Assignment report

Real-time Operating System - 48450

Student Name: Angze Li

Student ID: 99178333

Table of Contents

I. Introduction 3

II. Theory of operation 3

III. Operating condition 4

IV. Implementation 5

V. Experiments 8

VI. Conclusion on result analysis 10

1. Introduction

Two programs are included in this assignment, which is prog\_1 and prog\_2 respectively.

**Prog\_1** is an implementation of Round Robin CPU scheduling algorithm. Key knowledge addressed in this program are as below:

* Put theory of Round Robin algorithm into practice
* Use of pthread API and semaphores
* Real time FIFO reading/writing
* A generic Queue implementation using single linked list.
* File I/O

**Prog\_2** is an implantation of memory page replacement algorithm. Key knowledge addressed are as follow:

* Put theory of virtual memory management algorithm into practice
* A generic Queue implementation using single linked list.
* Linux signal handling

1. Theory of operation
2. FIFO (Named pipe)

Unlike the normal pipe, FIFO is the named pipe which we can operate on it with a user defined name. it can be used to achieve inter-process communication or cross threads communication.

Note that a FIFO must be open at both ends before reading or writing anything. Opening a FIFO for reading normally blocks until some other process opens the same FIFO for writing, and vice versa.

**Usage:**

To make a FIFO:

*#include <*[*sys/types.h*](https://linux.die.net/include/sys/types.h)*>#include <*[*sys/stat.h*](https://linux.die.net/include/sys/stat.h)*>*

*int mkfifo(const char \*pathname, mode\_t mode);*

To write/read from the FIFO:

*#include <*[*unistd.h*](https://linux.die.net/include/unistd.h)*>*

*ssize\_t read(int fd, void \*buf, size\_t count);*

*ssize\_t write(int fd, const void \*buf, size\_t count);*

To free the memory of the FIFO:

***#include <unistd.h>***

***int unlink(const char \*****pathname****);***

1. Queue

Queue is a well known and commonly used abstract data structure in software engineering. It only allows operations on head or tail end of the queue, just like what we do in the real life.

The queue can be either implemented using an array or linked list. Basic operation son the queue is as follow.

* Enqueue() – add an element to the end of the queue.
* Dequeue() – remove an element from the head of the queue.
* isEmpty()/isFull() – check if the queue is empty or full.

1. Signal handling

Signal in Linux system is used to notify a process or thread about the occurrence of the specific event or interrupt.

**Usage:**

**#include <signal.h>**

**typedef void (\*sighandler\_t)(int);**

**sighandler\_t signal(int** *signum***, sighandler\_t** *handler***);**

with this function, a user defined function can be set as the signal handler.

1. Round Robin

Round robin is one of many CPU scheduling algorithms. The main idea of this algorithm is that each process executes a specified time quantum, if a process can’t finish within one time quantum, it will get off the CPU and go to the end of the ready queue. When the next time quantum comes, the first element on the head of the ready queue will execute on the CPU.

1. Page replacement

Operating system manages memory using pages. Paging happens when a page fault occurs, when there no pages available for allocation.

So page replacement algorithm determines which memory page should be satisfied (swaped out from memory to the disk), and which page should be paged in (allocated to the memory).

Normally, to test the page replacement algorithm, a reference string will be used. Reference string is a sequence of pages, which will be served as the input to the program. Accordingly, the page fault number will be outputted.

1. Operating condition

It is important and necessary to understand the assignment requirements thoroughly before writing the program.

For the prog\_1. The general specification is:

1. Time quantum and file name should be given as the arguments from the terminal.
2. User should be able to input the process data such as arrival time and burst time from standard input.
3. Two threads will run currently.
4. Thread A can do round robin scheduling and calculate the average waiting and turnaround time.
5. After the scheduling and calculating, a FIFO should be created within thread A.
6. The averages should be written to the FIFO from thread A and read from thread B.
7. Thread B then should write the results in the file that given by the user.

