Philip Dumaresq

40082638

COMP 346 - Operating Systems

Goswami Dhrubajyoti

Assignment 1

Question 1

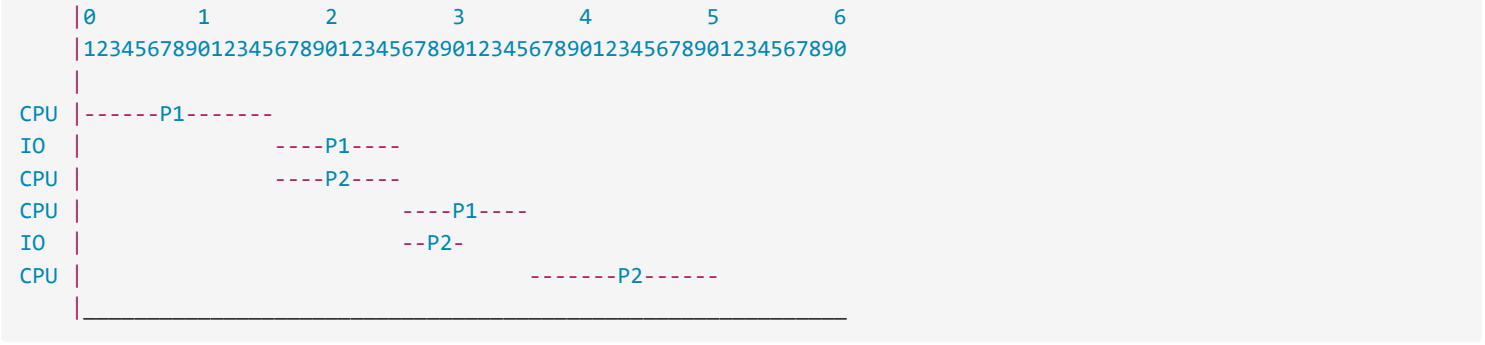
Consider a computer system with a single-core processor. There are two processes to run in the system: P1 and P2.

- **Process P1 has a life cycle as follows: CPU burst time of 15 units, followed by I/O burst time of minimum 10 units, followed by CPU burst time of 10 units.**
- **Process P2 has the following life cycle: CPU burst time of 10 units, followed by I/O burst time of minimum 5 units, followed by CPU burst time of 15 units.**

Now answer the following questions:

a) Considering a single programmed operating system, what is the minimal total time required to complete executions of the two processes? You should explain your answer with a diagram.

> The minimum total time required to complete the execution of the two processes will be 50 units



b) Now considering a multi programmed operating system, what is the minimal total time required to complete executions of the two processes? You should explain your answer with a diagram.

> The minimum total time required to complete the execution of the two processes will be 35 units

```
    |0        1         2         3         4         5         6
    |12345678901234567890123456789012345678901234567890123456789012345678901234567890
    |
CPU |------P1-------
CPU |----P2----
IO  |          --P2-
IO  |               ----P1----
CPU |               -------P2------
CPU |                         ----P1----
    |_____________________________________________________________
```

c) Throughput is defined as the number of processes (tasks) completed per unit time. Following this definition, calculate the throughputs for parts a) and b) above. How does multiprogramming affect throughput? Explain your answer.

If throughput (T) is the number of process completed (p) per time unit (t), then it can be written as $T = \frac{p}{t}$

> a) $\frac{6}{50}$
> b) $\frac{6}{35}$

Question 2

Suppose that a multiprogrammed system has a load of N processes with individual execution times of t1, t2, ..., tN. Answer the following questions:

a) How would it be possible that the time to complete the N processes could be as small as: maximum(t1, t2, ...,tN)?

> Running these process concurrently would make it so that the total time to complete all N processes would be the time taken to complete the longest process.

b) How would it be possible that the total execution time, T > t1+t2+...+tN? In other words, what would cause the total execution time to exceed the sum of individual process execution times?

> All processess need to be spawned from a main thread, and launching each of those processes is going to have overhead. Given the overhead of launching a new process, the total time will be greater than the sum of it's parts.

