Paging and virtual memory

Step : 6

Paging allows the OS to provide each application program running in unprivileged mode with a virtual (or logical) address space. The application's access can be restricted to this address space.
When a program is executed in unprivileged mode, the logical address must be translated on the fly to the corresponding physical addresses. XSM Unprivileged Mode Execution This requires support from the hardware

Conversion of logical address to physical address :

Location of page table entry = PTBR + 2 x (Logical Address / 512)
Location of page table entry = PTBR + 2 x (Logical Address / 512)
offset = Logical Address % 512
Physical Address = Physical Page Number x 512 + offset

Each paging table entry consist of two words

1. Physical address
2. R V W D bits

XSM Unprivileged Mode Execution :

Mov sp, 1000
Push R0
In unprivileged mode R0 is copied is determined in a different way. Content of SP treated as logical address(1001).

Initial setup we need to do before transferring control to program run in unprivileged mode :

Before running an application program in unprivileged mode, the OS must set up the application's page table data and values of PTLR and PTBR.

1. Set the PTLR register to define the address space maximum limit.
2. Set up a valid page table in memory and the PTBR register to the beginning

address of the page table **of the particular application**.
3. Set up the application's stack. Set SP to point to the top of the stack.
4. Compute the physical address corresponding to the logical address in SP. Then, copy the logical address of the first instruction (entry point) that much be fetched after IRET into this physical memory location and execute IRET.

Step 1 :

ExpOs kernel sets the address space of every application to 10 page.
So 0 to 5119 logical address space to every program any thing outside this address space will generate exception.

Step 2:
Each paging table consist of 20 words.
Proper values for Valid (V bit) and the Write permission bit (W bit) must be set by your privileged code of step 2 before starting unprivileged mode execution.
The Reference bit and the Dirty bit are set by the machine.

{
Basically, there must be some pre-defined convention regarding the initialization of the application's stack between the application and the OS. Similarly the OS should know which is the first instruction that must be executed when the application is run in unpreviliged mode in order to transfer control to that instruction.
}

When code pages are loaded valid bit is set to 1 and write bit set to 0 because we can’t modify code during execution

Step 3:

{
how will you figure out where must be code pages and stack pages of the application loaded? Out of these 10 pages in paging table.
}
The ABI convention is that the application code must be loaded to logical pages 4,5,6 and 7.
Thus code area **start at address 2048;**
Stack pages : 8 and 9
Thus starting address of **stack 4096** (before application starts running sp must be set to 4095)

Step 4: Initilize the instruction pointer (IP) for the application

Each eXpOS application can have at most four code pages of machine

instructions. An eXpOS compatible XEXE executable file must contain these instructions listed in sequential order. file must contain an eight-word **header**

{
In the eXpOS system, you will be using the ExpL programming language for writing application programs and the ExpL compiler supplied to you will correctly generate target code and header (including entry point value) properly so that the ABI conventions are satisfied.
}

The IRET instruction :

It first change the mode to unprivileged, then transfer the contents at top SP(It should contain logical address of IP) into the IP register and decrement the stack pointer

What happened when software interrupt occurs:

Interrupt vector table is stored starting from physical address 492 of memory

Your bootstrap loader must load these interrupt handlers from the disk to the appropriate memory pages.

Assignments

We will INIT program which must be stored in block 7 or 8 of the XSM disk(See Disk Organisation). Loaded using XFS-interface. We will write the OS startup code such that it loads the INIT program into memory and initiate its execution at the time of system startup.

Code for the INIT program will loaded on dick block 7 and 8 (Disk Organisation).
We load this program in memory pages 65 and 66 (INIT program stored at this pages in memory)
Load load corresponding software INI to memory pages.

Now we set up Paging table at PAGE_TABLE_BASE (Paging table starts from PAGE_TABLE_BASE()
Two pages for program and one for stack needed total three pages needed

We want program execution starts from logical address 0.
So we set Current stack top with 0 and do IRETURN. (Stack for INIT program page 76 so starting address is 76 * 512)
Which will set IP with value at top of stack and change to mode to user so paging table will be used for address translation.

Here we are not using stack so we are concerned with stack (SP).