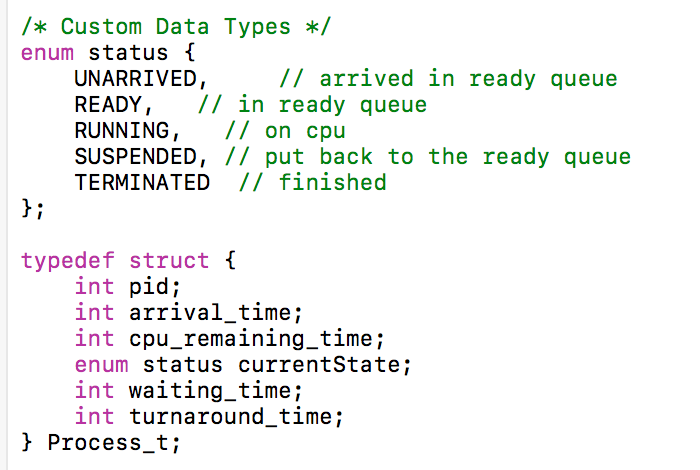
For the prog\_2. The general specification is:

1. The program should receive the number of frames from the terminal.
2. Then the program should iterate through the reference string, and check if there’s page fault occurs.
3. A state of each frame should be printed each time a page is checked
4. Upon occurrence of page faults, the page replacement algorithm will be applied.
5. The program will suspend when the paging is done.
6. When a key board interrupt signal is received, the suspended program should resume and print out the number of page faults.

1. Implementation
2. Method

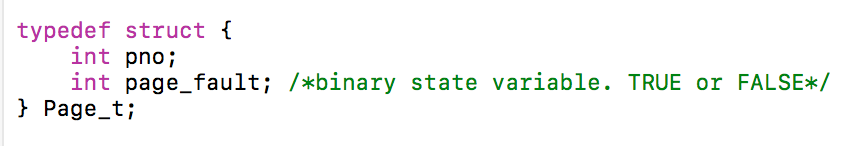
In this section, you might list how you solve the problem and how you complete your program. You might write your software design strategy about your programing.

A well defined struct (data structure) will help to write problem easily. In the prog\_1, I have defined a type of process, which is shown below:



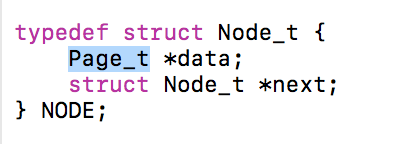
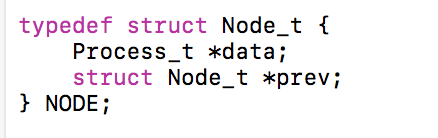
**Figure 1. data structure for process**

In the prog\_2, I have defined a type of page, which is shown below:



**Figure 2. data structure for memory page**

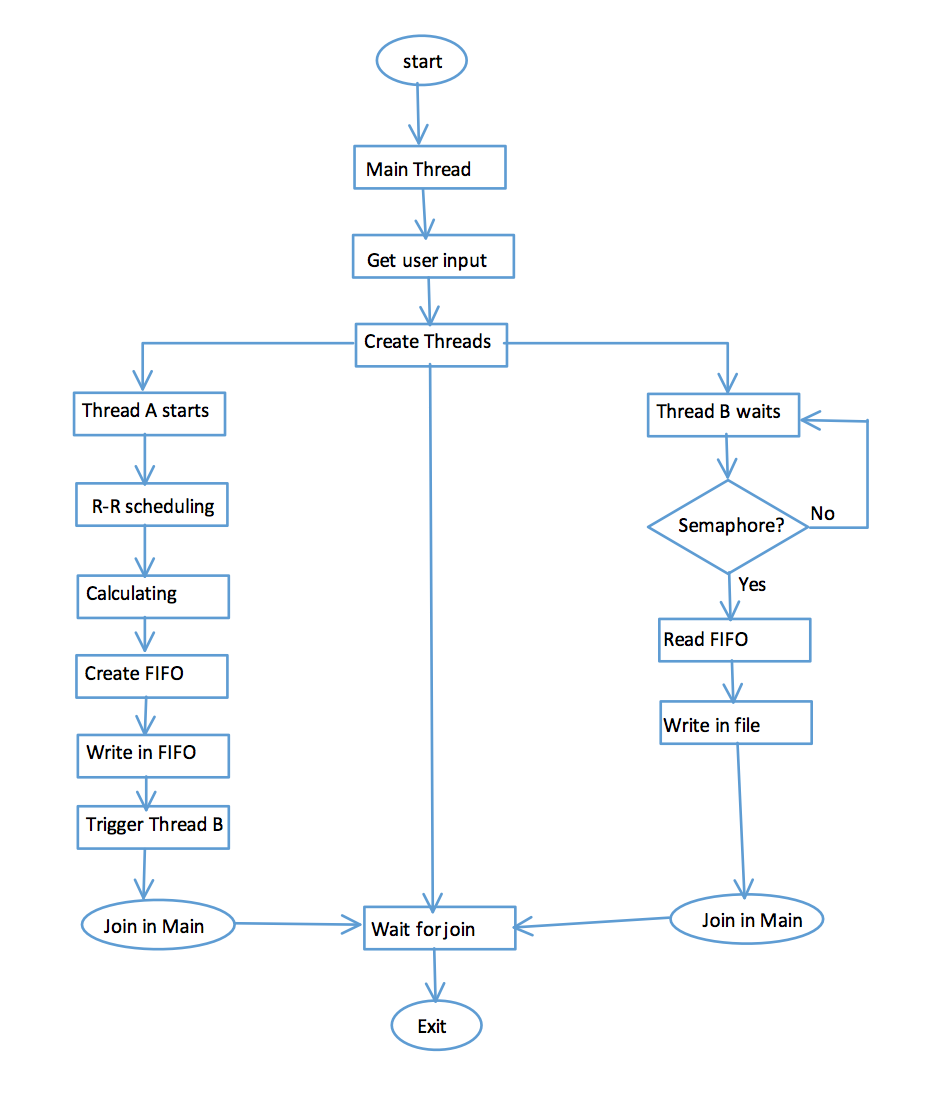
One of the techniques to achieve the First In First Out algorithm is the Queue which I mentioned in the previous section. Technically, I first implemented the queue using linked list, by which method the queue can be generic. Because, the type of data in a node can be any, for prog\_1 is process type, prog\_2 is page type as shown above. So with the support of generic queue, I can simply just enqueue() or dequeue() a process or memory page when its needed.



