Scheduler Documentation

Xavier Guinto Rios A01366106
Gustavo Santamaría A01362484

Objective

Given a list of processes, the goal of this project is to implement a simple process scheduler capable of simulating the behavior of the following scheduling:

- **First Come First Served (FCFS)**
- **Preemptive and Non-Preemptive Shortest Job First (SJF) & Priority**
- **Round Robin (RR)**

The simulator should produce as output the average waiting time and average response time for all six scheduling algorithms described above.

Goals

The implementation of scheduling algorithm will test the comprehension of the scheduling algorithms, as well as verify that the student is capable of using basic dynamic data structures, file management,and general programming methodologies.

Problem Statement

We are required to implement a process scheduler capable of implementing the following algorithms:

- First Come First Served (FCFS)
- Non-Preemptive Shortest Job First (SJF)
- Non-Preemptive Priority
- Preemptive Shortest Job First (SJF)
- Preemptive Priority
- Round Robin (RR)

Our Program will take as input a *text file* with an arbitrary number of processes. Each line in the file specifies the **Priority**, **CPU Burst**, and **Arrival Time** for a process. For Round-Robin the **quantum** will be specified as a number in the first line of the file (an integer number in the line). We can assume that the input text file is passed as a parameter to our program upon invocation. For example:

- *scheduler process_file.txt*

As mentioned before, the file specifies one process per line with the following format: process_id, arrival_rime, CPU_burst, priority.

If the line starts with the # symbol then We must assume that the line is a comment and We simply ignore it. As an example, We are going to take the following table with information about five processes and *quantum* of 3 unities of time.

Table 1. Example of five processes

Process	Arrival Time	CPU Burst	Priority
P1	0	3	3
P2	2	3	0
P3	5	6	2
P4	6	2	1
P5	6	7	3

Then the text file describing the above table would be:

3
1 0 3 3
2 2 3 0
3 5 6 2
4 6 2 1
5 6 7 3

If there are gaps between the time a process finishes execution and another process starting execution that means that the CPU was idle and this will have an effect on the average waiting time and average response time.

Expected Outputs

The program should output the average waiting time and average response time for each of the six scheduling algorithms in the following order: First Come First Served (FCFS), Non-Preemptive Shortest Job First (SJF), Non-Preemptive Priority, Preemptive Shortest Job First (SJF), Preemptive Priority, and Round Robin. Use 32-bit floating point variables with four digits of precision when you display your results.

Consider the processes described in Table 1. The Expected Results Are:

Algorithm	**Average Wait Time**	**Average Response Time**	**Number of Process**
FCFS	3.2000	7.0000	5
Non-Preemptive Priority	2.4000	6.2000	5
Non-Preemptive SJF	2.4000	6.2000	5
Preemptive Priority	4.2000	5.4000	5
Preemptive SJF	2.4000	6.2000	5
Round Robin	3.6000	5.8000	5

Problem Statement

Taking previous knowledge of the class Data Structure we use a doubly-linked code to implement the running and waiting process list with the library GLib. But before start to code we did some example exercises with every algorithm with the purpose of understanding completely how it works and how to get the average time that is the goal of this project.

Once we obtain the information of the text file and save that information in the double-linked list we ordered in different ways for every algorithm so we can implement and resolve the average time of it. We take some precaution at the moment of reading the text file to ignore some character as a comments so we can take the real data that we will use that are the quantum, the number of process, arrival time, cpu burst, and priority. In case that there was a mistake reading the file we let you know with a print in the system.

So once we got that information our program start to run each algorithm one at a time. The nodes and pointers take a very important role because with them we do some comparison that help us to order the list by arrival time, priority, cpu burst for example. So in other file of the project we took that order list with their respective algorithm.

How To Build Executable Of The Program

Using a Linux environment of **64-bits**, to build the executable of the program is required the use of the terminal, also called *command line*.

