Processes and interrupts

In general, we will have a number of runnable processes in our uniprocessor machine, and one

process running at a given instant of time. When will the waiting processes get a chance to run?

There are two reasons for suspending the execution of the running process: either the time-slice

allocated to it has been exhausted, or it cannot proceed any further before some event happens.

For instance, the process must wait for input data to become available, or for a signal from

another process or the operating system, or it has to complete an output operation first, etc.

The programmer does not have to bother about this.

At a given point in the program, where it needs to have more input data, the programmer simply writes a statement such as:

read(file,buffer,n);.

The compiler will translate this into a call to a library function, which in

turn will make a system call, (or service request), which will transfer control to the kernel.

Our process becomes suspended for the time the kernel needs to process this system

call. In the case of a read operation on a file, the kernel will set this into motion, by emitting

the necessary orders to the disk controller. As the disk controller will need time to execute this

order, the kernel will decide to block the execution of the process which was running and

which made the system call. This blocked process will be put in the queue of waiting processes,

and it will become runnable again later, when the disk controller will have notified the kernel

--by sending a hardware interrupt-- that the I/O operation has been completed.

The kernel makes use of the scheduler to find, from the queue of runnable processes the one that

should now be run.

The kernel will then make a context switch and this will start our suspended process running.