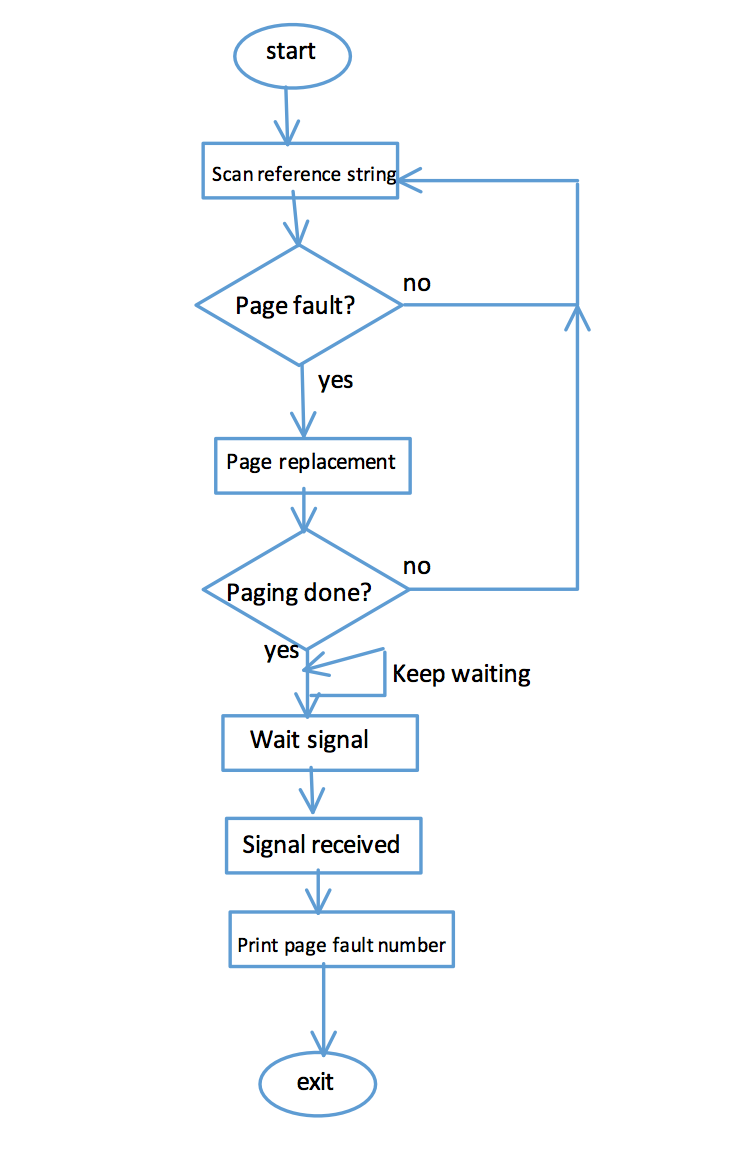
**Figure 3. type of data inside the NODE of the queue**