Question 3

Which of the following instructions should be privileged? Explain your answer.

(i) Read the system clock;

> You're reading a time that you have no control over that's pretty arbitrary, there's no reason for this to need to be priveleged. It's a safe operation.

(ii) clear memory;

> This should be priveleged because you don't want to allow memory to be cleared wherever and whenever. If it wasn't priveleged you could risk programs clearning memory for important tasks.

(iii) turn off interrupts;

> A process could use up the entire CPU if it could turn off interupts, so it should be priveleged.

(iv) switch from user to monitor mode;

> This is how you invoke system calls, so it needs to be unpriveledged.

(v) issue a trap instruction;

> Traps are software invoked, so they'd be able to run in user-mode and can be overrided by hardware interrupts

(vi) copy from one register to another.

> Priveldged. This needs to occur in kernel mode since you can't directly interact with hardware in user mode anyways.

Question 4

Answer the following questions:

a) Why need system calls? Explain from the following two perspectives: user's convenience and system protection.

User's convenience:

> As a developer, there's certain things you don't want to need to do yourself - for obvious reasons. Using syscalls you can interact at a lower level safely.

System Protection

> There are obviously things that you don't want to allow code to be able to do entirely unauthorized.

b) Explain how a system call utilizes interrupts, user and monitor modes, and privileged instructions. What is achieved in return? Explain your understanding.

Since user-written code can't interact directly with hardware, you need to use system calls to achieve this at a lower level. System calls use interupts, monitor monitor modes and privelged instructions to accomplish this. To perform a low level

operation you first need to be in monitor mode, which requires you to run as priveledged instruction. An interrupt is sent to make this call to return it to the executing code quickly so that it doesn't need to wait.

Question 5

Are interrupts always advantageous over polling? Explain your understanding with suitable example(s).

Interrupts are not always advantageous over polling. In a case where you're writing code for an embedded system like an arduino chip or a video game cartridge and you don't have thread support, then polling could be advantageous since you're working entirely synchronously and you'd know when to expect everything to happen.

Question 6 - TODO

Consider a preemptive operating system where processes have priorities and a running process gets preempted (i.e., forced to leave the CPU) as soon as a higher priority process is ready to run. The life cycle of a process, other than the very first process, begins with a "spawn" by another process and ends with either a regular "exit" by the process or a "terminate (process_id)" command by another process of equal or higher priority.

Each process is assigned an initial priority at spawntime and this priority remains unchanged during the entire life cycle. There are system resources, both hardware and software, for which a process can block if the resourceis not free. A process can also be suspended by another process of equal or higher priority through the call "suspend (process_id)". A suspended process is resumed by the call "resume (process_id)". Note that a process can be in any state (i.e., running, blocked or ready) when suspended.

Processes communicate with one another via "send" and "receive" message passing primitives. "Receive" is always blocking, i.e., the calling process blocks if the message is not available. "Send" is always non-blocking. Illustrate the complete life-cycle of a process with the help of a process state transition diagram.

Question 7

When there is a context switch from one process to another, the OS kernel invokes the function ContextSwitch which saves the context of the currently executing process into its PCB and then inserts the process to an appropriate queue (i.e., ready queue or a blocked queue). It is necessary that ContextSwitch is atomic (i.e., unbreakable: either done or not-done; nothing in between). Explain the following:

a) Why must ContextSwitch be atomic?

Because if it can get interrupted then the process that it's trying to save could get corrupted and stop working when it's attempted to be continued

b) Give an example scenario of what can go wrong if ContextSwitch is not atomic.

Say process p1 is running, and process p2 needs to run at a higher priority. ContectSwitch starts trying to save the context for p1. Before it can finish, process p3 needs to run at higher priority than p2. While ContextSwitch is trying to save p1, ContextSwitch gets invoked for process p2 and the contexts for each get weaved together and corrupt the processes.

c) How can it be made atomic in practice?

To make ContextSwitch atomic it would need to block other processess from executing until it was completed by forcing a lock until it's finished.