Tutorial on Memory Management, Deadlock and Operating System Types

1 Background

1.1 Memory management

Virtual memory: is a method of managing memory automatically by the operating sytem so that the combined size of memory available to all processes running on the computer may be more than the physical memory available by using the hard disk to hold what is not in physical memory.

swapping: involves moving all the content of memory associated with a process from RAM to hard disk, and back as the operating system needs memory to run other processes. This is inefficient for large processes. Results in holes, where RAM allocated to two large processes may have a hole between them which is too small to use for any process.

paging: all memory is divided into chunks called *pages*. All pages are of a fixed size. The pages in RAM used by a process do not have to be contiguous (next to one another), so no problem of holes.

Memory Management Unit: is hardware that sits between the CPU and the system buses, translating virtual addresses used by programs into physical addresses. The MMU is connected as shown in figure 1. The MMU and CPU organisation fix the size of the pages, page tables and various other aspects of paging.

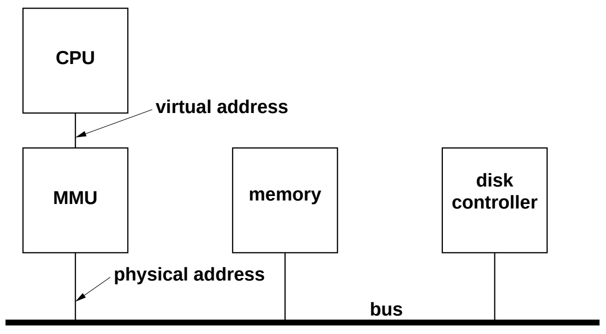

Figure 1: The memory management unit is shown here converting virtual addresses from the CPU to physical addresses.

Virtual addresses: are the addresses used by a user's program. The programmer can write the program as if the program has access to as much memory as required, without worrying about whether the addresses are used by other processes. All addresses in the program are *virtual addresses*, and are translated by the MMU to physical addresses.

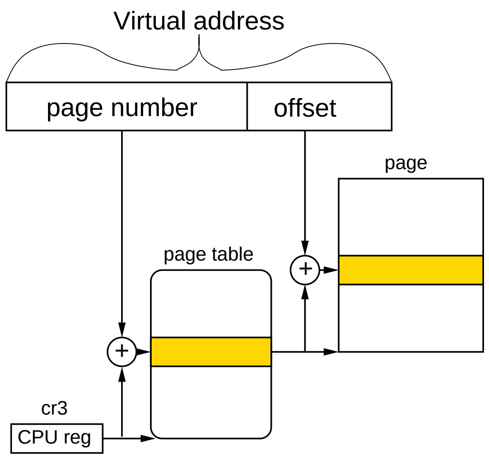

Figure 2: A single-level paging system. Virtual memory addresses are 32-bit. Pages are 4K each.

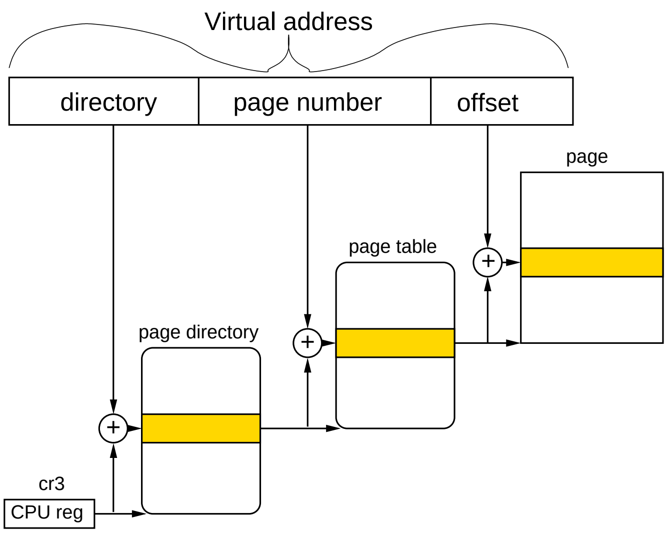

Figure 3: A multi-level paging system. On the Intel platform, virtual memory addresses are 32-bit. Pages are 4K each. The page tables are themselves pages, also of 4K each. Since each page table entry is 4 bytes in size on the Intel platform, there are $1024 = 2^{10}$ entries in each of the page tables. So there are ten bits required for the page number part of the virtual address. The page directory itself is a 4K page, each entry is 4 bytes, so there are 1024 entries, one for each page table. So there are ten bits required for the directory part of the virtual address.

What is going on in those diagrams? To make sense of figure 3, let's look at some of the main points. First, this is the representation of an Intel memory management system. Other architectures have different sizes of pages, different numbers of levels of page tables (e.g, the Compaq Alpha has three levels—see later).

- The virtual address is the address used by any user programs running on the computer.
- The page contains any data used by user programs, including the executable programs themselves.
- The directory table and page table are both pages themselves, and are the same size.

