University of Tunis El Manar
Higher Institute of Computer Science
December 2023

Advanced Operating Systems Mini-Project Development of a Process Scheduler Application

Members :
BARGHOUTI Yosr
KHARROUBI Wajdi
BENAMMOU Marwen
AJLANI Oussama

Group : 1 ING 1

Academic year : 2023-2024

Table of contents

Introduction

This project presents the implementation of diverse multitasking scheduling algorithms, offering a visual means to understand and compare their outcomes. The accompanying report provides a comprehensive overview of the implementation details, sprint planning and product backlogs to carry on the execution, alongside a user-friendly guide for installation and usage.

1. Conception

1.1. Data structures

- **Process data structure:**

- A group of processes is defined as a linked list where the data fields for each process describe the following:
    - Name of process
    - Time of arrival
    - Static priority
    - Execution Time
    - Remaining Time
    - Waiting time
    - Status
- Functions relating to groups of processes include the following:
    - Creating linked list from a text file containing the PCB
    - Sorting by date of arrival, by priority, by execution time
    - Displaying the list of processes to the end user
    - Indicating whether a process/all processes are done with execution
    - Comparing between a process' date of arrival and current CPU time.

- **Ready queue data structure:**

- The queue is needed in every scheduling algorithm to store processes that request CPU time when it's currently being used by another process.
- Each scheduling algorithm relies on a specific criteria to select the upcoming process to execute (e.g. priority, remaining execution time, waiting time in the queue)
- Functions relating to the ready queue include the following:
    - Enqueuing and dequeuing a process from the queue
    - Parsing the queue
    - Selecting a process node with properties that satisfy the scheduling algorithms criteria
    - Displaying the queue elements at a given time
    - Indicating the CPU is idle in case the ready queue is empty

1.2. Scheduling algorithms:

- **FIFO (First In First Out):**
    - **Processes** are executed in the order they arrive.
    - Once a process starts, it runs to completion without interruption.
    - The next process in line becomes the new head of the **ready queue** after the current one finishes.
- **Round Robin:**

    - Each **process** is given a fixed time slice (also called **quantum**) to execute on the CPU.
    - If a process doesn't finish within its time slice, it goes to the back of the **ready queue**.
    - The scheduler moves to the next process in the queue, ensuring fair execution for all processes.
- **Shortest remaining time:**
    - The **process** with the smallest remaining execution time is given priority.
    - If a new process arrives with a shorter execution time than the currently running one, it preempts and takes the CPU.
    - This ensures efficient utilization by minimizing the time each process needs to complete.
- **Preemptive priority**
    - Processes are assigned priorities, and the one with the highest priority currently in the ready queue gets the CPU.
    - If a higher-priority process arrives, it **preempts** the running process.
    - The process with the highest priority is always chosen for execution, promoting efficient handling of critical tasks
- **Multilevel**:
    - **Processes** are divided into multiple **queues** based on **priority** levels.
    - Each **queue** can have its own scheduling algorithm, like FIFO or Round Robin.

2. Implementation

2.1. Software environment and tools

Ubuntu, Kali and **Windows Subsystem Linux** as Linux environments.
VMware as a virtualization tool.
GNU GCC as C compiler.
GNU Make as a build tool.
VSCode as text editor.
GTK (GIMP Toolkit**)** as a widget toolkit for creating graphical user interfaces
Glade on Linux for GUI design.
Git as a version control system.

2.2. Repository structure

📂**process-scheduler**
-📂**algos**:
    - basicfifo.c
    - multilevel.c
    - premptivepriority.c
    - roundrobin.c
    - Srt.c
-main.c
-misc.h
-pcb.txt
-Makefile
-prototype.glade

`main.c`: source file

- Handles GUI logic with the help of GTK widget library
- Ensures dynamic menu rendering by parsing generated shared object files from the `/algos` directory.
  - `void getSOFiles(const char *directory, char ***soFiles, int *numFiles);` Function that parses the shared object files in the /algos directory
  - `typedef void (*SchedulingAlgorithm)(processus *);` Type referring to an algorithm scheduling function
  - `SchedulingAlgorithm loadSchedulingAlgorithm(const char *algorithmName)` Function that loads the scheduling algorithm function from the shared object files

`prototype.glade`:

- GUI description file written in XML and generated by Glade.
- Defines the structure and layout of graphical user interfaces for applications.

