Lab 2 Report

Name: Insiyah Ujjainwala
Email: ujjainwi@mcmaster.ca
Student ID: 400357483
Date: 18/10/2023

time-shm

- This is one method of creating an IPC mechanism that finds the time elapsed to run a command.
- We have a child process that writes the starting time to a region of shared memory.
- The parent process will read the starting time from the shared memory after the child process terminates.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <sys/shm.h>

int main(int argc, char *argv[])
{
   // Pointer to a timeval struct
   struct timeval *starting_time;

   // Create shared memory segment
   int shared_mid = shmget(IPC_PRIVATE, sizeof(struct timeval), IPC_CREAT | 0666);
   starting_time = (struct timeval *)shmat(shared_mid, NULL, 0);

   // Using fork() to create a child process
   pid_t pid = fork();

   // Child process
   if (pid == 0)
   {
       // Setting the start time
       gettimeofday(starting_time, NULL);

       // Executing the command
       execvp(argv[1], &argv[1]);

       // Error handling for execvp() system call -- executed only if execvp()
fails
       perror("Exec failed");
       return 1;
   }
   // Parent process
   else
   {
```

```
        // Waiting for the child process to finish
        wait(NULL);

        // Pointer to a timeval struct
        struct timeval ending_time;

        // Setting the end time
        gettimeofday(&ending_time, NULL);

        // Calculating and printing the elapsed time
        float elapsed = (ending_time.tv_sec - starting_time->tv_sec) +
                        (ending_time.tv_usec - starting_time->tv_usec) / 1e6;
        printf("Elapsed time: %.5f \n", elapsed);
    }

    // Removing the shared memory segment
    shmdt(starting_time);
    shmctl(shared_mid, IPC_RMID, NULL);

    return 0;
}
```

This is a shared memory IPC mechanism. It is an efficient way for processes to communicate as they can read and write to each other without much overhead and also, is one of the fastest IPC mechanisms since it allows direct access to memory addresses. It is also an appropriate way to transfer large volumes of data as there are no limitations on message size. However, since all processes share the same memory, there can be security risks associated with this method. We need to carefully manage who gets access to the shared memory regions to prevent unauthorised access.

time-pipe

- This is another method of creating an IPC mechanism that finds the time elapsed to run a command.
- The child process writes the starting time to a pipe.
- Parent process reads the starting time from the pipe after the child process terminates.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
```

```c
#include <sys/wait.h>

int main(int argc, char *argv[])
{
   // Creating a pipe to communicate between the parent and child
   int fd[2];

   // Error handling
   if (pipe(fd) == -1)
   {
       perror("Pipe failed");
       return 1;
   }

   // Using fork() to create a child process
   pid_t pid = fork();

   // Child process
   if (pid == 0)
   {
       // Closes the read end
       close(fd[0]);

       // Getting the start time of the command
       struct timeval starting_time;
       gettimeofday(&starting_time, NULL);

       // Writing the start time to the pipe
       write(fd[1], &starting_time, sizeof(struct timeval));

       // Closes the write end
       close(fd[1]);

       // Executing the command
       execvp(argv[1], &argv[1]);

       // Error handling for execvp() system call -- executed only if execvp()
fails
       perror("Exec failed");
       return 1;
   }
   // Parent process
   else
   {
       // Closes the write end
```

```
        close(fd[1]);

        // Waiting for the child to finish
        wait(NULL);

        // Reading the start time from the pipe
        struct timeval start_time, end_time;
        read(fd[0], &start_time, sizeof(struct timeval));

        // Getting the end time of the command
        gettimeofday(&end_time, NULL);

        // Finding elapsed time
        float elapsed = (ending_time.tv_sec - starting_time.tv_sec) +
                        (ending_time.tv_usec - starting_time.tv_usec) / 1e6;
        printf("Elapsed time: %.5f seconds\n", elapsed);
    }

    // Closes the read end
    close(fd[0]);

    return 0;
}
```

This question uses the pipe IPC mechanism. This is a simple IPC mechanism, pipes are fairly simple to create, and processes can easily send data to one another. Pipes are also used to synchronise processes. However, the disadvantages of this mechanism are that pipes allow data flow only in one direction, from writer to the reader. Pipes cannot be used for communication between processes on different networks.

Output

```
insiyah@cs3sh3:~$ cd Desktop
insiyah@cs3sh3:~/Desktop$ gcc time-pipe.c -o time-pipe
insiyah@cs3sh3:~/Desktop$ gcc time-shm.c -o time-shm
insiyah@cs3sh3:~/Desktop$ ./time-pipe ls
time-pipe  time-pipe.c  time-shm  time-shm.c
Elapsed time: 0.00381 seconds
insiyah@cs3sh3:~/Desktop$ ./time-shm ls
time-pipe  time-pipe.c  time-shm  time-shm.c
Elapsed time: 0.00518
insiyah@cs3sh3:~/Desktop$
```