Problem 1. (30 Points) Pre-condition: Assume that the system will receive random hardware interrupts due to unknown hardware malfunction. The random interrupts are caught at the following tick-marks:

{ 3, 10, 13, 21, 24, 32, 35, 40, 47, 49, 54, 58 }.

Assume that PA and PB are two programs. Assume that when PA is executed it takes 8 ticks to complete its task, and it forks new processes executing PB at tick-marks 2 and 6, and it also requests an I/O at tick-mark 4. Assume that when PB is executed it takes 4 ticks to complete its task, and it requests an I/O at tick-mark 2. Assume that 3 processes { P1, P2, P3 } had arrived before t=0; and P1 and P2 are executing PA while P3 are executing PB. Assume that each I/O request needs 4 ticks to be handled and I/O requests are scheduled in a FCFS manner. Draw a Gantt chart illustrating the scheduling if a FCFS method is used. Assume that the executing process will goes back to Ready Queue if it has caught an interrupt or it goes to I/O Queue if it makes an I/O request. Mark on the Gantt chart when a process is arrived or finished. Special note: if both an interrupt and an I/O request occurred at the same tick mark, then the I/O is bounded to be happened first. (Explain the reason why the previous sentence is correct for extra credits.)

Problem 2. (10 Points) Pre-condition: In order to prevent the back-and-forth track seeking, the read/write header of the I/O device will only move forward (from low to high), and the I/O queue will be sorted according to the track# and the block#. The queue will be implemented as a circular linked-list. When the queue is reached to its end, it will be returned to the beginning of the list (*hdr) directly, and at the same time the read/write header will be fast-backward to the track #0 within 1 μs (microsecond). Conceptually, the read/write header (marked as @) will be placed within the linked-list. For example: if the read/writer header is located at track #30 currently, then @ will be between I/O request for track #27 and track #34, since it only move forward, so the next I/O to be handled will be for track #34 (as shown below).

```plaintext
@  
^  
|   
|_____________________________________v
```

Let’s just focus on the seeking time only. Assume that the read/write header is currently located at track #123, and a number of I/O requests can be listed by their associated track # in their calling sequence as: { #183, #88, #175, #45 }. Assume that the step motor is moving in a speed of 1 μs per step. How long it takes to handle all these four I/O’s?
**Problem 3.** (10 Points) Assume that there are three processes had arrived before $t=0$ in the order of {P1, P2, P3}. Assume that each process was divided into a sequence of CPU bursts by random interrupts and the sequence of CPU bursts for given processes are specified as:

P1: {3, 2, 2}, P2: {1, 4}, and P3: {1, 1, 5}.

Assume that we are calculating the ratio between the remaining time and the original burst time and the one with the highest ratio will be scheduled to execute. Draw a Gantt chart illustrating this scheduling.

**Problem 4.** (10 Points) Assume that there is only one processes P1 had arrived before $t=0$. Assume that this process will forks a new child process after half-tick of execution, the child process is going to execute the same program as its parent. So, an infinite number of processes will be created in this manner. Assume that each process can divided into two CPU bursts {1, 1} by a timely interrupt. Let’s compute the throughput every period of 4 ticks. As processes come and went, can we get a steady value of throughput if a FCFS scheduling is used? Assume that the executing process will goes back to Ready Queue after it has caught an interrupt.