|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | MMU is the ability to manage the programs running in their own private memory space  A task written to run under the control of an operating system with an MMU does not need to know the memory requirements of unrelated tasks. This simplifies the design requirements of individual tasks running under the control of an operating system.      The MMU acts as a translator, which converts the addresses of programs and data that are compiled to run in virtual memory to the actual physical addresses where the programs are stored in physical main memory.    This translation process allows programs to run with the same virtual addresses while being held in different locations in physical memory.  This dual view of memory results in two distinct address types: virtual addresses and physical addresses.  . Virtual addresses are assigned by the compiler and linker when locating a program in memory. Physical addresses are used to access the actual hardware components of main memory where the programs are physically located.      When the processor is instructed to jump to &8000 to begin executing an application, it passes the address &8000 to the MMU. This translates the address into the correct real address and outputs this on the address lines, say &12FC00. The processor is not aware of this, the application is not aware of this, the computer user is not aware of this. | |  |  | Page tables and the MMU  Page table setup is done by OS The page table exists to map each page into an physical address.  This allows the operating system to keep track of which memory is pretending to be which.  Some pages are remapped, doubly mapped, not for user mode, read only etc..    The MMU takes an address, looks it up in the page table, and spits out the correct address.    We'll assume a 4K page size. A 32bit address space has a million pages. With one million pages, you'll need one million entries. In the ARM MMU, each entry takes 7 words. So we are looking at seven megabytes just to index our memory.   * Some machines don't have enough space to accommodate this : * So there are three options:   + Have a huge array of fast registers in the MMU. Costly. Very.   + Hold the page tables in main memory. Slow. Very.   + Compromise. Cache the active pages in the MMU, and store the rest on disc. * Example explain in notes;     TLB - The Translation Lookaside Buffer is a way to make paging even more responsive.   * Typically, a program will make heavy use of a few pages and barely touch the rest. * A solution to this is to fit a little bit in the MMU that can map virtual addresses to their physical counterparts without traversing the page table. This is the TLB. It lives within the MMU and contains details of a small number of pages (usually between four and sixty four - the ARM610 MMU TLB has thirty two entries).      * Now, when we have a page lookup, we first pass our virtual address to the TLB which will check all of the addresses stored, and the protection level. If a match is found, the TLB will spit out the physical address and the page table isn't touched. This is called TLB hit * If the requested address is not present in TLB(i.e miss is encountered) , then the TLB will evict one entry from TLB and loads it back to page table entry. So, TLB will know the new page requested , so it can quickly satisfy the new page access , and chances are the next access will be in same page access. * All this is DONE by MMU H/W         What is a Memory Management Unit (MMU)?    The memory management unit is the part of the CPU that interprets virtual addresses. Attempts to read, write, or execute memory at virtual addresses are either translated to corresponding physical addresses, or else generate an interrupt (page fault) to allow software to respond to the attempted access.  Physical memory addresses are unique in the system, virtual memory addresses are unique per-process.          What is Page Fault ?   1. A page fault occurs when a process accesses a page that is mapped in the virtual address space, but not loaded in physical memory. In most cases, page faults are not errors.   *From <*[*https://www.cyberciti.biz/faq/linux-command-to-see-major-minor-pagefaults/*](https://www.cyberciti.biz/faq/linux-command-to-see-major-minor-pagefaults/)*>*     1. int fault(struct vm\_area\_sruct \*area, struct vm\_fault \*vmf) This function is invoked by the page fault handler when a page that is not present in physical memory is accessed.     What I smethod of recover ?        Kmalloc and Vmalloc  The kmalloc() function’s operation is similar to that of user-space’s familiar malloc() routine, with the exception of the additional flags parameter.The kmalloc() function is a simple interface for obtaining kernel memory in byte-sized chunks.  The function returns a pointer to a region of memory that is at least size bytes in length.3 The region of memory allocated is physically contiguous. On error, it returns NULL.  void \* kmalloc(size\_t size, gfp\_t flags)  GFP\_ATOMIC -> The allocation is high priority and must not sleep. This is the flag to use in interrupt handlers, in bottom halves, while holding a spinlock, and in other situations where you cannot sleep.    GFP\_KERNEL -> This is a normal allocation and might block. This is the flag to use in process context code when it is safe to sleep.      The vmalloc() function works in a similar fashion to kmalloc(), except it allocates memory that is only virtually contiguous and not necessarily physically contiguous. It does this by allocating potentially noncontiguous chunks of physical memory and “fixing up” the page tables to map the memory into a contiguous chunk of the logical address space.  The vmalloc() function, to make nonphysically contiguous pages contiguous in the virtual address space, must specifically set up the page table entries.  Worse, pages obtained via vmalloc() must be mapped by their individual pages (because they are not physically contiguous), which results in much greater TLB4 thrashing than you see when directly mapped memory is used. . Because of these concerns, vmalloc() is used only when absolutely necessary—typically, to obtain large regions of memory. For example, when modules are dynamically inserted into the kernel, they are loaded into memory created via vmalloc(). |
|  |  |  |  |  |
|  |  | MMU Address Translation Process       1. Whenever an address translation is requested (that is, for every access with the MMU enabled), the MMU first checks whether the translation is already contained in the TLB, which acts like a cache storing recent translations.      1. If the requested translation is not in the TLB, the table-walking logic retrieves this translation from the translation table(s), and then updates the TLB. The address translation is then performed.             Diagram  Description automatically generated      Virtual Address = Enter Page Directory -> it gives page table address  then enter Page Table -> That gives offset .    These two will make virtual Address = Page Table + Offset     |  |  | | --- | --- | | Page Table | Offset |   Diagram  Description automatically generated      Paging level: 2 -level paging -> ex: Above diagram  3- Level Paging  4- level Paging | |  |
|  |  |  |  |