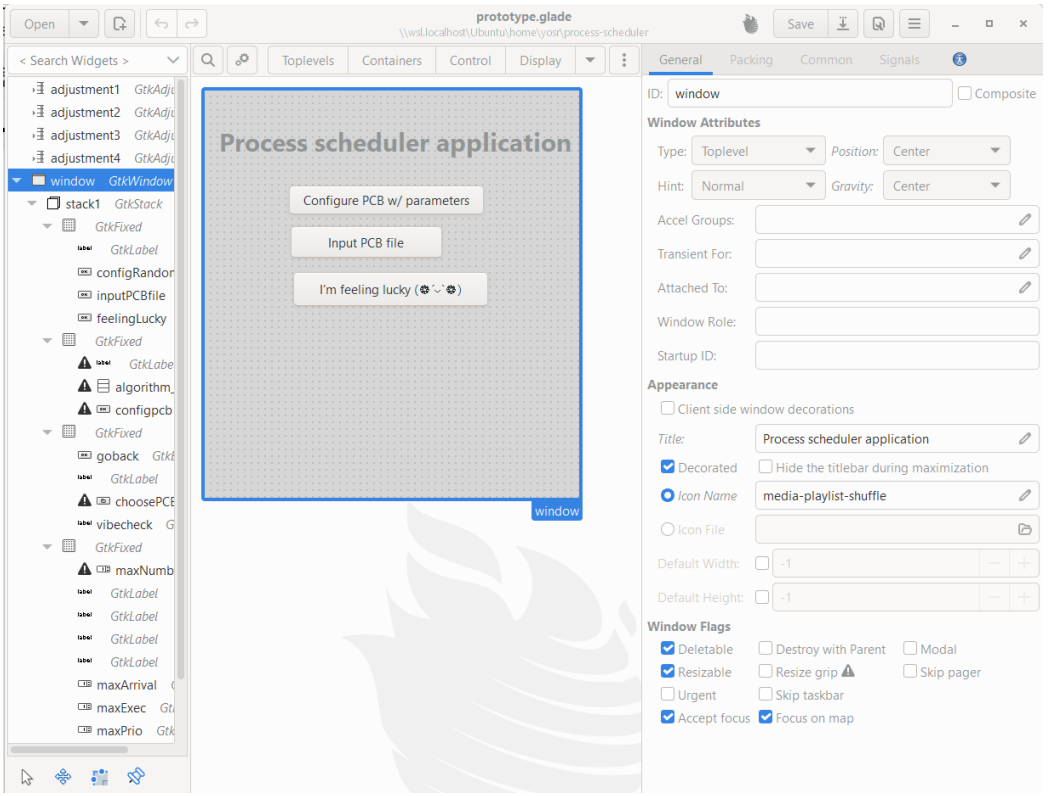

Figure: Glade software for building GUIs

`pcb.txt`: main text file containing the process control block.

```
name,tA,tE,Prio
P0;2;7;3
P1;2;9;4
```

Figure: pcb.txt snippet

`misc.h`: header file

- Handles **text file manipulation** by generating a process linked list from pcb.txt
  - `typedef struct` **`processus`** `:` structure representing a process
  - `processus *`**`enreg_pcb`**`(FILE *file):` function that generates a `processus` linked list from the pcb.txt file
  - `void` **`generateFileParam`**`(int maxNumber, int maxArrival, int maxExec, int maxPrio)` : enables configuration of pcb.txt with parameters provided by the user
  - `bool` **`verifFile`**`():` checks if a submitted file is empty

-Handles **display** operations for processes
  - `typedef struct` **`viewProcess`**`:` structure suited for Gant diagram display
  - `void` **`displayTab`**`(processus *tab):` display PCB in tabular form
  - `processus` **`*sortProcesses`**`(processus *head)` : sort processes by date of arrival
  - `viewProcess *`**`inseretProcess`**`(viewProcess *head, viewProcess *x)` used for building viewProcess linked lists
  - `void` **`GantAndStatistic`**`(viewProcess *view):` generates Gant diagram after scheduling and provides performance metrics

