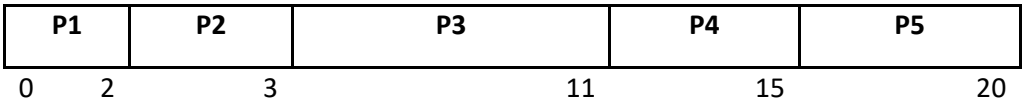
5.9 Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?

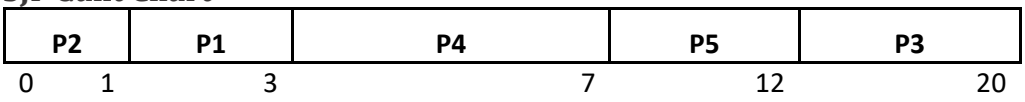
Answer: The scheduler needs to distinguish between I/O-bound and CPU-bound processes because of their CPU burst times. An I/O-bound process has many short CPU bursts while a CPU-bound process may have a few long term CPU bursts. This may cause many I/O-bound processes to wait for the CPU after finishing their I/O while a CPU-bound process holds the CPU. Once the CPU-bound process releases the CPU to do some I/O, the I/O-bound processes may be allocated the CPU and finish executing before the CPU-bound process is done with the I/O. This type of back and forth execution doesn't utilize the CPU and I/O to the fullest, this is one of the reasons why it is important for the scheduler to distinguish between I/O-bound and CPU-bound processes.

5.12 Consider the following set of processes, with the length of the CPU burst time given in milliseconds:

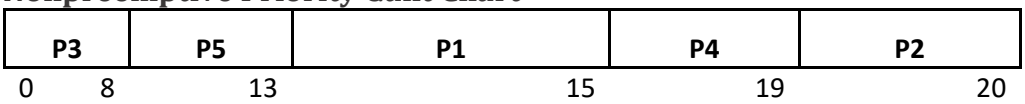
FCFS Gant Chart



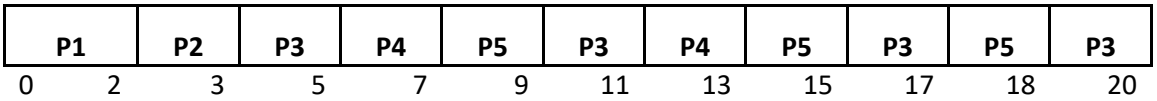
SJF Gant Chart



Nonpreemptive Priority Gant Chart



RR Gant Chart



Turnaround, Waiting, and AVG Waiting Times

Process	Criteria	P1	P2	P3	P4	P5	AVG Wait Time
FCFS	**Wait time**	0	2	3	11	15	6.2
FCFS	**Throughput**	2	3	11	15	20	6.2
SJF	**Wait time**	1	0	12	3	7	4.6
SJF	**Throughput**	3	1	20	7	12	4.6
Nonpreemptive Priority	**Wait time**	13	19	0	15	8	11
Nonpreemptive Priority	**Throughput**	15	20	8	19	13	11
RR	**Wait time**	0	2	12	9	13	7.2
RR	**Throughput**	2	3	20	13	18	7.2

5.14 Consider a variant of the RR scheduling algorithm where the entries in the ready queue are pointers to the PCBs.

a) What would be the effect of putting two pointers to the same process in the ready queue?
Answer: It would get double the execution time. Having two pointers to the same process would allow the double the CPU time so it would finish twice as fast.

b) What would be two major advantages and disadvantages of this scheme? Answer:

Advantages:

- The process would execute twice as fast by gaining double the CPU time.
- The Average wait time would decrease for all processes.

Disadvantages:

- Two CPU's on a multiprocessor machine can be allocated the two process pointers and could cause a CPU lock.
- Moving the resources of that process from one CPU to the other would be time costly every time they are allocated that process.

c) How would you modify the basic RR algorithm to achieve the same effect without the duplicate pointers?
Answer: You could give make the time quantum 50% of the process execution time or double the time quantum from what it currently is.

6.9 The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, P0 and P1, share the following variables. Answer:

1) Mutual Exclusion is achieved by the flag and turn variables. Even if both process i and J set their flags to true, only the process who's turn it is will succeed in going into the critical section. This is made sure by the if statement in the while loop.

2) Progress is achieved by the flag and turn variables as well. If a process wishes to execute their critical section, it just sets its flag to true. However, if turn is set to the other process, it will not execute until the other process is done with its critical section and sets turn to the process requesting to execute its critical section.

3) Bounded Waiting is preserved by the turn variable. If both processes want to execute their critical section, then they both set their flag to true but only one can proceed due to the turn variable being set to only one of those processes. While one process executes its critical section, the other process waits for its turn, once the process executing its critical section is done then it sets the turn variable to the other process. This ensures that the process hands over the critical section to the other process once it is done with its own critical section execution.

6.11 What is the meaning of the term busy waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.

Answer: Busy waiting is a means of a process waiting on anther process. The process that is busy waiting is in a continues loop consuming CPU execution cycles.

Another kind of waiting is to put the process in a waiting queue. In other words, block the process (put it to sleep) until it is awoken later on.

No, Busy waiting cannot be avoided because it is sometimes more beneficial to have a process in a spinlock on multiprocessor systems. This benefit is that a process that only has to wait for a short period of time could wait in a spinlock and avoid a costly context switch.

6.21 Write an algorithm for a bounded-buffer monitor in which the buffers (portions) are embedded within the monitor itself.

Answer:

```
public class BoundedBufferMonitor<E>
{
   private final int BUFFER_SIZE = 20;
   private E buffer[] = (E[]) new Object[BUFFER_SIZE];

   private BoundedBuffer producer;
   private BoundedBuffer consumer;
   private int in, out, count;

   public BoundedBufferMonitor()
   {
      in = 0;
      out = 0;
      count = 0;
   }

   public void insert(E e)
   {
      //if count is full, then wait
      if (count == BUFFER_SIZE)
         producer.wait();

      buffer[in] = e;
      ++count;

      in = (in + 1) % BUFFER_SIZE;
```

```
        consumer.signal();
    }

    public E remove(E e)
    {
        //if count has no items then wait
        if (count == 0)
            consumer.wait();

        e = buffer[out];
        --count;

        out = (out + 1) % BUFFER_SIZE;
        producer.signal();
    }
}
```