**Part 1: Theoretical Questions**

1. siglongjmp and sigsetjmp
   1. sigsetjmp sets a save point for the current user-thread.

It saves the PC, SP and CPU registers in a struct to be used when siglongjmp is called.

siglongjmp swaps between user-threads. Given the state of a thread, saved in the struct mentioned before, the function swaps the current running thread with all the necessary data stored in the struct. It causes the program to jump to the code where sigsetjmp was called. It also sets the corresponding sigsetjmp return value to be an integer value of choice ‘val’.

* 1. The sigsetjmp function receives a sort of Boolean value for whether to keep the masking of the current thread when returning to that point using siglongjmp. So the user has the ability to save the current masking for the thread or to forget about them.

1. One use of user-level threads would be running a program that performs multiple tasks at the same time where the tasks are no I/O related.

User-level threads are a reasonable choice here because it has less overhead when compared to kernel-level threads or running multiple processes.

In addition, the usual problem with user-level threads, which is that I/O operations halt the entire process, doesn’t hold here since, as mentioned, the tasks are computational.

1. **Advantages:**

Each process has its own address space, which provides better isolation.

Since processes are isolated, they are more robust against errors and memory corruption. One process crashing does not take down the entire application.

Threads share the same process resources, which means that resource limits set for the process apply to all threads. Having multiple processes allows different resources for each tab and more resources in total.

**Disadvantages:**

Creating a new process involves more overhead than creating a thread.

Inter-process Communication is more complex and slower compared to threads, which can share memory directly.

Processes typically have a higher resource consumption.