Lab 2 Report

Name: Ricky Fan
Email: fanh11@mcmaster.ca
Student ID: 400248976
Date: 23/10/2022

time-shm.c

This C program determines the amount of time needed for a command to run on the command line using the shared memory IPC mechanism. The program can be run as

./time-shm <command>

and will output and amount of time elasped to run the input command.

```
/**
 * time-shm.c
 *
 * C program that calculates
 * the execution time of commands
 * using share memory object
 *
 * fanh11@mcmaster.ca
 */

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

int main(int argc, char *argv[])
{
```

```
// define the size of the share memory object
const int SIZE = 4096;
// define the name of the share memory object
const char *name = "OS";
// a file descriptor integer referring to the
   share memory
int fd;
// a pointer to the shared memory object
void *ptr;
// process id
pid_t pid;
// variable to keep track of the current time
struct timeval current_time;

/* create the shared memory segment */
fd = shm_open(name, O_CREAT | O_RDWR, 0666);

/* configure the size of the shared memory segment
    */
ftruncate(fd, SIZE);

/* map the shared memory segment in the address
   space of the process */
ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE,
   MAP_SHARED, fd, 0);

/* fork a child process */
pid = fork();

if (pid < 0)
{
    fprintf(stderr, "Fork failed");
    return -1;
}
/* child process */
else if (pid == 0)
{
    /* get a record of the current time before
       executing the input command */
    gettimeofday(&current_time, NULL);
```

```
    /**
     * calculate the current time by converting it
         into seconds
     * then write the result to the shared memory
        region
     */
    sprintf(ptr, "%lf", current_time.tv_sec +
       current_time.tv_usec / 1000000.0);

    /* execute the input command */
    execvp(argv[1], &argv[1]);
}
/* parent process */
else
{
    /* wait for the child process to complete */
    wait(NULL);

    /* get the current time after the input
       command has completed in the child process
       */
    gettimeofday(&current_time, NULL);

    /* open the shared memory segment */
    fd = shm_open(name, O_RDONLY, 0666);

    /* read from the shared memory region */
    ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, fd,
        0);
    if (ptr == MAP_FAILED)
    {
        fprintf(stderr, "Map failed");
        return -1;
    }

    char *rmn;
    /* convert the start time record from the
       chlld process to numeric value */
    double st_sec = strtod((char *)ptr, &rmn);
```

```
        /* calculate the current time by converting it
            into seconds */
        double ct_sec = current_time.tv_sec +
           current_time.tv_usec / 1000000.0;

        /* display the elasped time for executing the
           input command */
        printf("\nElasped time: %lf seconds\n", ct_sec
            - st_sec);

        /* remove the shared memory segment */
        shm_unlink(name);
    }

    return 0;
}
```

time-pipe.c

This C program determines the amount of time needed for a command to run on the command line using the pipe IPC mechanism. The program can be run as

./time-pipe <command>

and will output and amount of time elasped to run the input command.

```
/**
 * time-pipe.c
 *
 * C program that calculates
 * the execution time of commands
 * using pipe
 *
 * fanh11@mcmaster.ca
 */

#include <sys/types.h>
#include <stdio.h>
```

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

// define the index for the read end of the pipe
#define READ_END 0
// define the index for the write end of the pipe
#define WRITE_END 1

int main(int argc, char *argv[])
{
    /**
     * define an array to store the file descriptors
     * for the read & write end of the pipe
     */
    int fd[2];
    // process Id
    pid_t pid;
    // variable to keep track of the current time
    struct timeval current_time;

    struct timeval *ptr = &current_time;

    /* create the pipe */
    if (pipe(fd) == -1)
    {
        fprintf(stderr, "Pipe failed");
        return -1;
    }

    /* fork a child process */
    pid = fork();

    if (pid < 0)
    {
        fprintf(stderr, "Fork failed");
        return -1;
    }
    /* child process */
    else if (pid == 0)
```

```
{
    /* close the read end of the pipe when write
       is about to happen*/
    close(fd[READ_END]);

    /* get a record of the current time before
       executing the input command */
    gettimeofday(&current_time, NULL);

    /* write the address of the current time to
       the pipe */
    write(fd[WRITE_END], ptr, sizeof(struct
       timeval));

    /* close the write end of the pipe*/
    close(fd[WRITE_END]);

    execvp(argv[1], &argv[1]);
}
/* parent process */
else
{
    /* wait for child process to complete */
    wait(NULL);

    /* close the write end of the pipe when read
       is about to happen*/
    close(fd[WRITE_END]);

    /* read the address of the start time record
       from the pipe */
    read(fd[READ_END], ptr, sizeof(struct timeval)
       );

    /* calculate the start time by converting to
       seconds */
    double st_sec = ptr->tv_sec + ptr->tv_usec /
       1000000.0;

    /* get the current time after the input
```

```
            command has completed in the child process
            */
        gettimeofday(&current_time, NULL);

        /* calculate the current time by converting to
            seconds */
        double ct_sec = current_time.tv_sec +
            current_time.tv_usec / 1000000.0;

        /* display the elasped time for executing the
            input command */
        printf("\nElapsed time: %lf seconds\n", ct_sec
            - st_sec);

        /* close the read end of the pipe*/
        close(fd[READ_END]);
    }

    return 0;
}
```