# Operating Systems

Inge Frick
inge@nada.kth.se



## What is an Operating System

When talking about an operating system, one usually means one of two things:

- A kernel that sits between the hardware and the user programs.
- A computer environment with a kernel and a lot of system programs like command line interpreter (shell), compilers, editors etc.
- In this lecture we will use the first meaning of OS



## What does the kernel do?

The kernel:

- Runs programs (handling processes)
- Handles the (virtual) memory
- Offers an API to handle the hardware. This:
- Handles I/O
- Handles file systems
- Handles networks
- Detects and handles errors and other interrupts
- Allocates resources
- Protects the security and stability of the system



## Process

A program running on the computer with resources is called a process.

A process has an owner, allocated memory, open files and other resources.

The kernel creates processes, setting up all data structures to handle them.

When a process I finished or killed prematurely, the kernel cleans up.

Switching from one process running to another is called a “context switch”. The kernel sets up a number of registers with information of the current process.

The memory available to a process contains three parts:

- The static part (the compiled program with all its (static) library routines).
- The dynamic part. The size of this part can vary during execution. A process can ask for dynamic memory. Here are also stored dynamic library routines.
- The stack. This memory changes in size to allow for recursive function calls.

# Multitasking

Modern computers run many programs in parallell.

On a computer with 1 CPU with 1 Core this is simulated by Time multiplexing: Each process is running in turn, switching so fast that they seem to run I parallell.

A process must often wait, e.g. waiting for I/O, a block from the hard disk, a character or a line of characters from the keyboard etc. When a process waits, another process can run.

- Primitive OS (and some Real Time systems) have Co-operative multiplexing: A running process decides when it can wait and let another process run.
- Modern OS use Preemptive multiplexing: Every time a process calls the OS (a syscall), the OS decides if some other waiting process shall run instead. In addition to this there is a timer which forces a process to call the OS after a short time. This guarantees that a process can't occupy the processor indefinetely.



# Threads

A process can be divided into two parts:

- The execution thread with the content of the general registers, the program counter and the stack.
- The environment with the memory, open files and other resources.

Many operating systems allow more than one executing thread in a process.

Using a thread for each task instead of a process for each task has both advantages and disadvantages:

- + It is faster to create a thread than a process. In Unix the difference is a factor of two, but in e.g. Windows the difference is much bigger.
- - Using the same memory requires careful coding, although this also allows easier communication between threads.

Threads can be implemented in two ways:

- As a package of routines running in a process where the OS only sees the process.
- The OS knows about each thread.



# Virtual memory

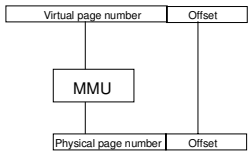
One of the resources a process has is memory.

On modern computers processes don't address memory directly: Each virtual memory address is translated to a physical memory by special hardware (the MMU). The way it works is the following:

The memory is divided into pages of the same size. A typical page size is 4K bytes. This means that a memory address can be divided into two parts: The page number and the offset. The virtual address has a virtual page number and the physical address has a physical page number. The offset points at where on the page the addressed object is. The offset is the same for virtual and physical addresses.



# Virtual memory cont.




The MMU uses a “page table” to translate from virtual page number to physical page number.

The page table is situated in the memory and is unique for each process.

The MMU has a register that contains the physical address of the page table for the currently running process.

To speed up the MMUs translation, most machines have a TLB, a cache memory that contains most recently used parts of the page table.

All virtual page numbers don't have a corresponding physical page number. Instead in the page table there is information that the corresponding page is situated on the hard disk and usually also where on the hard disk it is.

When the corresponding page is situated on the hard disk, the MMU throws an interrupt, telling the OS that a page fault has occurred. Up to this point the MMU has handled the translation but now the OS takes over. The process must wait while the page is loaded into memory.

# Virtual memory cont.

Using virtual memory has a number of consequences:

- To each process it looks like it has the whole (virtual) memory to itself.
- A process can use more virtual memory than there is physical memory on the machine. More important, the total memory used by all the processes on the machine can be much more than the physical memory.
- By letting each physical page only appear in the page table of at most one process, different processes can't read or write each others memory. This is an important safety feature. It is possible to intentionally have a physical page appear in the page tables of more than one process. This gives these processes an efficient way of communicating.
- The page table has flags for each page that can e.g. be used to mark a page as read only.



# File systems

To simplify handling of permanent memory (hard disk, diskettes, tapes etc.), the OS organizes data in files.

A simple Unix file system organizes data on a logical devise (e.g. a partition of a hard disk) and has two main parts:

- A collection of blocks containing data.
- A vector of inodes, each with information about one file.
- An inode contains various dates, a counter of hard links (see below), owner of the file and other meta data. It also uses thirteen pointers to point to the files data blocks. The first then pointers point to the first ten blocks of data. If this is not enough, the eleventh pointer (the single indirection pointer) points to a block of pointers pointing to blocks of data. If even this is not enough, the twelfth pointer (the double indirection pointer) points to a block of pointers that point to blocks of pointers that point to blocks of data. If the file is truly huge so this is not enough, the thirteenth pointer (the triple indirection pointer) points to a block of pointers that point to blocks of pointer that points to blocks of pointer that points to blocks of data. This is enough! One thing an inode does not contain is a file name. Names are assigned to files in another way:

Files are organized in a tree of files were the inner nodes are directory files and the leaves are data files. A directory file is a table with two columns. Each row in the directory has a name and an inode number (an index into the vector of inodes). This associates a name to a file. It is called a hard link. Each name is unique in the directory, but the inode number can appear with many names and in many directories. This way a file can have many names and it can appear in many directories. As inode numbers are only unique within one file system, hard links can only point into the same file system. To be able to point between file systems, symbolic (or soft) links are used. A symbolic link is a special little file that contains the string of names that defines a path through the directory tree to a file.



# Communicating with the kernel

- A user talks with the kernel through:
  - A command line interpreter (a shell)
  - A GUI
- Programs talk with the kernel through:
  - A set of library routines (an API) as the programmer sees it.
  - System calls (syscalls) as the kernel sees it.



# Writing a kernel

Usually done in C with small parts written in assembler. Nothing strange but very sensitive to bugs as the kernel should check for errors.

- Monolithic vs Microkernel
- Modules