[2025 Network System Programming Homework 9]

Rules:

1. Please use C language in this homework and run your program on Ubuntu 24.04.
2. Please provide Makefile to compile your homework.
3. Do not copy the others homework definitely.
4. If you have any question, please send email to sp_ta@net.nsysu.edu.tw or drop by Room EC5018. However, TA will not help you to debug program.

Turn in your homework:

1. Please compress your homework into zip archive.
2. Naming rules: "SP_HW9.zip".
3. Upload your homework (zip file) to NSYSU Cyber University (網路大學).
4. Deadline: 2025/11/25 09:00. You cannot get any credit if you do not turn in your homework before the deadline.

Instruction of Homework 9:

The goal of this homework is to build a high-throughput *M*-Producer, 1-Consumer system on Linux. Implement the solution using processes, leveraging Shared Memory (System V) for data transfer and Semaphores (System V) for synchronization.
System Specification and Requirements:

Role	*M* Producer Processes	1 Consumer Process
Main IPC	System V Shared Memory (shmget, shmat)	System V Shared Memory (shmget, shmat)
Synchronization	System V Semaphores (semget, semop)	System V Semaphores (semget, semop)
Function	1. Each of the *M* Producers writes a unique Process ID (PID) and a random integer (1-10) pair into the shared memory *N* times each (Total *M X N* writes). 2. Must use a mutex semaphore to protect the shared memory index. 3. Must use a counting semaphore to signal the Consumer that a new data element is available.	1. Waits for the counting semaphore signal to read new data. 2. Reads the PID and integer from the shared memory buffer. 3. Calculates the total sum of all received integers. 4. Reads until the shared buffer is empty (indicated by a final signal). 5. Prints the final total sum and the number of data points processed.

Specific Requirements

1. Command Line Arguments: The program must accept two command-line arguments:
   - *M*: The number of Producer processes (e.g., *M*=10).
   - *N*: The number of data points each Producer generates (e.g., *N*=10).
2. Implement the shared memory as a Stack array of size M, where each stack slot holds a *PID/Value* pair. The system requires implementation of the standard stack operations, push() (Producer) and pop() (Consumer), which must be fully synchronized. The structure must contain:
   - An array to hold *M* data slots (*PID/Value* pairs).
   - int write_index: An index to track where the next Producer should write.
3. Semaphore Set: A single System V Semaphore set must be created and used, containing two semaphores:
   - Semaphore 0 (Mutex): Initial value 1. Used to protect the critical section (updating the write_index).

- Semaphore 1 (Full Count): Initial value 0. Used to count the number of data slots currently filled, signaling the Consumer.

4. Producer Synchronization: Before writing, the Producer must acquire the Mutex (Sem 0). After writing, it must release the Mutex (Sem 0) and increment the Full Count (Sem 1).
5. Consumer Synchronization: Before reading, the Consumer must wait/decrement the Full Count (Sem 1).
6. Termination: The Consumer knows the total expected count is *M X N*. It exits once it has successfully processed that exact number of data points.
7. Resource Cleanup: The parent process (which creates the Consumer and Producers) must wait for all children to exit and then clean up all System V resources (Shared Memory and Semaphores) using shmctl and semct.

Suggested Linux System Calls / Functions

Shared Memory	Synchronization	Process Control	Auxiliary Functions
shmget()	semget()	fork()	srand() / rand()
shmat()	semop()	waitpid()	printf()
shmdt()	semctl()	exit()	sleep() (Optional for initial delay)
shmctl()			

Example Execution Flow (Conceptual)
$ ./data_aggregator 2 3 # M=2 Producers, N=3 writes each (Total 6 points)

Producer P1 (PID 500) starts writing...
Producer P2 (PID 501) starts writing...

Consumer (PID 400) waits on Semaphore 1 (Full Count)

P1 writes (500, 7), increments Sem 1.
P2 writes (501, 3), increments Sem 1.

Consumer reads (500, 7). Sum = 7.
Consumer reads (501, 3). Sum = 10.
... continues until 6 total points processed.

Output:
Total Data Points Processed: 6
Final Cumulative Sum: [Calculated Sum]