48450 Real Time Operating Systems
Assessment Task 3 Report

Due Date: 23:59 PM, 8 May 2022

Student Name: Jeong Bin Lee

Student ID: 12935084

Contents

Introduction

The purpose of this assignment is to develop two programs which involve CPU scheduling and memory management demonstrating key concepts learnt in the subject 48450 RTOS. The first program will use two threads controlled by using a semaphore. The first thread is used to simulate and schedule the processes as shown in figure 1 below, using shortest-remaining-time-first (SRTF) and to pass the average waiting time and turnaround time data to a FIFO (or a named pipe). The second thread will receive the data from FIFO and save the data into a .txt file. The file name will be passed in from the command line when executing the program.

Process ID	Arrive time	Bust time
1	8	10
2	10	3
3	14	7
4	9	5
5	16	4
6	21	6
7	26	2

Figure 1. CPU scheduling table.

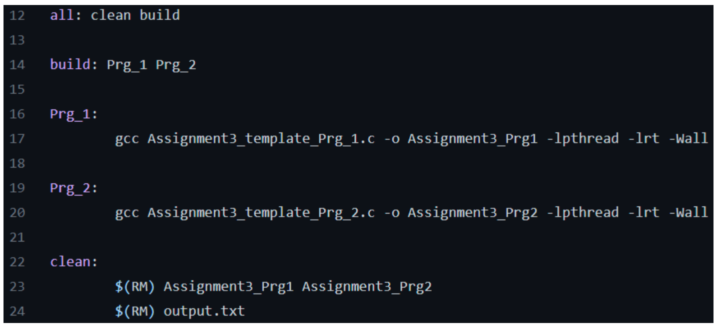
The second program is to demonstrate memory management and signals using FIFO algorithm to simulate page-replacement for virtual memory. The FIFO is to use 4 frames and the reference string (or array of numbers) as shown in figure 2 below. The program is required to check if a page fault has occurred and then output the current frame state including page number. The number of frames must also be given from the command line.

7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1,7,5

Figure 2. List of number for FIFO paging.

Compile & Run

To compile and run both the program a makefile is provided. Running command "*make clean*" will remove all built programs and the output.txt file. Running command "*make Prg_1*" or "*make Prg_2*" will build either program 1 or program 2. Running command "*make build*" will build both programs and running "*make*" will clean and build all programs. Compiling process used "*gcc Assignment3_template_Prg_1.c -o Assignment3_Prg1 -lpthread -lrt -Wall*" to ensure all warning are notified. See figure 3 for more information.



```
all: clean build

build: Prg_1 Prg_2

Prg_1:
	gcc Assignment3_template_Prg_1.c -o Assignment3_Prg1 -lpthread -lrt -Wall

Prg_2:
	gcc Assignment3_template_Prg_2.c -o Assignment3_Prg2 -lpthread -lrt -Wall

clean:
	$(RM) Assignment3_Prg1 Assignment3_Prg2
	$(RM) output.txt
```

Figure 3. Code inside makefile.

Program 1

Multiple functions were implemented to achieve program 1:

- input_process - is used to simulate and create the table of processes given using the process ID, arrive time and burst time.
- process_SRTF - function is used to schedule the processed as created. This is where the average wait time and average turnaround time is measured.
- calculate_average - function will calculate the average values.
- print_results - function prints the averages calculated.
- send_FIFO - function will create a named pipe and send data through and also signalling the semaphore before opening the FIFO.
- read_FIFO - function will receive the data and output it to a .txt file.

Implementation

Before writing the code, the process was calculated by hand to have an expected result. This result can be seen in figure 4 as a Gantt Chart. After analysing the expected result, the program was implemented step by step starting with thread 1 to creating the process, scheduling them and calculating the averages. Then printing the results on thread 2 and finally writing the output file. Thread 1 runs input_process, process_SRTF, calculate_average and send_FIFO functions in order. Thread 2 runs print_results and read_FIFO after waiting for the semaphore.

