Homework2 Report

Name: Ali Jahanshahi, Mohsen Karimi
Date: February 19, 2018

Question 1

Concurrency means two or some tasks need to happen independently of each other. In a scenario consisting two tasks A and B, as an example, task A starts running, and then B starts before A is finished. Concurrency can be achieved in various different ways. Running tasks on multiple CPUs at the same time, which is also called parallelism is one of them. However, using multiple CPUs is not the only way. Another way, which is used in single core processors, is achieved by task switching. In task switching, task A works up to a specific amount of time, then the CPU which is allotted to task A stops and switches to task B. CPU works on task B for a while, then switches back to task A. In this scenario, assuming small enough time slices, running tasks may appear to the user that both tasks A and B are running in parallel. However, they're actually running sequentially in background.

Question 2

a) Points are generated every 1000μs and the time to service each point is 100.01μs. Therefore, the user application has enough time to collect them all and loss ratio is zero

b) In this case, the points are generated every 10μs. Since time to service for each point is 100.01μs, there would be some losses. The loss ratio is as follows:

$$loss = \frac{100.01 - 10}{100.01} \approx 90\% \tag{1}$$

c) Points are generated every 10μs. Assuming the system call does not handle the incoming data points during the time of copying buffer from kernel memory to user memory which is equal to $1000 * 10ns = 10\mu s$, one point will be lost in every 1001 points. So the loss ratio is approximately 0.01%.

d) The implementation can be improved by dividing the buffer into two equal parts: one for the producer (data collector from the device) and the other for the consumer (the task that copies the so far collected points to the user memory). The producer and consumer would switch between buffers after every 500 point transaction. Producer should send an interrupt to the consumer after filling the corresponding half (for example first half) of the buffer and switches to the other half to fill it. The consumer starts copying the data from the corresponding half (here first half) after receiving the interrupt from consumer. After

copying all the data from the first half, the consumer switches to the other half and waits until the next interrupt comes. In order to prevent race condition, each half of the buffer should be protected by a locking mechanism like mutex. Although since consumer's speed is way higher than the producer's speed, in most cases, there would be no problem in switching to the other part of buffer by the producer but using lock is necessary.