex.

(Refer Slide Time: 13:12)

**Can P1 access the OS space Today? NO!**

- Hardware allows two modes: user mode and supervisor mode.
- Hardware bit that indicates the mode is called "supervisor bit".
- Supervisor mode has extra permissions compared to user mode.

Steps Involved in accessing OS pages

- P1 tries to access the OS pages in its Virtual Address space.
- Hardware checks if the page can be accessed in user mode (for P1).
- OS pages can be accessed only in supervisor mode (as per page access permissions).

Exception is raised by the hardware to prevent access.

So, far as I just summarized. Now, can P1 access the OS space today, as of now, no why. So, the way we have believed. So, far is that no, it cannot access. Why? Because this again, I have explained way in much detail, in the I S 2 course, the hardware allows two modes, user mode and supervisor mode. In actual Intel, we have four privilege levels, privilege level. 0 is the supervisor mode the highest powerful P L 1 P L 2 P L 3 right.

So, there is an hardware bit that indicates the mode and this is called a supervisor bit. You can see this in all your, descriptor tables, etcetera right. The, the supervisor mode will have, extra permission compared to user mode for example, it can in execute privileged instructions. You have seen, what are privileged instructions in the information security two course right. So, we can keep executing privileged instructions and so far.

Now, what is the step involved in actually accessing the OS page as a process P1, actually tries to access the OS page in it is virtual, address space. Hardware checks, if the page can be accessed in user mode. Now, it cannot be. So, essentially an exception is raised and the exception handler will take care of, whatever P1 wants and this is precisely what we also call as a system call. So, malloc, what I need? I need some memory. What I mean? I need memory so, the operating system. So, when you are executing program suddenly, you read lot of memory. So, you ask for this memory allocate.



Now, the operating system basically, maintains a list of free memories right. So, when I, when as a process I need memory, I just execute the command malloc, which essentially means, I have to go to that list, find out some memory chunk, it will have a list of free memories, means it will have list of chunks of oh, there at address 1000, there is some 5 12 bytes of memory free at address 2000s. There is some 3 M B of memory starting from 2000s free like that it will have the list. So, I need to go to the list and basically get the, get one chunk allocated for V.

Now, the free list will say ok, it will delete that part of the memory and give it to E, give it to the meaning that will be used by, the process. So, I have to go and access the free list. So, when I say malloc as a user process, I want to access the free list, which cannot happen, because the free list can be accessed only in supervisor mode. So, immediately there will be an exception that will be raised when their exception is raised again in I S 2, I have covered that, an interrupt service, routine well start executing and an interrupt service routine will find ok.

This is trying to access this, and this is a system call and. So, the operating system will allocate the memory. It will operating system, well go and look at the free list, it will find out, which chunk of memory I am asking for say 100 bytes, which 100 bytes I can give it to him and then it will remove from the free list and put it in the occupied list and then release back and. So, I can start using that 100 bytes.

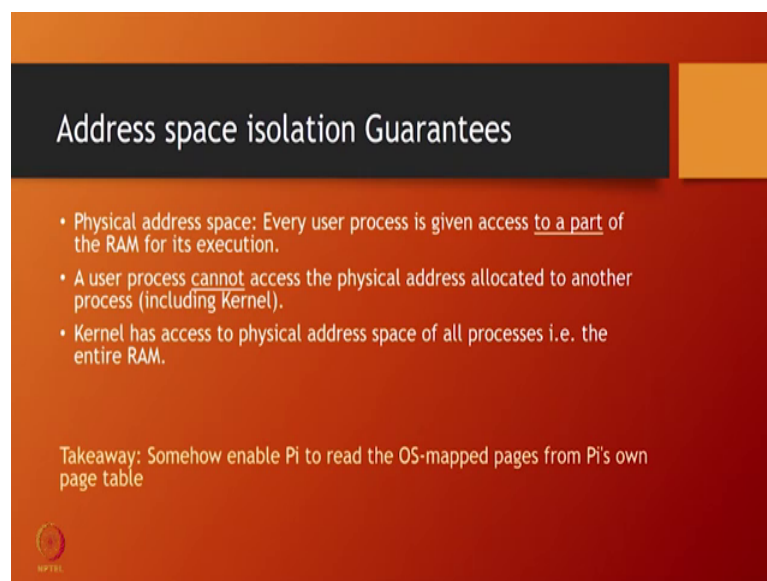
So, this is how the whole thing basically happens. So, that is also another reason why the kernel has to be shared like I asked for some free memory that, that I have asked for that free memory and that is allocated to me should be known to the other process also, if it is not known to the other process say, let, let us say there is a separate OS image for both of us, I asked for some memory then that memory the OS image has given me, it should be known to the other image also right. If it is not transmitted to the other image then there will be no consistency, two processes can get the same image and that is enough for also security vulnerability to end of story for that ok.

So, essentially there are a lot of things, the OS needs to carry, which are global in nature like importantly the free list of memory. How much memory to whom I have allocated and so, that I do not reallocate the same memory to other fellow so, essentially going back to this right. The, the pages were O, the pages corresponding to the OS in the P1

space table that you see on the extreme left and the P2 space table, those entries will point to the same OS pages and the necessity is that P1 and P2 P1, if there is say, for example, there is a memory allocation done for P1, it is very important for P2 to note, because when P2 ask for memory then, the same page should not be allocated. So, that it will go and corrupt that data and it will also not execute correctly.

So, this is also precisely a reason why every OS should have an access to the operating system. Every-every process should have an access to the operating system.

(Refer Slide Time: 17:57)



The slide has a dark red background with a lighter red header bar. The title 'Address space isolation Guarantees' is in white text on the header. Below the title, there are three bullet points in white text. At the bottom, there is a 'Takeaway' section in white text and a small logo in the bottom left corner.

### Address space isolation Guarantees

- Physical address space: Every user process is given access to a part of the RAM for its execution.
- A user process cannot access the physical address allocated to another process (including Kernel).
- Kernel has access to physical address space of all processes i.e. the entire RAM.

Takeaway: Somehow enable Pi to read the OS-mapped pages from Pi's own page table

Now, this type of an address space isolation, where there is an OS address space and then the process space, thing every user process is given access to a part of the RAM for it is execution that is the user space and the user process cannot access the physical address allocated to another process, including kernel. The kernel has access to physical address space of all processes that is, kernel will have access to all the entire RAM, but the vice versa is not to from this, what is going to be meltdown? Meltdown is going to melt this particular shake, this particular isolation guarantee.

So, what it says is somehow will enable the P is the processors to read the OS map page, from P is own page table right and that is what is achieved here and the moment I can go and access my same page table, if I am able to access the kernel page then what will happen; obviously, the kernel is shared, some other process would have written something into that kernel, which I can basically leak and get it out.

So, this is the entire concept of the meltdown, we will now, see out of order in the next session.

Thank you.