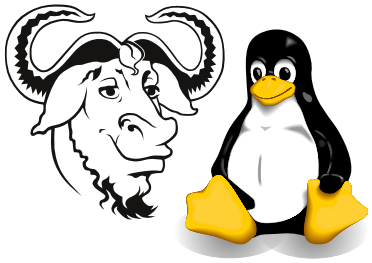

Overview of Lectures

Overview

Aim of subject: This subject aims at a practical understanding of operating systems supporting LPI and RHCE ceritfications rather than one oriented towards theory and operating sytem design.

Free software is a technical term defined by Richard Stallman, the founder of the *Free Software Foundation*. Stallman wrote emacs and the Gnu C compiler suit. Free software provides the following four freedoms:

- The freedom to run the program, for any purpose (freedom 0).
- The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbor (freedom 2).
- The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. (freedom 3). Access to the source code is a precondition for this.

Why use free software? Free software provides benefits to users of the software; if the company goes broke or decides not to support it anymore, then people can carry on using and developing the software. There is no vendor lock-in. if it doesn't work, no need to wait for someone else; you can fix it yourself. Good for infrastructure (such as operating systems!) Free software supports open protocols and open standards. The Internet is built on free software; that is why is is so successful. There is no "embrace and extend" of standard technologies.

Types of Operating System

What is the OS? The operating system is essentially the kernel that runs in a priveleged execution mode supported by the hardware of the CPU. This mode is called *supervisor mode*, or sometimes *kernel mdoe*. Application programs run in *user mode*, and CPU hardware prevents them from executing priveleged instructions to access some of the hardware. The kernel:

- manages hardware resources and shares them between applications; this is like a government.
- provides a standard set of *system calls* that allow programmer to interface to hardware at a higher level than by programming registers in hardware.
- These system calls wrapped in library functions.

What resources? The OS manages CPU, memory, files and disks, printing, network access, I/O devices,...

Multiuser and multitasking: OS must protect users from each other, and protect processes from each other. Processes have owners, and execute with the permission of the owner.

Types of OS There are two 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.

Microkernel or Layered Kernel is an organisation where as much of the OS function is done in "user space" rather than in the priveleged level of the kernel. This should result in a much more portable OS with a very small kernel.

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.

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

Processes

What is a process? A process is a program in execution. It has its own address space.

What is a thread? A *thread* is a lightweight process that shares its address space with other threads performing the same task.

A process has an owner The process executes with the permissions of its owner.

Process is born with `fork()` In Linux, a process is born by making a copy of its parent process wiht the simple `fork()` system call. So each process has a parent, and many processes have children. They form a family tree of processes.

Scheduler is an important component of an OS that determines which runnable process should run next. The aims of the scheduler include:

- maximising CPU usage
- maximising process completion
- minimising process execution time
- minimising waiting time for runable processes
- minimising response time

Process states Processes move through three main states, as in figure 1 on the following page. When a process changes state, this is called a *context switch*.

`vmstat` gives information about process states in Linux.

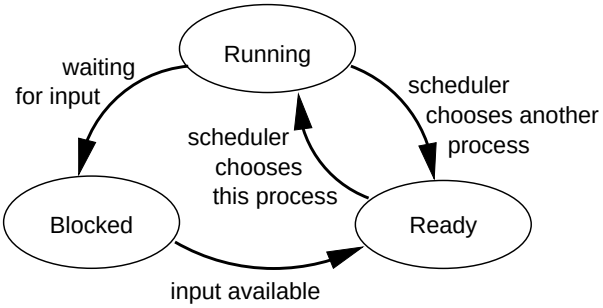

Figure 1: The states that a process may be in.

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 2. The MMU and CPU organisation fix the size of the pages, page tables and various other aspects of paging.

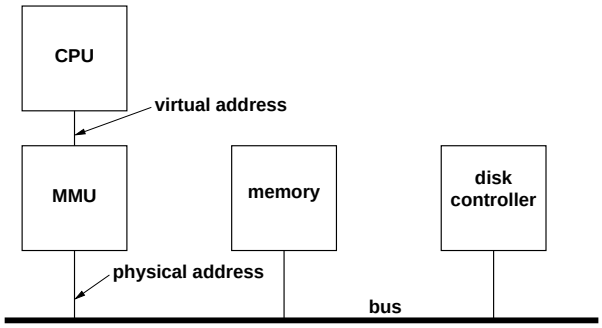

Figure 2: The memory management unit is shown here converting virtual addresses from the CPU to physical addresses.

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.

Why we need multilevel paging (single level paging won't do): Many people seemed unable to understand why a single level page table 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

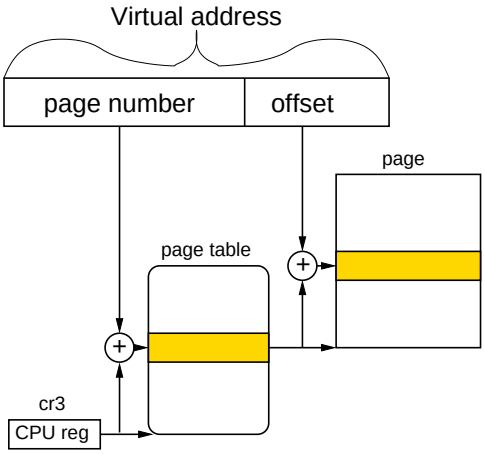

Figure 3: A single-level paging system. Virtual memory addresses are 32-bit. Pages are 4K each.

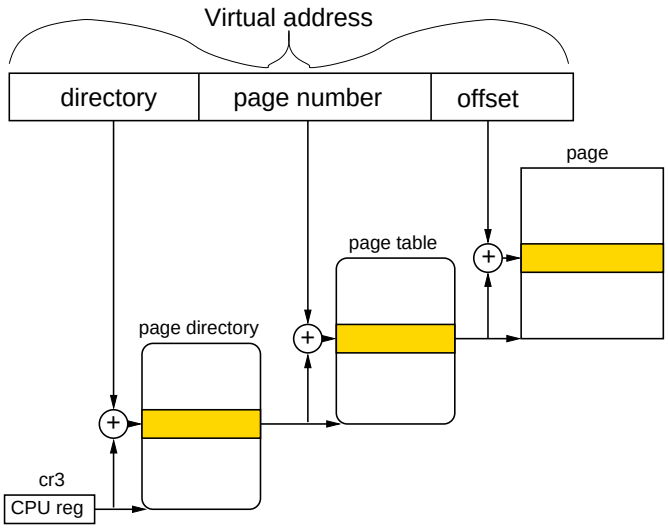

Figure 4: A multi-level paging system. Virtual memory addresses are 32-bit. Pages are 4K each. The page tables are themselves pages, also of 4K each. Since each page table entry is 4 bytes in size on the Intel platform, there are $1024 = 2^{10}$ entries in each of the page tables. So there are ten bits 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.

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 3 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}.