The prerequisites are:

1. Install C/C++ compiler and related tools

Though these are already installed as gcc (GNU Compiler Collection), it is not bad to update it and verify it is installed.
You can use the following commands to check the updates:

- $ sudo apt-get update
- $ sudo apt-get install build-essential manpages-dev

2. Install The Glib Library

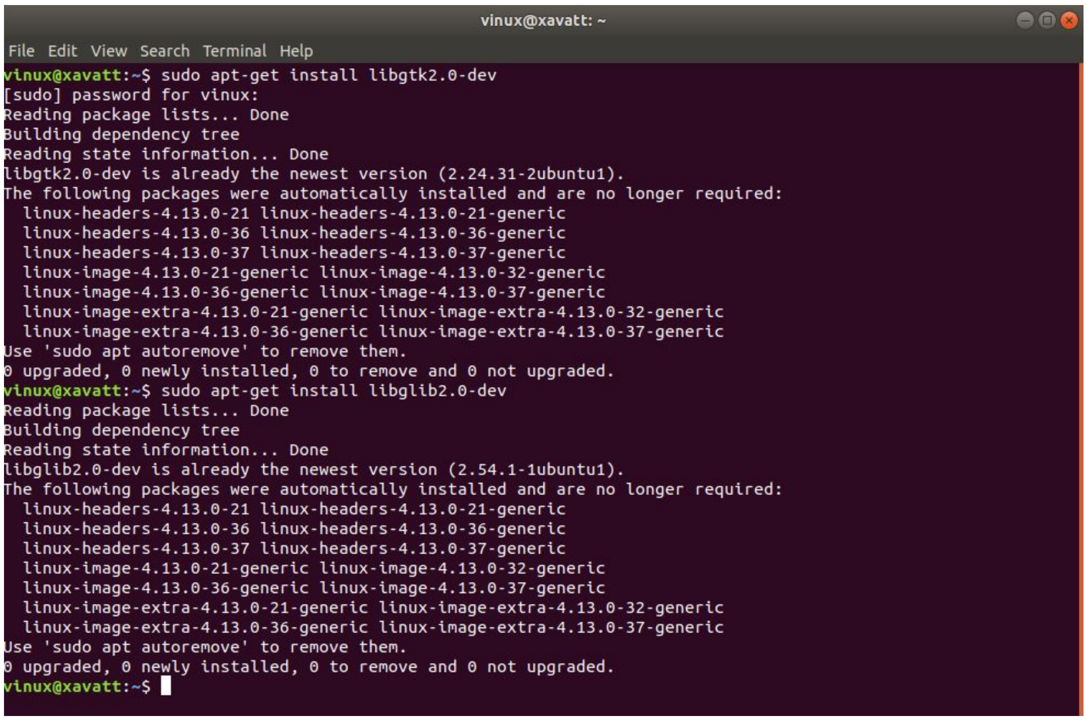
Since we are using the general-purpose library Glib, It is necessary to install it. According to the Glib documentation, Glib may make use of some GTK + dependencies, therefore, in order to install glib completely it is necessary to install GTK+.
GTK+ is a multi-platform toolkit for creating graphical user interfaces and a package that contains the shared libraries.
It is better to install GTK through the following command:

- $ sudo apt-get install libgtk2.0-dev

And verify Glib is installed through the next command:

- $ sudo apt-get install libglib2.0-dev

Here are a ScreenShot to make it more visible.



```
vinux@xavatt:~$ sudo apt-get install libgtk2.0-dev
[sudo] password for vinux:
Reading package lists... Done
Building dependency tree
Reading state information... Done
libgtk2.0-dev is already the newest version (2.24.31-2ubuntu1).
The following packages were automatically installed and are no longer required:
  linux-headers-4.13.0-21 linux-headers-4.13.0-21-generic
  linux-headers-4.13.0-36 linux-headers-4.13.0-36-generic
  linux-headers-4.13.0-37 linux-headers-4.13.0-37-generic
  linux-image-4.13.0-21-generic linux-image-4.13.0-32-generic
  linux-image-4.13.0-36-generic linux-image-4.13.0-37-generic
  linux-image-extra-4.13.0-21-generic linux-image-extra-4.13.0-32-generic
  linux-image-extra-4.13.0-36-generic linux-image-extra-4.13.0-37-generic
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
vinux@xavatt:~$ sudo apt-get install libglib2.0-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
libglib2.0-dev is already the newest version (2.54.1-1ubuntu1).
The following packages were automatically installed and are no longer required:
  linux-headers-4.13.0-21 linux-headers-4.13.0-21-generic
  linux-headers-4.13.0-36 linux-headers-4.13.0-36-generic
  linux-headers-4.13.0-37 linux-headers-4.13.0-37-generic
  linux-image-4.13.0-21-generic linux-image-4.13.0-32-generic
  linux-image-4.13.0-36-generic linux-image-4.13.0-37-generic
  linux-image-extra-4.13.0-21-generic linux-image-extra-4.13.0-32-generic
  linux-image-extra-4.13.0-36-generic linux-image-extra-4.13.0-37-generic
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
vinux@xavatt:~$
```