- Handles **ready queue operations** (creation, enquing, dequeing, display, sorting)
  - `typedef struct` **`QueueNode`** `:` structure containing a process
  - `typedef struct` **`Queue`**`:` linked list of QueueNode
  - `Queue *`**`createQueue`**`()`
  - `void` **`enqueue`**`(Queue *queue, processus *process)`
  - `processus *`**`dequeue`**`(Queue *queue)`
  - `void` **`stateOfQueue`**`(Queue *queue)` : prints out the nodes in queue with information like priority and remaining execution time
  - `void` **`sortByPriorityQueue`**`(Queue *queue)` : sorts adjacent nodes
  - `void` **`sortByLastWait`**`(Queue *queue)`
  - `void` **`sortByDurExecModifProcQueue`**`(Queue *queue)`

📂**`/algos.*.c`**`: source files for scheduling algorithms`
- Each source file includes `misc.h` header file
- Each file contains a single function with this signature:

  `void` *`algorithm_name`*`(processus *head)`

- Executing this function generates the scheduling output on the shell for a provided process linked list.

`Makefile: file used by the GNU Make build system`
- **Source Files and Libraries:**
  - `SRC_DIR` specifies the directory for algorithm source files.
  - `CFLAGS` defines compiler flags with include directory information.
  - `SRC_FILES` gathers all C files in the `/algos` directory.
  - `LIB_FILES` generates shared object (.so) files from source files.
- **Main Program:**
  - `MAIN` sets the main program's name as `process-scheduler`.
  - Compiles `main.c` and links necessary libraries, including GTK and dynamic

loading.

- **Library Generation:**
  - `libs` rule compiles source files into shared objects.
  - `$(LIB_DIR)/%.so` generates shared objects with header file `misc.h`.
- **Installation and Execution:**
  - `INSTALL_DIR` specifies the installation directory for the main program.
  - `LIB_DIR` defines the directory for storing generated dynamic libraries.
  - `run` rule creates necessary directories, copies files, and executes the main program.
  - `install` copies files to appropriate directories, ensuring proper installation.
- **Cleanup and Uninstallation:**
  - `clean` rule removes generated files (libraries and the main program).
  - `uninstall` removes the installed main program and associated directories and files.

2.3. Algorithms' pseudo code

NB: ***viewProcess*** are structures used in Gant Diagram display and are periodically updated with processes' information at the time of their execution.

2.3.1. First in first out

0. Initialize an empty viewProcess linked list (view)
1. Initialize a Queue data structure (processQueue)
2. Sort the input processes by arrival time (sortedProcesses)
3. Set current to the first process in sortedProcesses
4. Initialize simulation time (time) to 0
5. **While there are processes in sortedProcesses or the processQueue is not empty:**
   5.1. Enqueue processes from sortedProcesses whose arrival time is <= to the current time
   5.2 Print the content of processQueue
   5.3. Dequeue the front process from the processQueue (executingProcess)
   5.4.a. If executingProcess is not NULL:
      5.4.a.1. Print the execution information for executingProcess
      5.4.a.2. Create a viewProcess node (i) for the Gantt chart and
      5.4.a.3. Update the node with information from executingProcess
      5.4.a.4. Update time based on the execution time of executingProcess
   5.4.b If executingProcess is NULL:
      5.4.b.1. Print "CPU idle"
      5.4.b.2. Increment time
6. Generate Gantt chart and statistics based on the viewProcess linked list