Gantt Chart

The average wait time and turnaround time is hand calculated as shown below:

$$avg_{wait} = \frac{[(36-9)+(10-10)+(29-14)+(13-10)+(17-16)+(21-21)+(27-26)]}{7} = 6.714286$$

$$avg_{turn} = \frac{[(38-1)+(6-3)+(29-7)+(10-2)+(14-9)+(20-14)+(22-19)]}{7} = 12$$

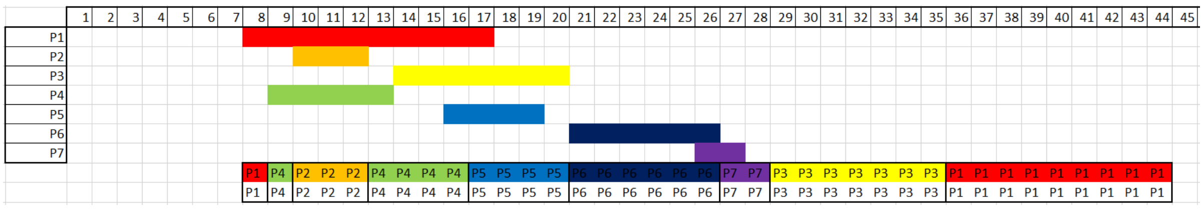
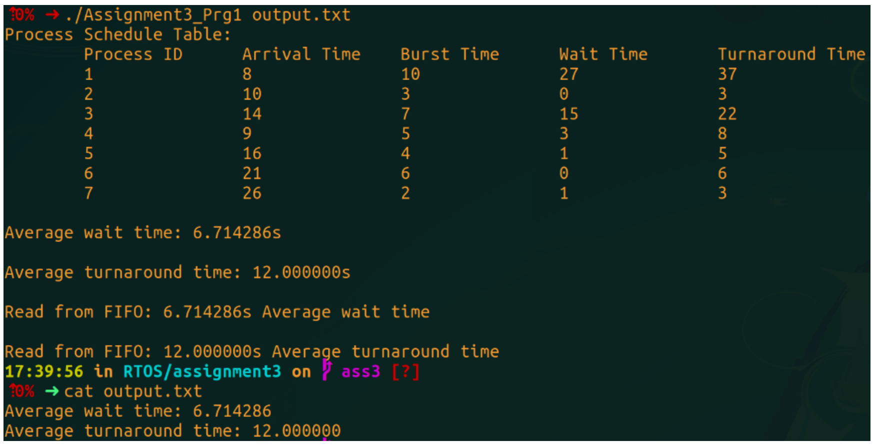


Figure 4. Shortest-remaining-time-first process scheduling Gantt Chart.

Result

Program 1 result is shown in figure 5. The average wait time and the average turnaround time is as calculated above. The result also shows the output.txt file which recorded the average wait time and the turnaround time and saved it.



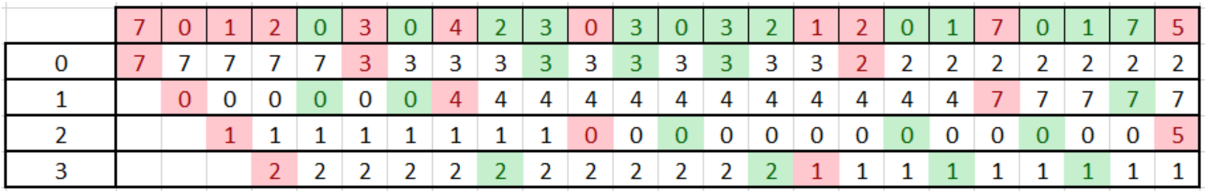
```
 10% ➜ ./Assignment3_Prg1 output.txt
Process Schedule Table:
        Process ID       Arrival Time     Burst Time       Wait Time        Turnaround Time
        1                8                10               27               37
        2                10               3                0                3
        3                14               7                15               22
        4                9                5                3                8
        5                16               4                1                5
        6                21               6                0                6
        7                26               2                1                3

Average wait time: 6.714286s

Average turnaround time: 12.000000s

Read from FIFO: 6.714286s Average wait time

Read from FIFO: 12.000000s Average turnaround time
17:39:56 in RTOS/assignment3 on  ass3 [?]
 10% ➜ cat output.txt
Average wait time: 6.714286
Average turnaround time: 12.000000
```

Figure 5. Program 1 result in command line.

