Real-time Operating system - 48450

Assignment 3 Report

Ashwin Rajesh - 14259321

Contents

1 Introduction 2
2 Building the project 2
2.1 Executing programs 2
3 Program 1 2
3.1 Outline 2
3.2 Gantt Chart 2
3.3 Implementation 3
3.3.1 Round Robin Scheduling (Thread 1) 3
3.3.2 Output to file (Thread 2) 3
3.3.3 Results 3
4 Program 2 4
4.1 Outline 4
4.2 Implementation 4
4.2.1 Memory management 4
4.2.2 Signal (Ctrl-C) 4
4.2.3 Results 4
5 Conclusion 5

1 Introduction

This assignment consists of two programs. Program 1 aims to simulate a round-robin CPU scheduling algorithm on 7 processes, with differing arrival times and burst times. The program then sends average wait and turnaround times to a different thread using a named pipe. Program 2 implements a FIFO page scheduling algorithm, and waits for a SIGINT interrupt from the user before outputting the total number of page faults from the algorithm. These results are summarised to show the strengths and weaknesses of algorithms for CPU and page scheduling in the realm of operating systems.

2 Building the project

The code for both programs can be built using the makefile, as described below.

```
make -f makefile.txt
```

```
[vanilla@viking-g15 rtos_assignment_3]$ make -f makefile.txt
gcc -Wall -lpthread -lrt -o assign3_part1  queue.o Assignment3_template_Prg_1.c
gcc -Wall -lpthread -lrt -o assign3_part2  Assignment3_template_Prg_2.c
[vanilla@viking-g15 rtos_assignment_3]$
```

Figure 1: Build command

2.1 Executing programs

Program 1:

```
./assign3_part1 4 output.txt
```

Program 2:

```
./assign3_part2 4
```

3 Program 1

3.1 Outline

Round Robin scheduling allocates each task an equal share of CPU time. At any given time, a process is selected and allowed to run on the CPU for a specified time quantum. After this time has passed, the task is 'pre-empted', and stopped mid-execution, after which the algorithm context-switches to a different process. This cycle is repeated until all processes in the ready-queue finish.

Once waiting times and turnaround times are calculated for each process, the averages are sent to thread 2 using a named pipe or FIFO. A named pipe is a method of inter-process communication in Unix systems, which is an extension to the traditional pipe concept. A named pipe can last as long as the system is up, beyond the life of the process.

3.2 Gantt Chart

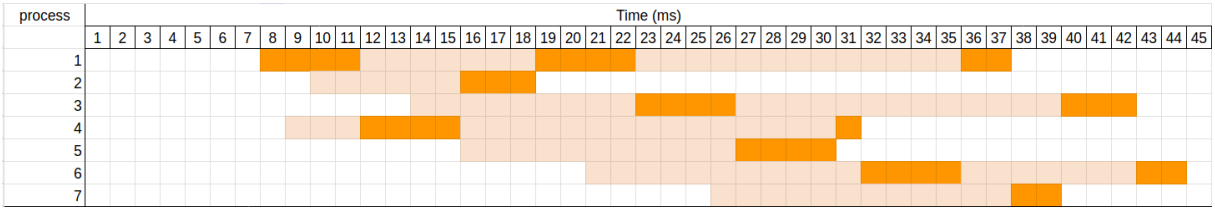

Figure 2: Round Robin Scheduling GANTT chart

3.3 Implementation

3.3.1 Round Robin Scheduling (Thread 1)

The round robin algorithm presented in this assignment firstly initiates a ready queue as well as a current time variable. The current time variable keeps track of the time that has passed since the beginning of the algorithm. The ready queue is the data structure that holds all the processes that have arrived and are ready to be run. When a process has the same arrival time as the current time, it is added into the ready queue.

The algorithm pops the first process out of the ready queue and allows it to run for the specified time quantum. While the process executes, new processes that arrive are pushed to the back of the queue. When the time quantum for the current process ends, it's execution is paused and it is placed at the end of the ready queue. This process is repeated until all processes are finished executing.

3.3.2 Output to file (Thread 2)

Once the algorithm has finished simulating round robin scheduling on the processes, the wait times and turnaround times that have been calculated for each process are averaged, and written to a named pipe.

A named pipe or FIFO, is a method of inter-process communication which is an extension to the traditional pipe concept on Unix. It can last as long as the system is up, beyond the life of the process, and can be deleted if no longer used. In the program, the FIFO is initialised using the `mkfifo()` command, which makes a special file using the given path. It is operated in a similar manner to an ordinary file, however, it has to be open at both ends simultaneously before input/output operations can take place.

