Prof Smruti Ranjan Sarangi

Operating Systems
Minor 2
COL 331, 633, and ELL 405

50 marks – 5 questions

EASY

1. What are the four necessary conditions for a deadlock? Briefly explain each condition. (10 marks)

2. Answer the following questions briefly (1-2 lines):
   1. Why do FIFO systems in your view suffer from the Belady's anomaly? (3 marks)
   2. Is implementing theoretical LRU practical? Justify your answer. (2 marks)
   3. How do we practically implement LRU? (3 marks)
   4. Does the Banker's algorithm prevent starvation? Justify your answer. (2 marks)

MEDIUM

3. Consider a typical disk that rotates at 15,000 rotations per minute (RPM) and has a transfer rate of 50 MegaBytes/sec. If the average seek time of the disk is twice the average rotational delay and the controller's transfer time is 10 times the disk transfer time, the average time (in milliseconds) to read or write a 512 byte sector of the disk is? Justify your answer. (10 marks)

   Disk latency = Seek time + rotation time + transfer time + controller time
   Transfer time = (block size) / (transfer rate)

4. Let us create an operating system for Twitter. Assume that each tweet (small piece of text) is stored as a small file. The file size is limited to 256 bytes. Given a tweet, a user would like to take a look at the replies to the tweet, which are themselves tweets. Furthermore, it is very much possible that a tweet may be retweeted (posted again) many many times. In every "retweet" a user's friends are able to see the "retweet" (new post). Note that there are no circular dependences. It is never the case that: (1) A tweets, (2) B sees it because B is A's friend, (3) B retweets the same message, and (4) A gets to see the retweet. (10 marks)

   Design a file system to store such tweets. How is the file system organized on the disk?

HARD

5. We often transfer data between user programs and the kernel. For example, if we want to write to a device, we first store our data in a character array, and transfer the array to the kernel. In a simple implementation, the kernel first copies data from the user space to the kernel space, and then proceeds to write the data to the device. Instead of having *two copies* of the same data, can we have a *single copy* of the data? This will lead to a more high performance implementation. How do we do it, without compromising on security?

   Now, consider the reverse problem, where we need to read a device. Here also, the kernel first reads the data from the device, and then transfers data to the user's memory space. How do we optimize this, and manage with only a *single copy* of data?
   [**NOTE**: The user's array can be anywhere in its memory space – in the heap, stack, or data section. Before allocating any such array, the user never asks the kernel. However, later on the user can pass a pointer to the array as an argument of a system call.]