Program 2

Program 2 uses 1 function SignalHandler to handle the signal when ctrl+c is pressed, rest of the program is implemented in the main function. Initially, the program will set up the SignalHandler, then set all values in frame to -1. The main part of the program is looping through the frame and reference string, the two values will be compared to see if there is a page fault. If a page fault is detected, the value is written to the FIFO's next position. Then number of page faults and FIFO's writing position is incremented. Before the next iteration of loop, there is a final check to see if end of FIFO has been reached. If this is true, the position is reset to 0 to loop around the FIFO. Lastly, the program will go in an infinite loop waiting for the interrupt signal or for the user to press ctrl+c, which will then print the number of page faults that has occurred throughout the program. See the flowchart for more information.

The expected result is as shown in figure 6, where the red squares indicate a page fault, and the green squares indicates a hit. The left column indicates the frame position inside the FIFO and the top string of numbers shows the reference string as shown previously in figure 2. See the result section below for comparison.



	7	0	1	2	0	3	0	4	2	3	0	3	0	3	2	1	2	0	1	7	0	1	7	5
0	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2
1		0	0	0	0	0	0	4	4	4	4	4	4	4	4	4	4	4	4	7	7	7	7	7
2			1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	5
3				2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1

Figure 6. Expected result of the simulated page-replacement in FIFO.

Flow Chart

The flowchart in figure 7 shows the flow of program 2 from initialisation to waiting for interrupt signal. Once interrupt signal is generated by the user by pressing ctrl+c, the program will print the number of page fault that has occurred and exit the program.

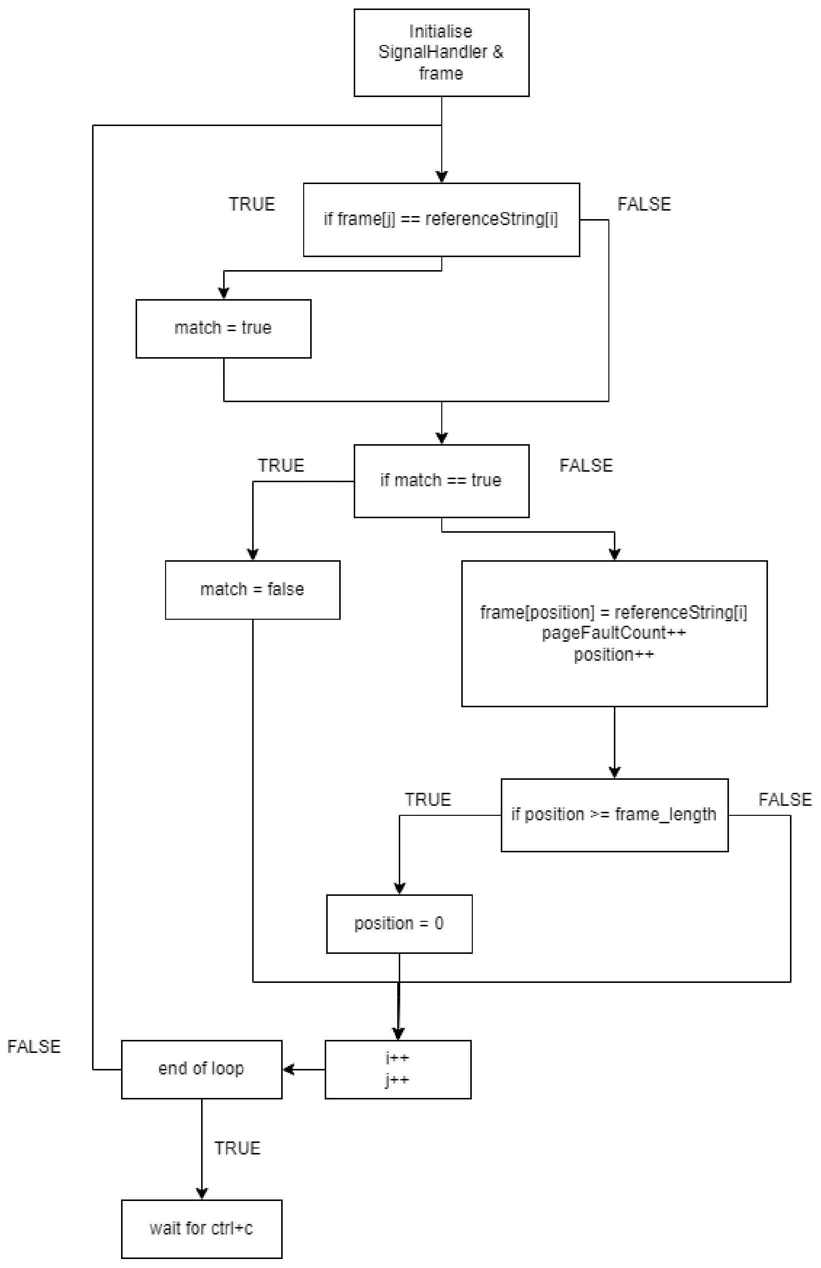
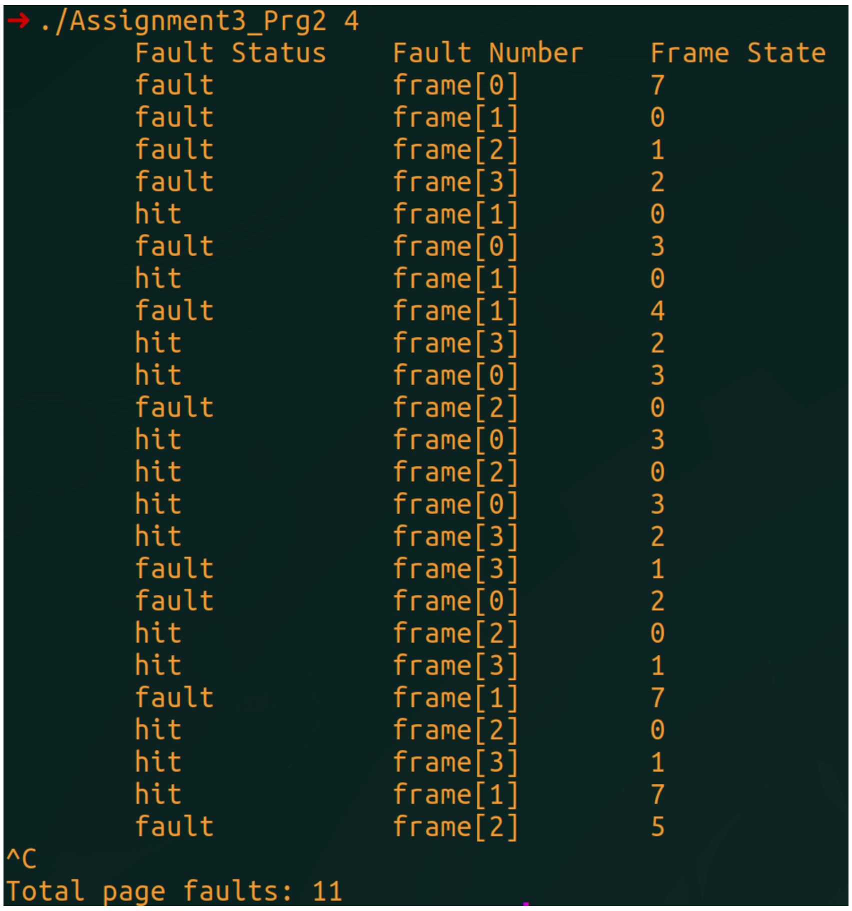