In the worker2 thread, this filename can be used to read the contents that have been written to it using the `read()` function. In the program, the average wait time and turnaround time is passed in by separating the two values using a comma. This is then decoded by the `strtok()` function, which takes the source string and splits it according to a delimiter string. This way, worker2 is able to receive average wait and turnaround times and write it to the output file.

3.3.3 Results

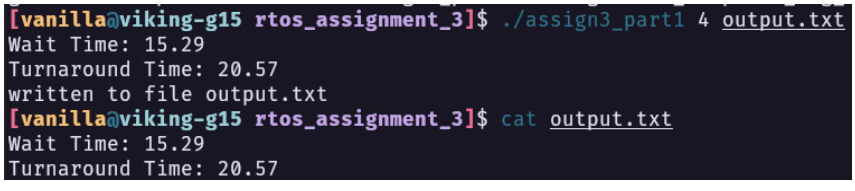

Figure 3: Round Robin wait & turnaround times

The results show that the average wait time for all 7 processes with a quant time of 7 is 15.29 ms. The average turnaround time for the processes is 20.57 ms. In a round robin implementation, no process should wait more than $(n-1)q$, or 24ms in this case. If we observe the process that waited the longest, 22 ms by process 4, we see that it is under 24ms. Generally, round robin scheduling has a higher average turnaround time compared to another scheduling algorithm such as SJF. However, it is more responsive, due to the fair nature of round robin scheduling, ensuring that all processes will get a slice of CPU time soon.

4 Program 2

4.1 Outline

Program 2 implements a FIFO based page replacement algorithm. In this algorithm, the earliest page that was loaded into the frame is replaced with a new page in the event of a page fault.

After printing all faults that occured, the program waits for the SIGINT signal from the system, which the user can administer by pressing Ctrl+C on the keyboard. The program handles this signal by printing the total number of page faults and exiting.

4.2 Implementation

4.2.1 Memory management

The algorithm begins by initialising all elements in the frame to −1, signifying that it is empty. It then begins looping through all reference string values, checking if the value exists within the frame values or not. If the value does not exist, then it means that there is a page fault. The oldest value in the frame can be kept track of by the page fault count, since we replace values in the frame *only* when there is a page fault, and in a sequential order. This frame value can then be replaced by the new page. As a result of the algorithm, the first page that is placed in the frame will also be the first page to be swapped out.

4.2.2 Signal (Ctrl-C)

The SIGINT signal is delivered to a process when a user presses Ctrl+C in the console. The default action is to terminate the process. This action can be overriden and the signal can be handled differently. This is done through the `sigaction()` function. It's parameteres are the signal to be handled, and a sigaction struct that describes the action to be taken when the signal is received. It includes the handler function that contains the code to be executed.

4.2.3 Results

```
[vanilla@viking-x200 assn3]$ ./assign3_part2 4
FAULT 1:
Frames: 7 -1 -1 -1

FAULT 2:
Frames: 7 0 -1 -1

FAULT 3:
Frames: 7 0 1 -1

FAULT 4:
Frames: 7 0 1 2

FAULT 5:
Frames: 3 0 1 2

FAULT 6:
Frames: 3 4 1 2

FAULT 7:
Frames: 3 4 0 2

FAULT 8:
Frames: 3 4 0 1

FAULT 9:
Frames: 2 4 0 1

FAULT 10:
Frames: 2 7 0 1

FAULT 11:
Frames: 2 7 5 1

^C
Total page faults: 11
```

Figure 4: FIFO page replacement faults

The results show that 11 page faults occur using a FIFO page replacement algorithm with 4 frames. Initially, there are 4 page faults consecutively, because the frame did not contain any cached pages. Once the frame started to fill up, the frequency of page faults dropped. A FIFO page replacement algorithm is a simple solution to determine which page to replace. However it suffers from Belady's Anomaly, which is a phenomenon where increasing the page frames would increase the page faults

rather than decreasing them. Other algorithms such as Least Recently Used (LRU) do not suffer from this as they assign a priority to every page.

5 Conclusion

In conclusion, implementing algorithms such as round robin CPU scheduling, as well as FIFO page replacement was beneficial in my understanding of how operating systems such as linux work. They uncovered the method by which compuer systems manage resoucres efficiently, which provided insights into how processes are managed and memory is scheduled. It also helps with designing efficient systems, by understanding trade-offs between various metrics like throughput, respons etime, and fairness.