2.3.2. RoundRobin

0. Initialize an empty viewProcess linked list (view)
1. Initialize a Queue data structure (readyQueue)
2. Sort the input processes by arrival time (sortedProcesses)
3. Set current to the first process in sortedProcesses
4. Initialize simulation time (time) to 0
5. Prompt the user for the quantum value and validate it
6. **While there are processes in sortedProcesses or the readyQueue is not empty:**
    6.1. Enqueue processes from sortedProcesses whose arrival time is <= to the current time
    6.2. Print the state of the readyQueue
    6.3. Sort the Queue by last Waiting time for each process **
    6.4. Dequeue the front process from the readyQueue (executingProcess) **
    6.5.a If executingProcess is not NULL:
        6.5.a.a. If the remaining time of executingProcess is less than the quantum:
            6.5.a.a.1 Print execution information for executingProcess
            6.5.a.a.2 Create a viewProcess node (i) for the Gantt chart
            6.5.a.a.3 Update view with information from executingProcess
            6.5.a.a.4 Update time based on the execution time of executingProcess
            6.5.a.a.4 Set the remaining time of executingProcess to 0
        6.5.a.b. Else:
            6.5.a.b.1 Print execution information for executingProcess with quantum units
            6.5.a.b.2 Create a viewProcess node (i) for the Gantt chart
            6.5.a.b.3 Update view with information from executingProcess
            6.5.a.b.4 Update time based on the quantum
            6.5.a.b.5 Decrement the remaining time of executingProcess by the quantum
            6.5.a.b.6 Update the **last wait time** of executingProcess

        6.5.a.a. If executingProcess has completed execution:
            6.4.a.a.1. Print "Process is done with execution"
        6.5.a.b. Else:
            6.5.a.b.1. Print "Process still has units remaining and is added to the queue"
            6.5.a.b.2. Enqueue executingProcess back to the readyQueue

    6.5.b. If executingProcess is NULL:
        6.5.a. Print "CPU idle"
        6.5.b. Increment time

7. Generate Gantt chart and statistics based on the viewProcess linked list

---

NB: 6.3 and 6.4 before dequeuing the front process, a queue sort function iteratively compares **adjacent** processes in the queue based on their **last_wait** times and swaps them if they are out of order.
This favors the processes that have been waiting the most and ensures a fair execution, in accordance with the round-robin scheduling algorithm.
NB: At the start, the last_wait_time is the process' date of arrival.

2.3.3. Shortest remaining time:

0. Initialize an empty viewProcess linked list (viewProcess)
1. Initialize a Queue data structure (readyQueue)
2. Sort the processes by arrival time (sortedProcesses)
3. Set **current** to the first process in sortedProcesses
4. Initialize simulation time (time) to 0
6. While there are processes in sortedProcesses or the readyQueue is not empty:
   6.1. Enqueue processes from sortedProcesses whose arrival time is <= to the current time
   6.2. Print the state of the readyQueue
   6.3. Sort the Queue by increasing the remaining time for each process
   6.4. Dequeue the front process from the readyQueue (executingProcess)
   6.5.a. If executingProcess is not NULL:
      6.5.a.1 Create a viewProcess node (i) for the Gantt chart
      6.5.a.2 Update view with information from executingProcess
      6.5.a.3 Decrement the process remaining time
      6.5.a.4 Increment time
      6.5.a.5.a If executingProcess has completed execution:
         6.5.a.5.a.1. Print "Process is done with execution"
      6.5.a.5.b Else:
         6.5.a.5.b.1. Enqueue executingProcess back to the readyQueue

   6.5.b. If executingProcess is NULL:
      6.5.b.1. Print "CPU idle"
      6.5.b.2. Increment time
7. Generate Gantt chart and statistics based on the viewProcess linked list

---

NB: 6.3 and 6.4 before dequeuing the front process, a queue sort function iteratively compares **adjacent** processes in the queue based on their **remaining execution** times and swaps them if they are out of order.
This favors the processes that have the least remaining units to execute, in accordance with the SRT scheduling algorithm.
NB: At start, the remainingExecutionTime is the process' time of execution.

