Tutorial: Process Control Blocks — Solutions

1. What is a *process control block* (PCB)?

A process control block is a *data structure* used by an operating system to manage processes.

2. Is a PCB only found in a Linux or POSIX OS or is it found in all operating systems?

As I mentioned in the notes, process control blocks are found in all modern operating systems. The structure of PCBs will be different on each operating system though.

3. Give *ONE* other name for a PCB.

It is sometimes called a *process descriptor*, or a *task descriptor*. In the 2.6 source code of Linux, the data type is a structure called `task_struct`

4. Give *ONE* name for the data structure that holds the PCBs.

The *process table*. The PCBs may be put into a number of lists at the same time. The process table describes the system by which the operating system finds a PCB by its process ID.

5. Briefly define the term *program counter*.

The *program counter* is a CPU register that holds the address of the next instruction that will be fetched. As the CPU fetches each instruction, the PC is automatically incremented to point to the next instruction.
You might like to play with the simple Macromedia Flash model of a simple computer consisting of a CPU and RAM connected by a 4-bit data bus, 3-bit address bus, and 2-line control bus. You can find it at `http://ictlab.tyict.vtc.edu.hk/ossi/lectures/processes/fetching.swf`. The RAM contains three instructions, each occupies two 4-bit memory locations.

6. What data is held in the *program counter*?

The address of the next machine instruction that will be fetched and executed.

Process Control Blocks: In figure 1 on the following page we see three processes, P1, P2 and P3. There are seven moments in time where the main values of the PCBs of the three processes are shown. The time intervals are not equal; they are just where the scheduler changed the process. The size of all I/O instructions is *three bytes*. In each PCB I have shown the program counter value, and the process state. The process state can be running (i.e., executing on the CPU), ready (i.e., ready to run), and sleeping (i.e., blocked). There are two queues shown: a wait queue for processes that are blocked waiting for I/O, and a ready queue, for processes that are in the ready to run state.

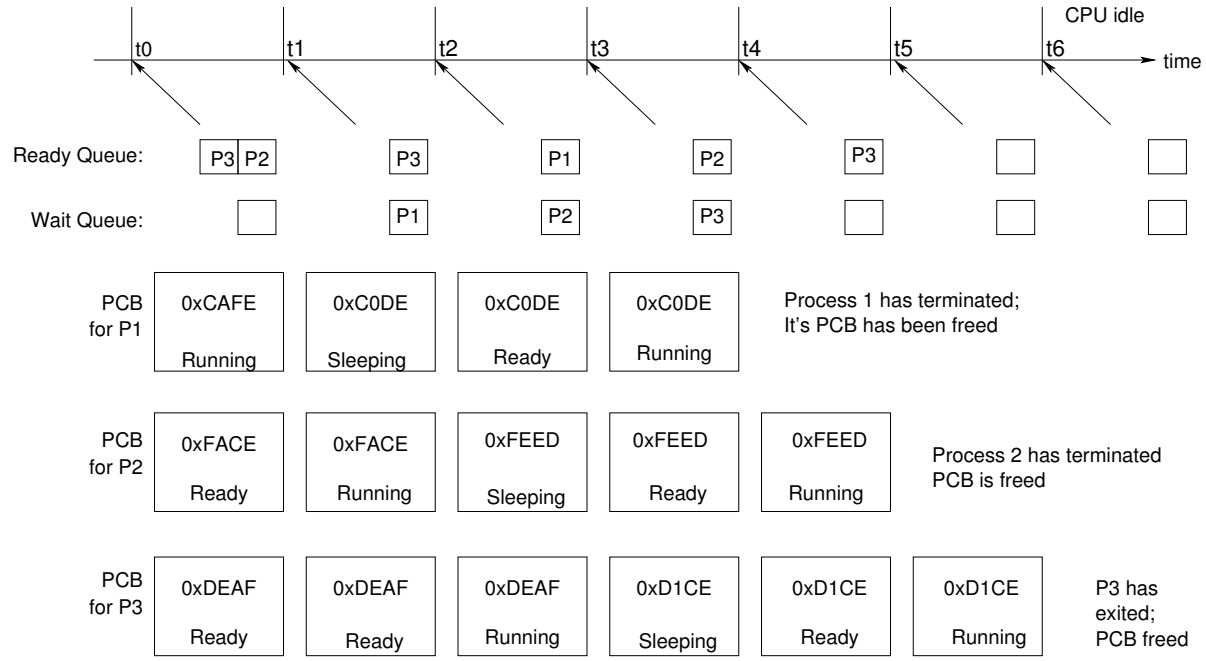

Figure 1: An example of three processes executing, showing their process control blocks.

1. State the technical term for what the scheduler does at each of the times t_0, t_1, t_2, t_3, t_4, t_5 and t_6.

Context switch.

2. Does process P1 perform I/O? If so, at what address is the I/O instruction?

Yes, you can see that it does, because the state of that process is sleeping after t_1, and you can see that it appears in the *wait queue*, where it waits for the input or output to finish.
We are told that the I/O instructions are *three bytes long* in *this particular example*, as specified in the question. At t_1, the CPU has already executed the I/O instruction. This is what caused the OS to put the process onto the wait queue, since I/O devices are typically much slower than the rate at which the CPU can execute instructions. The OS put the process into the sleeping state when the program counter was pointing to the *next instruction to be fetched* that comes immediately *after* the I/O instruction.
So we know that the address of the instruction after the I/O instruction is at address C0DE_{16}. The size of the I/O instruction is three bytes, i.e., it is three address locations before C0DE_{16}, so its address is $\text{C0DE}_{16} - 3 = \text{C0DB}_{16}$.

3. Does process P2 perform I/O? If so, at what address is the I/O instruction?

Yes, we can see that P2 moves to the sleeping state at time t_2, where we see that it is also on the wait queue.
Using the same reasoning as before, we can see that the address of the I/O instruction is $\text{FEED}_{16} - 3 = \text{FEEA}_{16}$.

4. Does process P3 perform I/O? If so, at what address is the I/O instruction?

Yes, P3 is blocked for I/O at time t_3. The address of the I/O instruction is $\text{D1CE}_{16} - 3 = \text{D1CB}_{16}$.

5. List at least *FIVE* other kinds of data that will be stored in a PCB that is not shown in figure 1 on the previous page.

If you look at the definition of the data type `task_struct` in the header file `/usr/src/linux-2.4.22-1.2188.nptl/include/linux/sched.h` and also see the definition of `thread_struct` in `/usr/src/linux-2.4.22-1.2188.nptl/include/asm/processor.h`, you will see that the PCB contains a great deal of information about the process, including:

- The process ID
- the process state (for example, whether it is executing, ready to run, or sleeping)
- program counter (stored in `thread_struct`)
- the scheduling priority
- information about the user and groups associated with this process
- pointer to the parent process PCB
- pointer to a list of child process PCBs
- pointers to the process's data and machine code in memory
- pointers to open files and other resources held by the process