Catherine Schnelle
Proj02

1)Briefly explain why this application would be difficult to write using multiple processes instead of threads. This is because multiprocessing is a system that has more than two processors whereas multithreading is a technique that allows a single process to have multiple segments that run concurrently. The multiprocessing process is very slow and resource specific versus multithreading's more economical efficiency with time and resources.

2) What is the significance of 'workperformed'? How is it used?
The workperformed flag in the case of a thread being in mid swap so that the string is kept temporarily in order while the swap is in process, this controls multiple access while also saving from data loss, saving the shared variable in this critical section from mutual access.

3) Explain exactly what is the type of 'fp' in the following declaration: void *(*fp)(void *)
fp is a pointer, pointing to a function that takes an argument of type void and returns a pointer of type voice.

Questions:

1. **Why do we not detach any of the enzyme threads? Would the program function if we detached the sleeper thread?** This would cause the program to terminate if the thread object is no longer linked to any thread function.

2. **Why does the program use sched_yield? What happens if this is not used? Will the swap counts always be identical?** This function is used to check if other threads at the same priority are ready to run. If so, then the calling thread yields by placing itself at the end of the queue. The swap count will always change according to the increment of the values.

3. **Threads are cancelled if the string contains a 'C' e.g. "Cherub". Why do we not include cancelled threads when adding up the total number of swaps?**
Because we want the swap count to remain in synchronicity. Including cancelled threads would make them asynchronous. A thread may need to be cancelled with all further operations of a set of threads becoming unnecessary or not wanted.

4. **What happens when a thread tries to join itself? (You may need to create a test program to try this? Does it deadlock? Or does it generate an error?**
If the current thread calls the join() function and attempts to join itself, this will leave the current thread waiting until it completes thereby waiting forever.

5. **Briefly explain how the sleeper thread is implemented.**
When a thread is created, the sleeper function gets called within that thread, it will then wait p seconds and then print the statement and exit the thread, waiting for all functions used in all threads to complete.

6. **Briefly explain why PTHREAD_CANCEL_ASYNCHRONOUS is used in this MP.**
   This is used to terminate the thread immediately because the default cancelability type for newly created threads only defers termination.

7. **Briefly explain the bug in #2 of fix the program section above.**
   This is a memory leak because setting the thread info to make a thread was outside of the function loop.