2.3.4. Pre-emptive priority

0. Initialize an empty viewProcess linked list (viewProcess)
1. Initialize a Queue data structure (readyQueue)
2. Sort the processes by arrival time (sortedProcesses)
3. Set current to the first process in sortedProcesses
4. Initialize simulation time (time) to 0
6. While there are processes in sortedProcesses or the readyQueue is not empty:
   6.1. Enqueue processes from sortedProcesses whose arrival time is <= to the current time
   6.2. Print the state of the readyQueue
   6.3. Sort the Queue by decreasing priority for each process
   6.4. Dequeue the front process from the readyQueue (executingProcess)
   6.5.a. If executingProcess is not NULL:
      6.5.a.1 Create a viewProcess node (i) for the Gantt chart
      6.5.a.2 Update view with information from executingProcess
      6.5.a.3 Decrement the process remaining time
      6.5.a.4 Increment time
      6.5.a.5.a If executingProcess has completed execution:
         6.5.a.5.a.1. Print "Process is done with execution"
      6.5.a.5.b Else:
         6.5.a.5.b.1. Enqueue executingProcess back to the readyQueue

   6.5.b. If executingProcess is NULL:
      6.5.b.1. Print "CPU idle"
      6.5.b.2. Increment time
7. Generate Gantt chart and statistics based on the viewProcess linked list

---

NB: 6.3 and 6.4 before dequeuing the front process, a queue sort function iteratively compares **adjacent** processes in the queue based on their **priority** and swaps them if they are out of order.
This favors the processes that have higher priority and enables them to preempt the current process by placing them on top of the ready queue, in accordance with the preemptive priority scheduling algorithm.

2.3.5. Multilevel

0. Initialize an empty **viewProcess** linked list (view)
1. Initialize an empty **viewProcess** linked list (q)
1. Count the number of processes in the list (countP)
2. Convert the linked list of processes to an array of Process structures (processTable)
3. Sort the processes by arrival time (processTable)
4. Prompt the user for the quantum value and validate it (quantum)
5. Initialize simulation time to 0 (time)
6. Initialize a variable executedTime to 0 (executedTime)
6. Initialize variables for highestPriorityPosition and highestPriority to -1 and 0, respectively
7. While not all processes are completed in processTable:
    7.1 Reset highestPriorityPosition to -1
    7.2 For each process in processTable:
        7.2.a. If the process is not completed, arrived, and has a higher priority than highestPriority:
            7.2.a.1. Update highestPriorityPosition and highestPriority

    7.3.a If highestPriorityPosition is still -1:
        7.3.a.1 **For** each process in processTable:
            7.3.a.1.a. If the process is not completed, arrived, and has the highest priority:
                7.3.a.1.a.1 Update highestPriorityPosition and ***break***

    7.3.b If highestPriorityPosition is not -1:
        7.3.b.1. Print a message indicating the execution of the process
        7.3.b.2.a If q is NULL or the code of q is different from the executing process:
            7.3.b.2.a.1 Create a viewProcess node (i) with information from the process
            7.3.b.2.a.2 Add i to the view list
            7.3.b.2.a.3 Update q to point to i

        7.3.b.2.b. Else:
            7.3.b.2.b.1 Increment the end time of the existing viewProcess node (q)
            7.3.b.2.b.2. Decrement the remaining execution time of the process

        7.3.b.3. If the process has completed its execution:
            7.3.b.3.1 Print a message indicating the completion of the process
            7.3.b.3.2 Reset highestPriority to 0.
        7.3.b.4. *If the process's priority is still the highestPriority and the executed time is equal to the quantum and is not finished yet:*
            7.3.b.4.1. *Rearrange the processTable to move the process to the end of the processTable*