**Overall, with the help of well-defined data structure and generic queue, I can more concentrate on the actual logic of the algorithm itself. The description of the round robin and page replacement algorithm will be presented in the next section.**

1. Flow chart

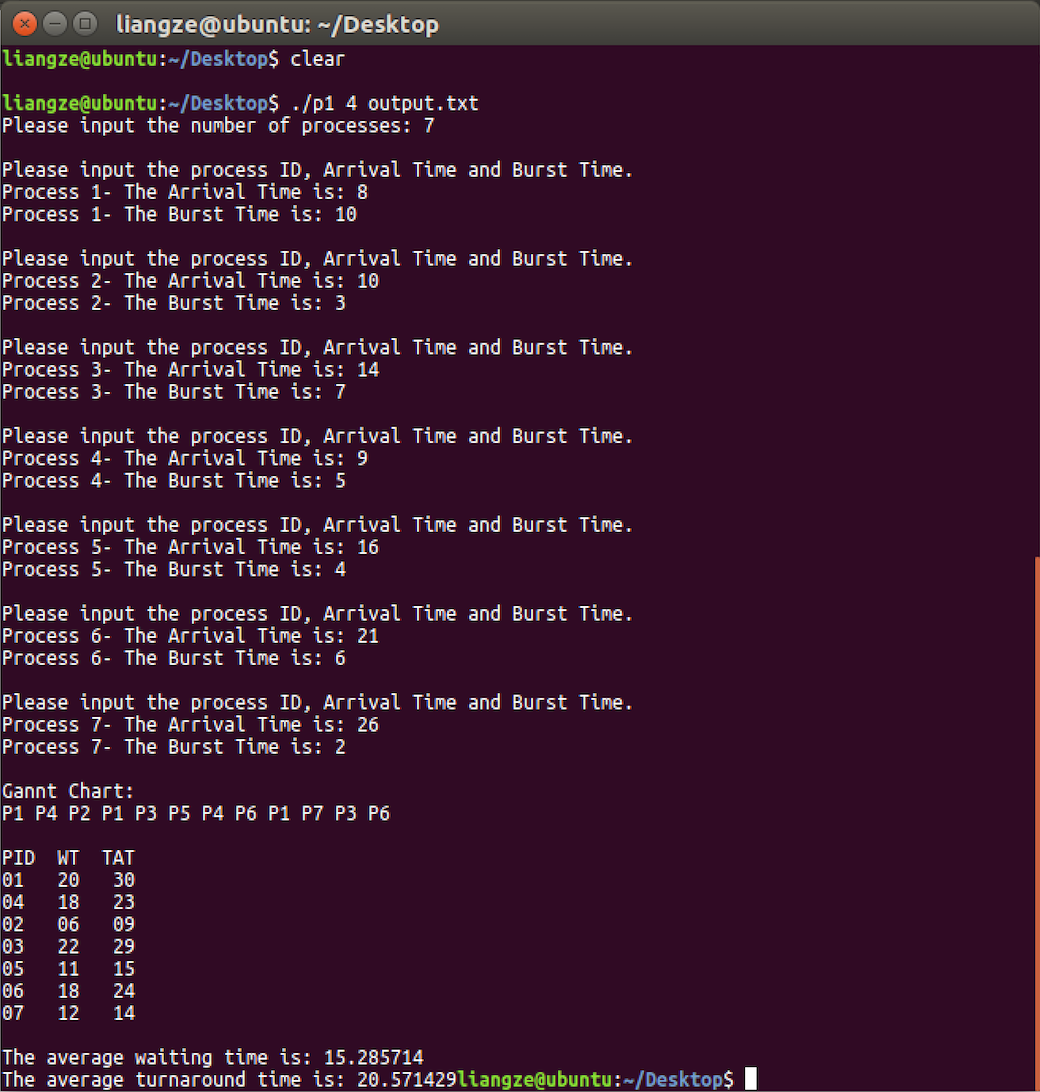


**Figure 4. Flow chart for prog\_1**

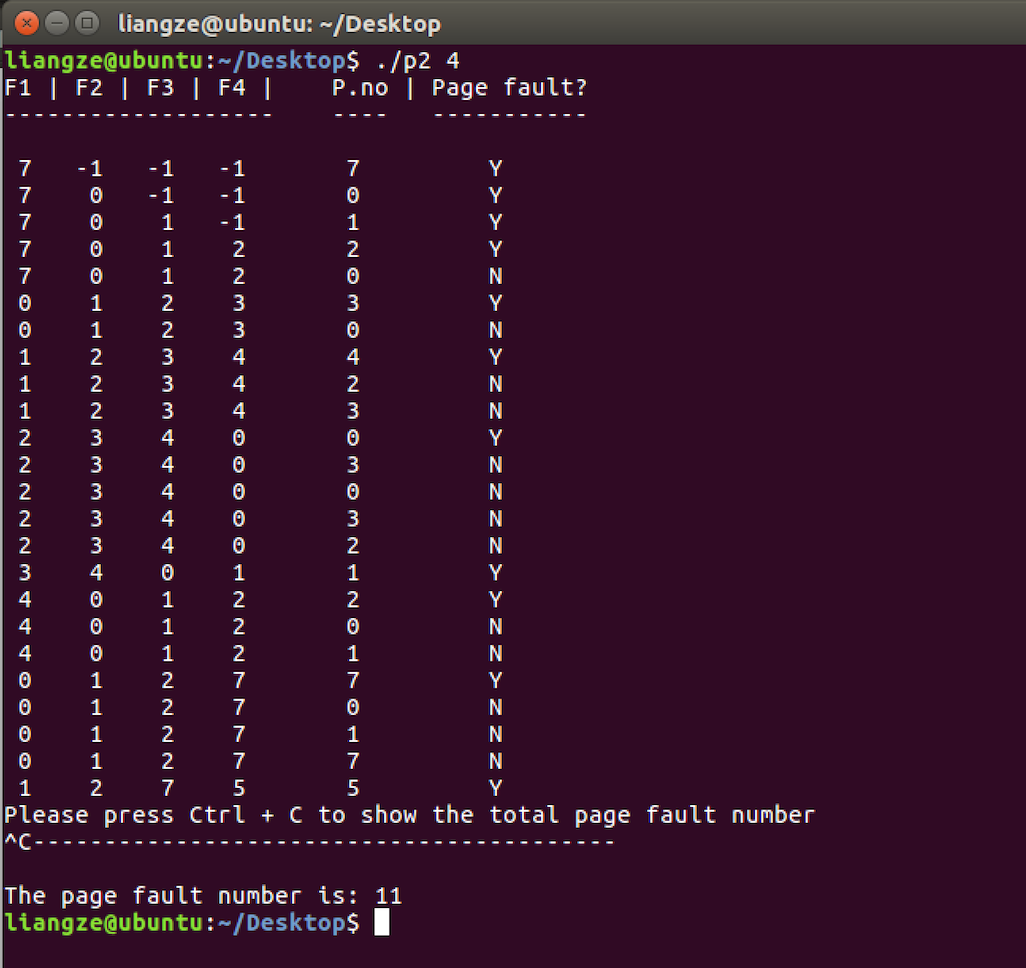
****

**Figure 5. Flow chart for prog\_2**

1. Experiments



**Figure 6. Output of Prog\_1**



**Figure 7. Output of prog\_2**

1. Conclusion on result analysis

**Prog\_1 Analysis**

One important fact about prog\_1 is that the results vary depending on the specific scheduling policy. In my implementation, I defined the following rules to resolve the scheduling issues:

1. If two or more new processes arrive at the same time stamp, they will go into the ready queue in the order of the process ID.
2. say process PA does not finish within the previous time quantum, then it is pre-empted from the cpu at time (*t*). At the same time, a new process process B arrives.

In this situation, that PA and PB both waiting to go into the ready queue, PB will go into the queue first. Because, PA which is the one just pre-empted from the CPU, will be context switched before going into the queue. As this result, PA will actually arrive into the ready queue slightly later than PB due to the time taken by context switch.

According to the above rules strictly, my program will output as shown above.

**Prog\_2 Analysis**

Also, note that the output of the prog\_2 will be slightly different from the lec slides in terms of the fram states. The reason why is that my 4 frames are organised as a queue, and each time a page in the queue is swapped out, the rest of the elements will shift accordingly, that is, previous 2nd element becomes the current 1st element in the queue, previous 3rd element becomes the current 2nd element in the queue. But the overall result is correct.

Overall, we can see that a scheduling algorithm purely depends on the FIFO is not sufficient. To be more efficient, we should also take other factors such as priority into account to make the algorithm more robust.