COMPSCI 3SH3 Winter, 2021
Student name: Khizar Siddiqui
Student ID: 400109902
Date: 12-February-2021

Lab 2 Report

Q1: time-shm.c

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <sys/time.h>

int main(int argc, char **argv) {
    // For shared memory
    const int SIZE = 4096;
    const char *name = "OS";
    int fd;
    char *ptr;
    fd = shm_open(name, O_CREAT | O_RDWR, 0666);
    ftruncate(fd, SIZE);
    ptr = (char *)
    mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

    // For child process
    pid_t pid;
    pid = fork();

    // Storing time
    struct timeval before, after;
    // Error creating child
    if (pid < 0) {
        fprintf(stderr, "Unable to create child processes \n");
        return 1;
    }
    // Child process
    else if (pid == 0) {
        // Get start time and add to shared memory
        gettimeofday(&before, NULL);
        char convert[20];
        sprintf(convert, "%ld", before.tv_usec + before.tv_sec*1000000);
        sprintf(ptr, "%s", convert);
        ptr += strlen(convert);
        // Execute argument
        execvp(argv[1], &argv[1]);
    }
    // Parent process
    else {
        // Wait for completion of child
        wait(NULL);
        // Take end time and subtract from start time to show total time
        gettimeofday(&after, NULL);
        float time = after.tv_usec + after.tv_sec*1000000 - atof((char *)ptr);
        printf("Elapsed time: %.5f\n", time/1000000);
        // Remove shared memory
        shm_unlink(name);
        return 0;
    }
}
```

My code creates a shared memory object initially. Then a child process takes the current time (by adding the seconds and microseconds after conversion) and writes to this shared memory. The parent process waits for the child process to complete, once it does it extracts the start time from the shared memory and the current time and takes the difference of both.

- Advantage: Faster than pipes as it uses less resources
- Disadvantage: Multiple processes can access shared memory at the same time unless semaphores or mutex is used

Q2: time-pipe.c

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/time.h>

#define READ_END 0
#define WRITE_END 1

int main(int argc, char **argv) {
    // Create pipe
    int fd[2];
    if (pipe(fd) == -1) {
        fprintf(stderr,"Pipe failed\n");
        return 1;
    }

    // Storing time
    struct timeval before, after, rewrite;

    // For child process
    pid_t pid;
    pid = fork();

    // Error creating child
    if (pid < 0) {
        fprintf(stderr, "Fork Failed\n");
        return 1;
    }
    // Parent process
    if (pid > 0) {
        // Wait for completion of child
        wait(NULL);
        // Close write end and get end time
        close(fd[WRITE_END]);
        gettimeofday(&after, NULL);
        // Get start time from pipe and subtract from start time to show total time
        read(fd[READ_END], &rewrite, sizeof(rewrite));
        float time = (after.tv_usec + after.tv_sec*1000000) -
                     (rewrite.tv_usec + rewrite.tv_sec*1000000);
        printf("Elapsed time: %.5f\n", time/1000000);
        // Close read end
        close(fd[READ_END]);
    }
    // Child process
    else {
        // Close read end and write start time to pipe
        close(fd[READ_END]);
        gettimeofday(&before, NULL);
        write(fd[WRITE_END], &before, sizeof(before));
        // Close write end and execute argument
        close(fd[WRITE_END]);
        execvp(argv[1], &argv[1]);
    }
    return 0;
}
```

My code creates a pipe initially. Then a child process takes the current time (by adding the seconds and microseconds after conversion) and writes to this pipe. The parent process waits for the child process to complete, once it does it extracts the start time from the pipe and the current time and takes the difference of both. Once the pipe is closed (for parent or child) then it switches to the other process (child or parent).

- Advantage: Easy to setup and no need to free up memory after done using
- Disadvantage: Limited data capacity compared to shared memory