En-Yu (Mike) Lee, elee353

Question 1

There is a comment saying " Wow! " in the switcher function in OSA1.c . This is to indicate something very bad is happening here. Explain what it is.

The free() function is not normally re-entrant and uses a static data structure to track free memory blocks.

As a result, it would be problematic in a multithreaded application and no library functions that allocate or free memory are reentrant. This includes functions that allocate space to store a result.

It can easily introduce memory leaks into the application code implemented using the free() and malloc() functions. This is caused by memory being allocated and never being deallocated. Such errors tend to cause a gradual performance degradation and eventual failure. This type of bug can be very hard to locate.

Question 2

Why are the time-consuming calculations in threads3.c required in order to demonstrate the effectiveness of the pre-emptive scheduler?

Pre-emption is the act of temporarily interrupting a [task](https://en.wikipedia.org/wiki/Task_(computing)) being carried out with the intention of resuming the task at a later time.

The time-consuming calculations are required to demonstrate the pre-emptive scheduling behaviour. The thread should not finish its tasks before it gets interrupted and gets switched to another thread. The time-consuming calculations extend the time the thread required to process its tasks and hence allow the pre-emptive scheduling to happen before the thread finishes.

Question 3

In threads3.c there is some code around the call to rand() to block signals and then allow them again.

Explain what can happen if this is not done. Also give an explanation as to why this can happen.

The function rand() is not reentrant, since it uses hidden state that is modified on each call. **If the** function rand() is called multiple times **within a certain granularity period, t**he random number generator will be re-seeded with the same value. Duplicate and non-random numbers can be returned.

This can happen if we do not block the signals before calling the rand() function. Multiple threads can call the rand() function **within a certain short period, making it return** duplicate and non-random numbers. This defeats the purpose of the rand() function.