Figure 7. Flow chart for program 2.

Result

The result shows the program 2 executed with 4 frames and reference string from figure 2. Debug prints shows when a fault or a hit has occurred while running the program. After ctrl+c is pressed (shown as ^C) the total page fault of 11 is shown as expected from figure 6.



```
➔ ./Assignment3_Prg2 4
        Fault Status    Fault Number    Frame State
        fault           frame[0]        7
        fault           frame[1]        0
        fault           frame[2]        1
        fault           frame[3]        2
        hit             frame[1]        0
        fault           frame[0]        3
        hit             frame[1]        0
        fault           frame[1]        4
        hit             frame[3]        2
        hit             frame[0]        3
        fault           frame[2]        0
        hit             frame[0]        3
        hit             frame[2]        0
        hit             frame[0]        3
        hit             frame[3]        2
        fault           frame[3]        1
        fault           frame[0]        2
        hit             frame[2]        0
        hit             frame[3]        1
        fault           frame[1]        7
        hit             frame[2]        0
        hit             frame[3]        1
        hit             frame[1]        7
        fault           frame[2]        5
^C
Total page faults: 11
```

Figure 8. Result of program 2.

Reference

A. Silberschatz, P. B. Galvin & G. Gagne, 2012, Operating System Concepts, 9 th edn, John Wiley & Sons, New York.