

Practical time: THURSDAY 5:00pm

OPERATING SYSTEMS 200

OS200 Assignment Report

CONNOR BEARDSMORE - 15504319

5/1/2016

**Source Code**

aa

**Mutual Exclusion**

Mutual exclusion was achieved in different ways for the multiprocessing and the multithreaded solutions. The multiprocessing solution utilized three POSIX semaphores. One represented a standard mutex lock, while the others representing full and empty variables. The following code segment below indicates the basic locking sequence utilized by both the producer and consumer processes:

**Producer:**

wait(empty);

wait(mutex);

// critical section

signal(mutex);

signal(full);

**Consumer:**

wait(full);

wait(mutex);

// critical section

signal(mutex);

signal(empty);

The multithreading solution also employed the use of three locks. The first was a regular POSIX mutex lock, while the others were condition variables representing the "full" and "empty" conditions. These condition variables deny "hold and wait" by forcing a thread to give up its mutex lock while waiting for the condition. Thus, deadlock is not possible in this situation. The following code segment below indicates the basic locking sequence utilized by both the producer and consumer threads:

**Producer:**

lock(mutex)

cond\_wait(full) // implicitly unlock mutex

// critical section

cond\_signal(empty)

unlock(mutex)

**Consumer:**

lock(mutex)

cond\_wait(empty) // implicitly unlock mutex

// critical section

cond\_signal(full)

unlock(mutex)

**Shared Memory**

Threads share the address space of the process they are created within and thus, no shared memory is required for threads. For the threading solution, variables were simply declared globally, so each thread could access without additional function import overheads. POSIX shared memory was used in the solution, with shm\_open(), ftruncate() and mmap() being the primary functions employed.

Child processes however do not inherit their parents address space on creation. As a result, shared memory is required for both parent and child to access the same data concurrently. All three matrices were created in a separate shared memory block, as was the subtotal struct and the struct containing the semaphore variables. The product matrix differed from the other shared memory segments in that only one process ever writes to one specific row. Hence, no synchronization was required to read and write to the product matrix.

**Zombie Processes**

The creation of zombie processes was an issue in the multiprocessing solution. The parent was required to perform work as the consumer and thus, could not wait on its child. To avoid the prevention of zombie processes, the "double fork" method was utilized. This includes forking an initial child on which the parent waits. This child then forks another child (a grandchild), immediately exits and is reaped by its parent. Since the grandchild is now an orphan, it is adopted by init which will periodically call wait. An alternative solution would have been to implement signals.

**Errors and Bugs**

I am currently unaware as to any error conditions that would cause either Part A or Part B to fail. There are limits however to the maximum number of processes and threads that can physically be handled. The maximum limit of processes my program can handle is ....... and the maximum limit of threads is......

While there is primitive error checking throughout my program, more technical error conditions such as certain file input formats can result in unspecified behaviour.

**Testing**

Both solutions have been thoroughly tested on the laboratory computers in 314.219 and on *ssh* access to these computers via "*saeshell".* Functions were created that print the entire contents of the matrices to enable error checking and debugging. The table below lists the types of all test files the program was tested on prior to submission.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

In addition to the basic test files, two test scripts were utilized to test a large number of possible number ranges and to ensure that no deadlocks were ever reached. The scripts run the both Part A and Part B on a matrix filled with values of 1, with every possible M,N and K values for 1 to 100. Copies of these test scripts have been submitted with the other test files.

**Sample Input and Outputs**

The section below indicates the input and output from the running of the several test files mentioned previously:

**References**

Silberschatz, Abraham, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. 9th ed. Reading, MA: Addison-Wesley, 1994.

Soh, Sie Teng. "Process Synchronization." Class lecture, Operating Systems from Curtin University, Perth, Australia, April 1 2016.

Soh, Sie Teng. "Process and Threads." Class lecture, Operating Systems from Curtin University, Perth, Australia, April 1 2016.