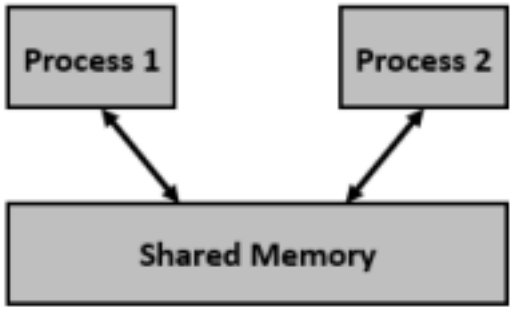
Operating Systems (CS F372)
Tutorial Sheet 5
Shared Memory

In this tutorial sheet, we shall explore the shared memory model of communication.



Interprocess communication through the shared memory model requires communicating processes to establish a shared memory region. In general, the process that wants to communicate creates the shared memory region in its own address space. Other processes that wish to communicate to this process need to attach their address space to this shared memory segment. By default, the operating system prevents processes from accessing other process memory. The shared memory model requires processes to agree to remove this restriction.

As shared memory is established based on the agreement between processes, the processes are also responsible to ensure synchronization so that both processes are not writing to the same location at the same time. If both processes try to write to the shared memory region at the same time, the result would be unpredictable and could lead to errors in one or both processes.

Two functions, **`shmget()`** and **`shmat()`** are used for IPC using shared memory. `shmget()` function is used to create the shared memory segment, while the `shmat()` function is used to attach the shared segment with the process's address space. Once you are done passing messages between processes, you can detach a process from an already attached shared memory segment using **`shmdt()`** function and delete the segment using **`shmctl()`**.

Questions:

1. Write a C program that forks to create a child process. The parent process should create and attach to a shared memory segment, then write its pid to it. The child process should

attach to the same memory segment. In the child process, read the pid from the shared memory segment and verify that it matches the pid of the parent process. Finally, detach and delete the shared memory segment.

2. Which is better for exchanging a large number of messages between processes: message queues or shared memory? Why?

3. Write two separate C programs, writer.c and reader.c, for inter-process communication using shared memory. The writer.c program should continuously read '\n' terminated strings from the terminal and communicate each string to the reader.c program via shared memory, as soon as it's read from the terminal. The reader.c program should be designed to continuously read messages from the shared memory segment, print them on the terminal. Both programs should remain active until manually terminated by the user.

Take home exercise: Write a C program that takes three file names as command line arguments. The program should spawn two child processes. The first child reads the contents of the first file, and the second child reads the contents of the second file. Both children should communicate the data they've read to the parent process using shared memory. The parent process, upon receiving data from both children, should write the combined content into the third file. The roles of the processes and the order of the files are flexible. The goal is for two processes to read from two different files and communicate their contents to a third process via shared memory, which then writes the combined content into another file.