- On the Intel architecture, each entry in the page and directory tables is four bytes in size.
- The most significant 10 bits of the virtual address are an index into the directory table, and are in the range of 0 to $2^{10} - 1 = 1023$. Let's call it the directory number.
- Similarly, the ten bits for the page index number are also an index in the range 0 to 1023.
- The `cr3` register is a CPU register that stores the address of the directory table.
- To determine the address of the current page table entry, the MMU adds the contents of `cr3` to the directory number $\times$ the size of a directory table entry (4).
- To determine the address of the current page table, the MMU reads the content of the current directory table entry.
- To determine the address of the current page, the MMU adds the number $\times$ the size of a page table entry to the address of the current page.
- To determine the actual entry in the page (the content of the virtual address), the MMU adds the offset from the virtual address to the address of the current page table.

Why we need multilevel paging (single level paging won't do): Many people seemed unable to understand why the single level page table shown in the lectures needs $2^{20} \times 4$ bytes of RAM, and why multilevel paging does not. The key is that for virtual memory to work, there must be page table entries for the entire virtual address space. If the virtual addresses are 32 bits long, and each page is 4-kilobytes (2^{12} bytes) in size, then there *must* be $2^{32} \div 2^{12} = 2^{32-12} = 2^{20}$ entries, one for each page. On the Intel platform, each entry is 4 bytes, so the total size of all page table entries is $4 \times 2^{20} = 4\,\text{MB}$. The single-level paging scheme shown in figure 2 on the preceding page shows the page table in one piece, so it must all be in RAM.

It is silly to keep all this in memory at once, since most page table entries are never used. For example, virtual memory in today's computers is unlikely to be 4 GB unless the computer is a very busy server.

The solution is to have only the necessary page entries in RAM, and to have any others that have been used some time ago on the hard disk. *It is more sensible to split the page table into pages that can be paged in and out of memory.* This is what multilevel paging achieves.

Explaining the "Example of paging: Intel x86" The example in the notes has confused many people since many of you do not know the meaning of `0x20000000` (or what `0x`*nnnn* means). The prefix `0x` indicates a hexadecimal value in the C programming language. You can have statements such as:

```
int i = 0x123;
```

which assigns the value 123_{16} to the variable `i`. All that I am showing here is what the components of the virtual address are. The example is simply showing the components of the virtual address `0x20021406` (i.e., 20021406_{16}).

The offset within the page is the least significant 12 bits, i.e., 406_{16}.

The page number is the next 10 bits, i.e., 21_{16}. Note that the only allowable page numbers are 0 to $3F_{16}$, since the process has been allocated pages in the range 20000000_{16}–$2003FFFF_{16}$. If any address used by this process were outside that range, the OS would terminate the process with a segmentation fault.

The most significant ten bits give the directory:

2	0	0	2	1	4	0	6
0010	0000	0000	0010	0001	0100	0000	0110

If you count the ten most significant bits, you get $0010\,0000\,00_2$, i.e., 80_{16}.

The Compaq Alpha architecture uses a three-level paging system: The Compaq Alpha processor has 64-bit virtual addresses, and a 64-bit data bus. It also has 8 kB pages. There are a lot of 8 kB pages in 2^{64} bytes: 8 kB = 2^{13} bytes, so we have $2^{64} \div 2^{13} = 2^{64-13=51}$ pages. Each page must have one page table entry. So we would need 2^{51} = 2,251,799,813,685,248 page table entries. This is too many to fit in a two-level paging system, so the Alpha memory management system uses three level paging.

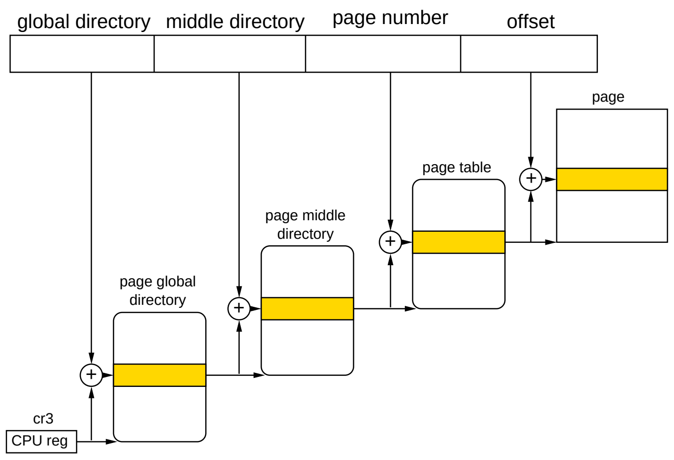

Figure 4: A three-level paging system.

Linux uses a three-level paging system: Since some architectures, such as the Compaq Alpha architecture, use three-level paging, to ensure portability, Linux implements a three-level paging system.

1.2 Types of Operating System

Types of OS There are four main types of operating system (according to Andrew Tanenbaum, *Modern Operating Systems*):

Monolithic OS is an OS where there is one level of privelege, and where all kernel functions execute in the same address space. Andrew described this organisation as “the big mess” which would be very hard to port to new architectures.

Layered Kernel is an organisation where the kernel is divided into different layers, where a special system call is required to communicate between each layer. Each layer may have a different level of *privelege*. However, there is some overhead in communicating between each layer, and dividing operating system functions into layers that minimise communication between the layers is hard.

