Why use an Operating System?

- Provides **a set of services** to system users (collection of service programs)
- **Shield** between the user and the hardware
- **Resource manager:**
  - ➝ CPU(s)
  - ➝ memory and I/O devices
- **A control program**
  - ➝ Controls execution of programs to protect information against accidental or unauthorized access

Operating System Definition

- No universally accepted definition
- “Everything a vendor ships when you order an operating system” is good approximation
  - ➝ But varies wildly
- “The one program running at all times on the computer” is the **kernel**. Everything else is either a system program (ships with the operating system) or an application program

Interrupts

- The machine has two modes, user mode and kernel mode.
- The machine has a status register (PSW - processor status Word) that determines the machine’s priority level and mode (user/kernel mode)
- Then an interrupt occurs, the machine’s PSW and PC are saved on kernel stack and a new PSW with the kernel mode bit set is loaded from the interrupt vector area.
- At execution of the return-from-interrupt instruction, PSW and PC are restored from kernel stack.
- Certain instructions are privileged and can only be carried out in kernel mode - for example load-psw.

Definition of process

A process is a program in execution.

A program is passive while a process is active.

A process consists of:

- Program code.
- Data area.
- Stack.
- General registers.
- Program counter.

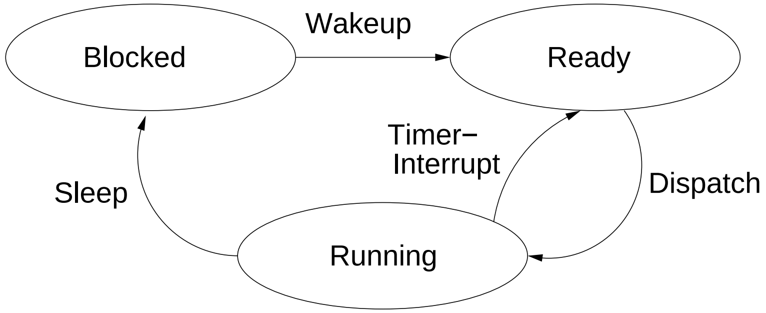
Concurrent processes

Why do we want concurrent processes?

- Sharing of physical resources (CPU, disks).
- Convenience. It may be good to be able to run several programs at the same time.
- Increased speed of calculation.
  1. Decreased waiting times through more efficient use of the processor.
  2. On a multiprocessor a program may be divided in several processes that execute concurrently.

Most of these reasons are valid also for a single user system.

Process State Diagram



Running:	The executing process
Ready:	Processes ready to run
Blocked:	Processes waiting for other resource than CPU

Process Control Block (PCB)

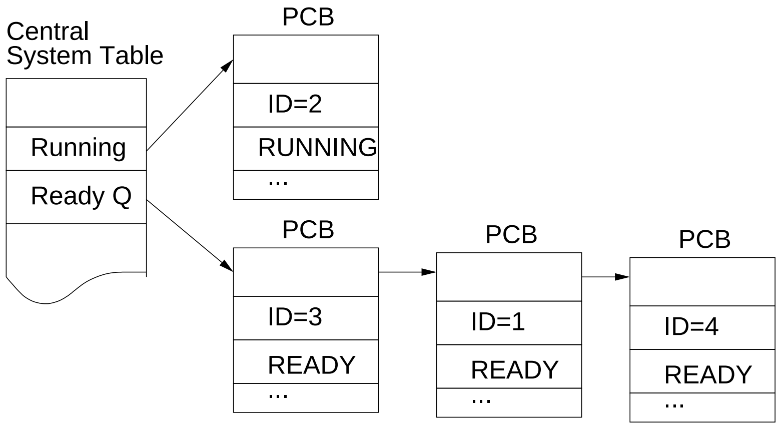
Data structure that contains information about the process.

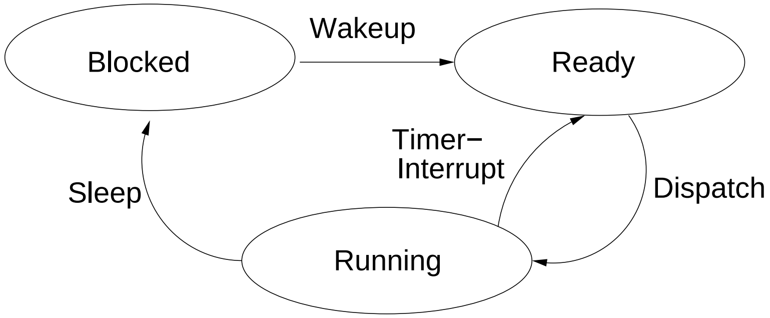
A PCB contains among other things:

- Pointer to the next PCB.
- Process state.
- Process identifier.
- Scheduling information. e.g. process priority.
- Memory-management information.
- Pointers to child and parent processes.
- Save area for processor registers.

The PCB is the data structure that defines the process for the operating system.

Process Queues





State transitions

The assignment of the processor to the first process in the ready queue is performed by a system entity called the *dispatcher*.

The dispatcher performs:

- Save the state (registers) of the interrupted process in it's PCB.
- Fetch the state for the next process to run from it's PCB.
- Activate the next process.

This state change can be described as: dispatch(process-name): ready -> running.

The dispatcher is often called by other operating system subroutines as:

- sleep(process_name): running -> blocked.
- wakeup(process_name): blocked -> ready.

The only state transition that is initiated by the process itself is block. The other transitions are initiated by events outside the process.

Unix sleep and wakeup routines

- Unix/Linux processes that execute user code executes in *user mode*.
- Preemptive scheduling is used in *user mode*.
- A process that has made a system call - and is executing operating system code - executes in *kernel mode.*
- Older Unix systems used non-preemptive scheduling in kernel mode but nowadays preemptive scheduling is used also in kernel mode.
- A process that needs to wait (for example for data from a disk memory) releases the processor by calling the subroutine *sleep*.
- When the awaited event occurs the process is woken up by the interrupt handler calling the subroutine *wakeup*.
- Wakeup can also be called by another process executing in kernel mode. Wakeup changes the state for the awoken process from *sleeping* to *ready*.

Operations on processes

System calls for process management:

- Create process.
- Kill process.
- Start execution of a program.
- Create a communication channel to another process.