CS 4375 – Operating Systems
Lab 1 Report

Context Switching

By: Matthew S Montoya
ID: 88606727
Instructor: Dr. Steven Roach
TA: Aneesa Judd

1 Introduction

In this lab, a program will further expand on the topic of context swapping, which has been introduced in lecture throughout the last four weeks. That is, a program will measure the time it takes to swap between two processes that are executed on a single CPU core. The objective for this lab is to measure the cost (time) it takes for the OS to perform context swapping, exploring this process by running experiments. This report details the findings.

2 Proposed Solution & Design Implementation

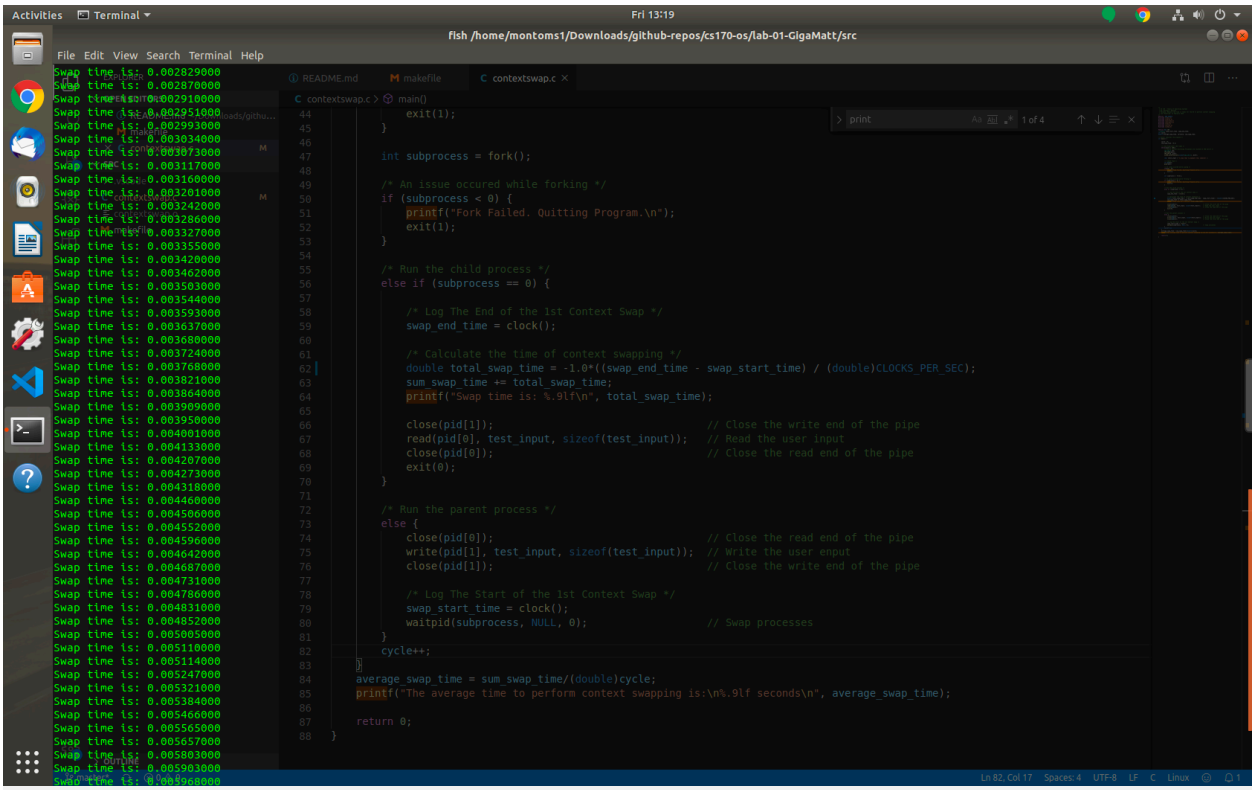
My initial design implementation called for forking the program once, using two pipes in the process. This design method was derived from the pipe() tutorial by [2]. In the tutorial, it was noted that connecting one process to another could involve the use of two pipes; one pipe to write from the user input (process one) to a pipe and read from the pipe to the second process (user output), and a second pipe to redo this process from the second process to the first process. An issue with this initial design was my misinterpretation of the term process.

In designing the initial implementation, I understood _process_ to be the execution of an external program (i.e. *read_input.c*). This led to a design where two pipes would execute external *read_input.c* and *write_output.c* programs through the *execlp* command. This conflicted with step 2 in the six-step process described above, as these programs transformed from function calls to executable programs with their own main() functions. This misinterpretation existed for a week until I questioned why I was calling each external program twice (four (4) total external program calls) to perform the same computation, leading to inefficient, extra code in three (3) .c programs.