Finally, **To compile** the executable Schedler the following command is required:

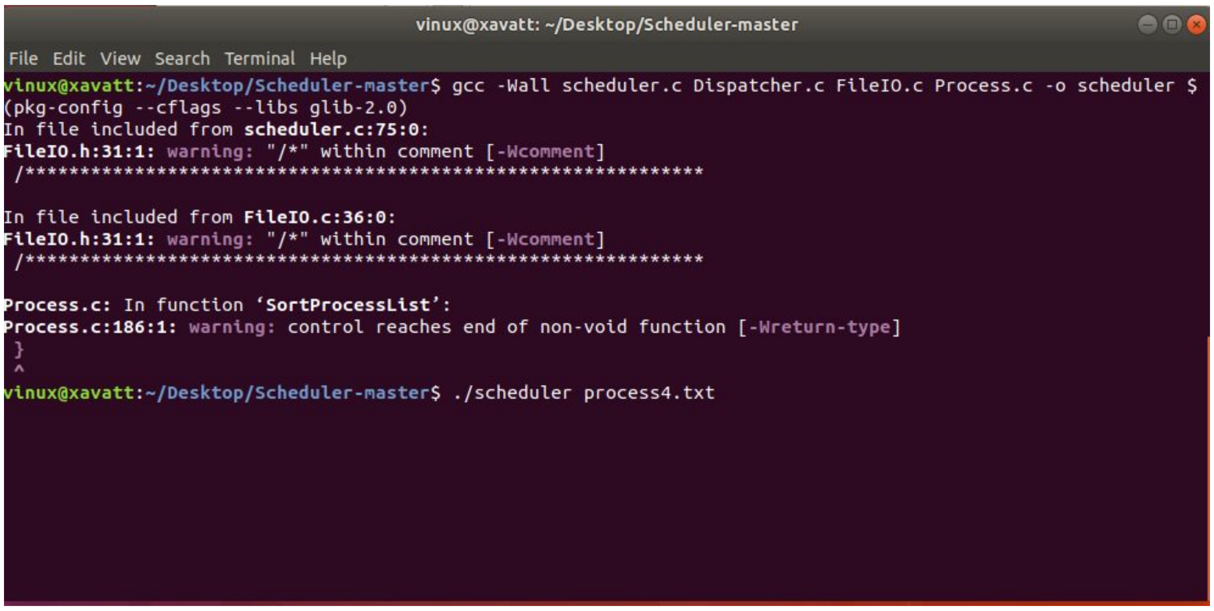
- gcc -Wall Scheduler.c Dispatcher.c FileIO.c Process.c -o scheduler $(pkg-config --cflags --libs glib-2.0)

Explication of the command

- gcc : Is the command to invoke gcc compiler.
- -Wall : Enables all compiler warning messages. (This command is optional)
- Scheduler.c Dispatcher.c FileIO.c Process.c : To compile the program from multiple source files.
- -o : It will define the output file with the following name:
    - scheduler : In this case, the name of the output file.
- pkg-config : Search in the package configuration for an easier compilation.
- --cflags : Are the name of environment variables for the C compiler.
- --libs[*library*] : To fetch required include, lib flags, and it's dependencies for the followed *library*. This assumes *library-dev* package is already installed.
- [**glib-2.0**] : The library used in this case to compile along with the program.

To run the executable just created, the next command is applied:
- Directory/folder$ ./scheduler process4.txt

Here are a ScreenShot to make it more visible.



```
vinux@xavatt:~/Desktop/Scheduler-master$ gcc -Wall scheduler.c Dispatcher.c FileIO.c Process.c -o scheduler $
(pkg-config --cflags --libs glib-2.0)
In file included from scheduler.c:75:0:
FileIO.h:31:1: warning: "/*" within comment [-Wcomment]
 /*****************************************************************

In file included from FileIO.c:36:0:
FileIO.h:31:1: warning: "/*" within comment [-Wcomment]
 /*****************************************************************

Process.c: In function ‘SortProcessList’:
Process.c:186:1: warning: control reaches end of non-void function [-Wreturn-type]
 }
 ^
vinux@xavatt:~/Desktop/Scheduler-master$ ./scheduler process4.txt
```

NOTE: Since the source code is in several files, the files MUST be in the same directory