Microkernel client-server is an organisation where as much of the OS function is done in "user space" servers rather than in the priveleged level of the kernel. The kernel mainly routes requests from client programs to these user-space servers. This should result in a much more portable OS with a very small kernel.

Virtual Machine is another organisation for operating systems; most famous example is IBM 390 mainframe. Now it runs Linux on hundreds or thousands of virtual machines with virtual hardware. If one virtual machine crashes, no problem to other virtual machines. The crashed virtual machine can simply be rebooted without affecting any others.

The main advantage is that the peak demands on all the servers can be averaged out. Normally, a smaller server must be made powerful enough to handle peak demand. Most of the time, this extra capacity is unused. With hundreds of virtual servers, some virtual servers can meet peak demands (and so get more processing power from the mainframe), while others that have less than average demand can use less CPU power from the mainframe. In this way, the total CPU power of the mainframe can be less than the total CPU power of many racks of smaller servers, (and require much less electrical power, less air-conditioning and cooling, and less physical space), but still meet all the peak demands placed on the individual virtual servers.

VMware provides software equivalent of a virtual machine. IBM 390 has hardware support.

Andy was wrong! Linux has a *monolithic* organisation, whereas Windows NT/2000 has a microkernel organisation. Andrew Tanenbaum argued with Linus Torvalds about the design of Linux for a long time. However, history has shown that Andy was wrong in some respects.

The Linux kernel is much smaller and simpler than the Windows kernel, and is far easier to understand. The Linux kernel divides the hardware control into dynamically loadable kernel modules. Linux runs on a huge number of hardware platforms; Windows has reduced the number of platforms it can run on to one.

2 Questions

2.1 Memory Management

1. What is *virtual memory*?

✐

2. Why do we need memory management?

✐

3. What is a *Memory Management Unit* (MMU)?

✐

4. Paging in a virtual memory system allows nearly all contents of RAM to be paged to and from the hard disk. However, there must be some pages that are never swapped to disk. List one example, and give reasons.

✐

5. For a hypothetical computer architecture, the following assumptions are made:

- we are using two-level paging;
- the virtual addresses are 64 bits in size;
- size of a page is 8 kB;
- size of both a page table entry, and also a directory table entry is 8 bytes;
- the size of the directory table and page table are as close to each other as possible,

calculate the size of the page directory and page table, in bytes. Note: they will need to be bigger than a single page.

✐

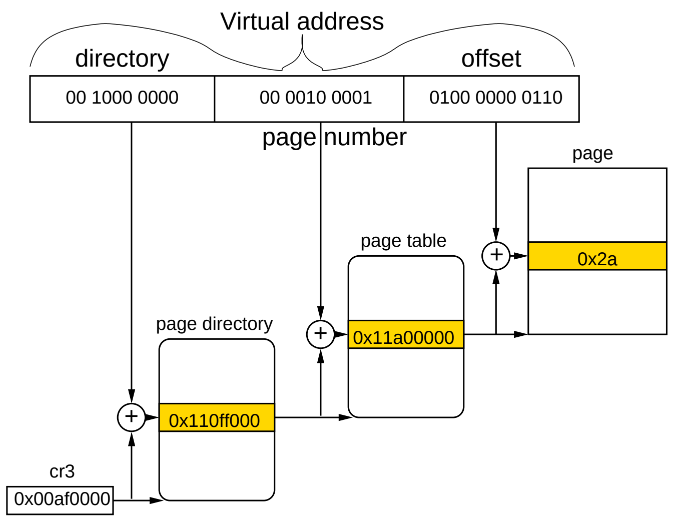

Figure 5: An example of an Intel two-level paging system.

6. Figure 5 on the previous page shows an example of an Intel two-level memory management system, with both page directory entries and page table entries occupying four bytes. Some values are shown. If this shows a snapshot of the system while a program accesses a character variable,

(a) What is the address of the variable as the program sees it?

✐ . . . .

(b) What is the address of the coloured entry in the directory table?

✐ . . . .

(c) What is the address of the coloured entry in the page table?

✐ . . . .

✐ **(d)** What is the physical address of the variable? . . .

✐ **(e)** What is the content of the variable? . . . .

2.2 Deadlock

1. What is *deadlock* in the context of a computer operating system?

✐

2. List four conditions for deadlock to occur in a computer's operating system.

✐

```
Process P      Process Q
...            ...
Get A          Get B
...            ...
Get B          Get A
...            ...
Release A      Release B
...            ...
Release B      Release A
...            ...
```

Figure 6: Two processes that can deadlock.

3. Two processes are shown in figure 6.

(a) Show some steps that each process can take so that deadlock must happen.

✐

(b) Explain one method that you could use to avoid any possibility of these two processes being deadlocked.

✐

4. A client-server application has at least two processes, each on a different computer. The client makes requests of the server, the server responds by transferring data over the network to the client. Generally, the client will initiate the requests; the server does not initiate any requests. However, both the client and the server perform two way communication over the network.

Describe how a client-server application can be deadlocked.