Because of the inefficiency of using two pipes to duplicate a single process, my next design implementation called for forking the program once, using one pipe in the process. This design was implemented by referencing the algorithm provided in slide 10 of [3] to understand how a single pipe could functionally serve to accomplish the end goals of this lab. After learning more about fork() and pipe(), careful consideration was given to these functions wherein, should any of the two (2) function calls result in an error, the program would exit noting the error(s). The program runs 100 times to ensure there is enough data to calculate the mean and variance of the swap times.

3 Experimental Results

Partial output of the code (found in the appendix) and the calculation of the variance is shown below. Note: Some images may include code (which can also be found in the Appendix section), due to terminal transparency settings.



Note: From the images above, we can see that with a population size of 100, we have a mean of 0.00324543 seconds and a variance of ~ 2.49*(10^-6) seconds.

4 Discussion

In this lab, a program expanded on the topic of context swapping, which has been introduced in lecture throughout the last four weeks. That is, a program measured the time it took to swap between two processes that are executed on a single CPU core. The objective for this lab is to measure the cost (time) it takes for the OS to perform context swapping, exploring this process by running experiments. This report details the findings.

Throughout this lab, I learned the logic of the fork() and pipe() functions, and how to properly implement them in the C language. This helped me learn and understand the concept of context swapping, as well as how to measure the average time and variance between the swaps. Despite minor output issues (negatives instead of positives, means equaling zero) on an Ubuntu VM, running the code on the UTEP 3432 *SystemsVM* virtual

machine counters some of these issues. Given the length of the test string (found in the appendix, the resulting mean and variance (measured in seconds) match the output we did not expect. This results for the variance seem too large. However, when graphing the data, the times increase linearly, with the measured time of the 100 context swaps being within milliseconds of each other on a single CPU core. This part is consistent with what we expect.

5 Appendix

5.1— Disclosure

- Disclosure: Though keeping the original digital format of the code (indention lengths, spaces, etc.) the use of Microsoft Word has reformatted the code in the document to be of different space and tab lengths. The font size has been reduced to help alleviate some visual misrepresentations.

5.2— Student Code

5.2.1— *contextswap.c*

```
/*
 * CS 170 - Theory of Operating Systems
 * By: Matthew S Montoya
 * Purpose: Measure the time it takes for the OS to perform context swapping
 * Last Modified: 14 February 2020
*/
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
#include <sched.h>

#define MAX 300
__clock_t swap_start_time, swap_end_time;
int cycle;
double average_swap_time, variance, sum_swap_time;

/* Runner executes the program */
int main() {

    cycle = 0;
    sum_swap_time = 0.0;

    /* Loop Experiment 100 times */
    while(cycle < 100){
        /* Ensure Context-Switching Processes are located on CPU Core 0 */
        cpu_set_t set;
        CPU_ZERO(&set);
        CPU_SET(0, &set);
        sched_setaffinity(0,sizeof(cpu_set_t), &set);

        char *test_input = "I pray that I graduate this semester.";

        /* Create Pipe */
        int pid[2];
        pipe(pid);

        /* An issue occured while piping */
        if(pid < 0){
            printf("Pipe Failed. Quitting Program.\n");
            exit(1);
```

```
                }

                int subprocess = fork();

                /* An issue occured while forking */
                if (subprocess < 0) {
                    printf("Fork Failed. Quitting Program.\n");
                    exit(1);
                }

                /* Run the child process */
                else if (subprocess == 0) {

                    /* Log The End of the 1st Context Swap */
                    swap_end_time = clock();

                    /* Calculate the time of context swapping */
                    double total_swap_time = (swap_end_time - swap_start_time) /
(double)CLOCKS_PER_SEC;
                    sum_swap_time += total_swap_time;
                    printf("Swap time is: %.9lf\n", total_swap_time);

                    close(pid[1]);    // Close the write end of the pipe
                    read(pid[0], test_input, sizeof(test_input));   // Read the user input
                    close(pid[0]);    // Close the read end of the pipe
                    exit(0);
                }

                /* Run the parent process */
                else {
                    close(pid[0]);    // Close the read end of the pipe
                    write(pid[1], test_input, sizeof(test_input));  // Write the user
enput
                    close(pid[1]);    // Close the write end of the pipe

                    /* Log The Start of the 1st Context Swap */
                    swap_start_time = clock();
                    waitpid(subprocess, NULL, 0);                  // Swap processes
                }
                cycle++;
            }
            average_swap_time = sum_swap_time/(double)cycle;
            printf("The average time to perform context swapping is:\n%.9lf seconds\n",
average_swap_time);

            return 0;
        }
```