Sebastian Torrejon Alonzo
Fabian Kirberg

Lab 6

1. **What exactly does sampleProgramOne intend to do (i.e., who is responsible for what operations)?**
   It demonstrates how two processes can communicate and collaborate using shared memory. They both swap the values stored in the shared memory locations.

2. **Given that, what is the program's expected output?**
   The parent process and child process will both swap the values in the shared memory locations in their respective loops. So the expected output should not change and be 0 and 1.

3. **Describe the output of the sample program as the loop values increase**
   As the value of the loop variable increases in the sample program, the output will vary, and the behavior may become more unpredictable due to a lack of synchronization between the parent and child processes.

4. **Describe precisely what is happening to produce the unexpected output. Your answer should tie into the concepts discussed involving process synchronization.**
   Both the parent and child processes are simultaneously trying to swap the values in the shared memory locations within their respective loops. There is no synchronization mechanism in place to ensure that one process completes its swap before the other begins. This means that they might interfere with each other's swaps, leading to inconsistent results.
   For example, if the parent process reads sharedMemoryPointer[0] and stores it in temp, and at the same time, the child process reads sharedMemoryPointer[1] and stores it in temp, the swap will not work as expected because they both have the same value in temp. The more loops we have, the more interleaving swaps can occur.

Sebastian Torrejon Alonzo
Fabian Kirberg

5. **Name and describe in your own words the use of the three fields of the sembuf structure**

   sem_num: When we have a spemaphore set, we can specify which semaphore in the semaphore set to manipulate by specifying a sem_num value, similar to an index value.

   sem_op: This field specifies the operation that is to be performed on the semaphore. This includes increasing or decreasing the semaphore value or testing the semaphore value which indicates that the semaphore can acquire or release resources, which allows the program to wait or signal.

   sem_flg: You can add flags for the semaphore behavior. You can choose the semaphore perform with or without waiting, undo a operation, or hande unexpected termination.

6. **What is the purpose of the SEM_UNDO flag (i.e., why would you use it)?**

   It indicates that the semaphore operation should be automatically undone if the process that performed the operation terminates unexpectedly (e.g., due to a crash or signal). So that the resources are not left in an inconsistent state if a process that acquired or released a semaphore unexpectedly exits without properly releasing the semaphore.