

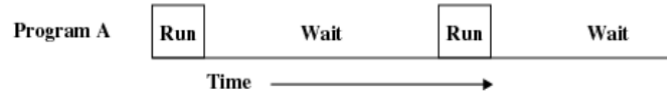
Assignment1: Questions 1 - 3

Stuart Mashaal

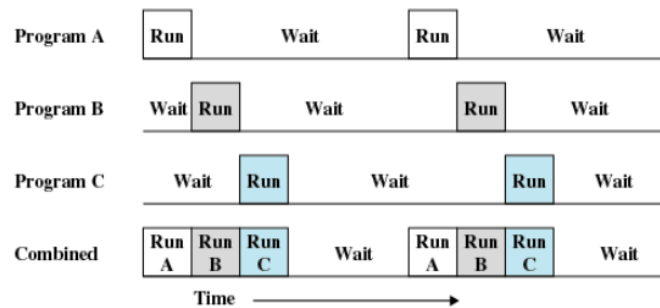
Sunday, October 1, 2017

Question 1

Uniprogramming is a computing paradigm wherein the computer can only manage and run a single process at once. This means that if there are three process to run and each one performs a large IO operation, the computer will run them each separately and sequentially with the CPU idle during each of the large IO operations.



Multiprogramming is a computing paradigm wherein the computer can manage multiple processes at once, whether or not it can actually execute the instructions of multiple processes in parallel. This means that if there are three processes to run and each one performs a large IO operation, a single-core CPU would not idle during the large IO operations. Instead, while the first process is waiting for IO, it will run another process and while the second process is waiting on IO it will run another (and so on, as necessary).



Time-sharing is the ability of a computer system to accommodate multiple users simultaneously. A time-sharing system is one that uses multiprogramming to allow multiple users to interact with the system at the same time. While all but one of the users is idle, the system can record (for example) the key-presses of one user. Time-sharing systems are really just multiprogramming systems in which one of the processes that the system is constantly switching between is the reading in of multiple users' input.

Question 2

As stated in my answer to Question 1, a uniprogramming system is a system that can only run and manage a single process at once. So, once a job starts, no matter how many parts of that job don't even use the CPU, the CPU will not execute the instructions of any other job until that job finishes. This means that jobs A, B, and C will run one after the other. Therefore, the time taken to finish the three jobs on a uniprogramming system is given by

$$3 \cdot (2\text{ms} + 10\text{ms} + 4\text{ms}) = 48\text{ms}$$

Question 3

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <fcntl.h>
4
5  int main()
6  {
7      close(1); // close stdout
8      open("redirect.txt", O_WRONLY, O_CREAT); // open redirect.txt
9      printf("A simple program output.");
10     return 0;
11 }
```

Calling `close(1)` closes the connection to `stdout` and removes file descriptor 1 from the process's list of file descriptors. Then, calling `open("redirect.txt", O_WRONLY, O_CREAT)` opens a connection to the file `redirect.txt` and puts the file descriptor to it in the slot where we just removed the file descriptor for `stdout`. The flags `O_WRONLY` and `O_CREAT` mean that the connection to `redirect.txt` is 'write-only' and that `redirect.txt` will be created if it doesn't already exist.