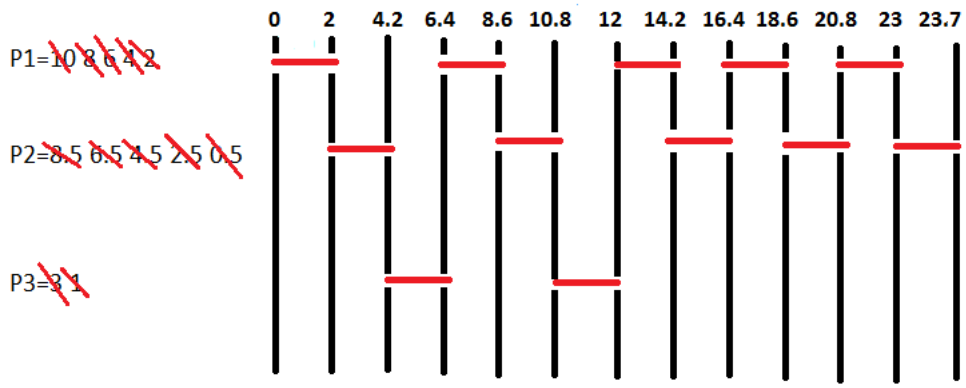
Rawan Alzowaid
CSCI-3753
Problem set 1

Q1:

```
P1          | P1          | P1     | P1     | P1
  | P2          | P2          | P2     | P2     | P2
      | P3          | P3
```

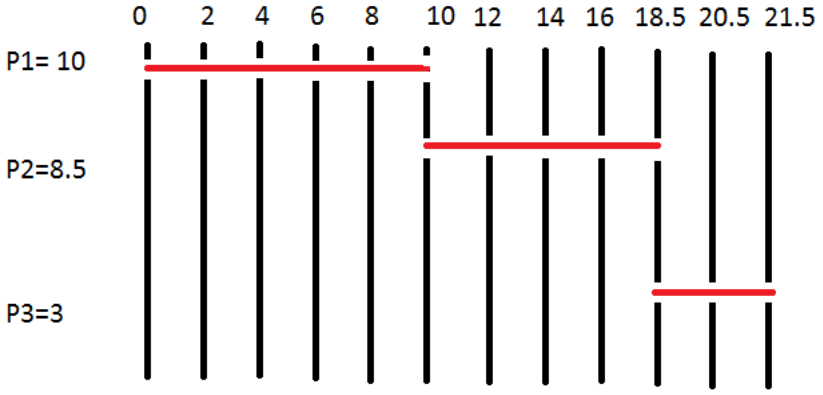


P1		P2	P3	P1	P2	P3	P1	P2	P1	P2	P1	P2
0	2	4.2	6.4	8.6	10.8	12	14.2	16.4	18.6	20.8	23	23.7

- Total Execution Time: 23.7 seconds
- P3 Finishes: 12 seconds
- P1 Finishes: 23 second
- P2 Finishes: 23.7 seconds
- Context Switches: 11 times
- Total Time Spent on Context Switches: 2.2 seconds
- Context Switch Overhead (Percent): 10%

Q2 :

```
P1
  | P2
      | P3
```



P1		P2	P3
0	10	18.5	21.5

Rawan Alzowaid
CSCI-3753
Problem set 1

- Total Execution Time: 21.5
- P1 Finishes: 10 seconds
- P2 Finishes: 18.5 seconds
- P3 Finishes: 21.5 seconds

Batch Mode Multiprogrammed finished first in this instance. The reverse would be true if a program failed to finish executing due to an error or if the CPU is blocked because of an I/O bound job in which case other programs would be starved of CPU time and unable to make any progress.

Q3:

- A type is used as a software interrupt in system calls and it gives control from the APP to the kernel.
- A fault is something that is used for an error that can occur and can be fixed.
- An interrupt is used to send a signal that an I/O task is complete
- Abort is used for problems or errors with no solution (cannot be fixed)

A software interrupt handles the system calls in an APP and it causes the running APP to stop executing and then goes to the kernel mode and gives it control to execute the system call that we want. After we finish executing the system call, it goes back to the user mode and the App gets back in control. Whereas, a hardware interrupt shows the application that an I/O task has been completed. And then it gives the APP the permission to skip its waiting on the OS to return. The hardware interrupt makes the CPU stop executing and runs the OS handler after it gets done with that it can go back to whatever it was executing.

Q4:

The trap table contains the memory location (pointer) to the trap handler for the system call in question. The trap handler is then executed and control is given back to the application that called the system call.

Q5:

Overlapping I/O with CPU processing is very advantageous because it allows the CPU to do something else useful rather than idling or polling the I/O device to check if the I/O task has completed, saving a lot of CPU cycles. The two ways to overlap I/O with CPU execution are direct I/O with interrupts and DMA (Direct Memory Access) with interrupts. The use of direct I/O with interrupts allows the transfer of data to commence without the CPU continually polling to check if the I/O has completed, the CPU receives notification of completion through an interrupt. The use of DMA with interrupts allow the transferring of data from an I/O device to memory without interrupting the CPU multiple times. Instead, the CPU is notified once the data transfer fully completes. This substantially improves the performance of large I/O transfers because the CPU is not constantly being interrupted.

Rawan Alzowaid
CSCI-3753
Problem set 1

Q6:

System Call Steps:

1. Application calls the system call write()
2. Trap/"software interrupt" immediately follows
3. Mode bit is set to kernel bit.
4. Kernel begins to execute the system write driver
5. The write driver obtains the status from the device controller
6. Add to request queue
7. Give control back to application

If device is not busy:

1. Device handler is executed
2. Pull data from device controller
3. Return data to application.

If device is busy:

1. If device is busy, interrupt occurs when it is available
2. Interrupt handler is executed, find out which device raised the interrupt flag.
3. Device handler is executed
4. Pull data from device controller
5. Return data to application.