    7.3.c. Else:
        7.3.c.1. Print a message indicating CPU is idle

    7.4. Increment simulation time (time)

8. Generate a Gantt chart and statistics based on the viewProcess linked list

3. Scrum planning

3.1. Global use case diagram

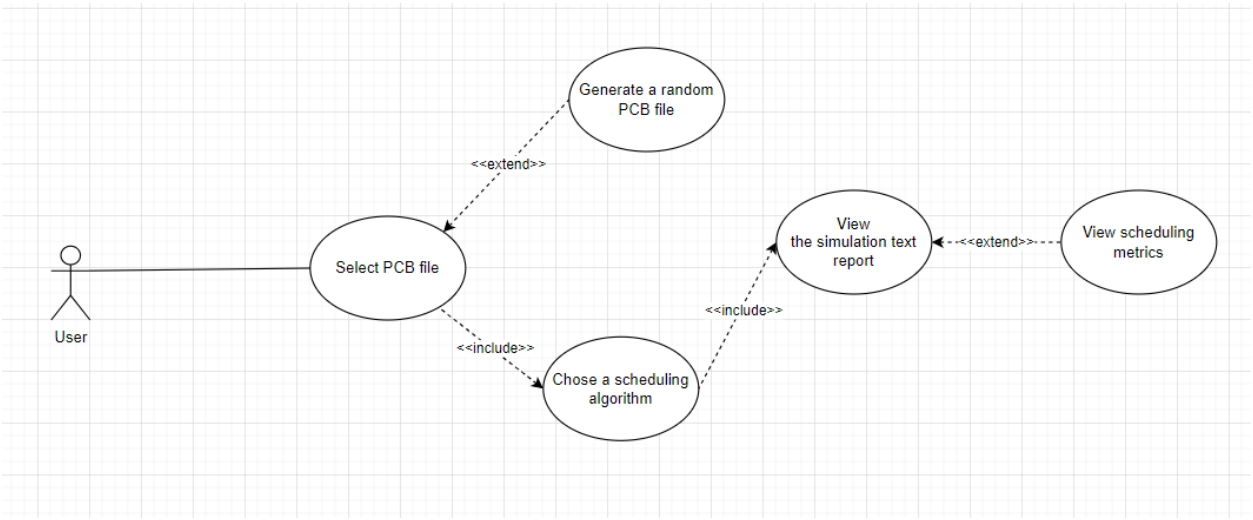

Figure: Global use case diagram

3.2 Product Backlog

The Product Backlog is the list of priority functionalities to be implemented by the team. Initially, Product Backlog items include preparation of the development environment, text file manipulation, file system configuration, scheduling algorithms, metrics and GUI design.

Id Spot	Sprint	Stains	Priority
1.1	1	Preparing the working environment	High
1.2	1	Review of Linux and C concepts	High
1.3	1	Division of the project into major parts	High
1.4	1	Text file handling	High
1.5	1	Data structures and algorithms	High
1.6	1	User interface design with GTK and Glade	Average
2.1	2	Command line menu display	High
2.2	2	Process structure definition	High
2.3	2	Process queue management	High
3.1	3	Implementation of FIFO, RoundRobin, SRT and preemptive prority algorithms	High
3.2	3	Algorithm command line display	High
3.3	3	Validation of FIFO, RoundRobin, SRT, Preemptive Prority	Average

3.4	3	Metrics	Average
3.5	3	Shared libraries/dynamic links	Average
3.6	3	Creation of the first graphical interface	Average
4.1	4	Improved console output	Average
4.2	4	Implementing the Multilevel algorithm	High
4.3	4	File configuration with user input in GUI	Average
4.4	4	Finalizing metrics for algorithms	Average

Notes :

- Tasks have been prioritized according to their importance to the success of the project.
- The most important tasks are those that must be completed before the others can begin.
- Tasks not directly related to product functionality have been given medium priority.
- Tasks that are optional or can be carried out in different ways have been given low priority.

Possible adaptations:

This order book is a suggestion. It can be adapted to the specific needs of the project. For example, tasks can be grouped or divided according to their interdependence. Priorities can also be adjusted according to the needs of the team or users.

3.3 Sprint Backlogs

Sprint Backlogs detail the tasks specific to each sprint, with checks for each item to mark progress.

3.3.1. Sprint 1: Planning Sprint (Oct 25 -> Nov 7)

Id Spot	Stains	Priority	Estimate
1.1	Preparing the working environment	High	2 days
1.2	Review of Linux and C concepts	High	2 days
1.4	Text file handling	High	2 days
1.5	Data structures and algorithms	High	3 days
1.6	User interface design with Glade	Average	3 days

3.3.2. Sprint 2: (Nov 8 -> Nov 12)

Id Spot	Stains	Priority	Estimate
2.1	Command line menu display	High	2 days

2.2	Process structure definition	High	1 day
2.3	Process queue management	High	2 days

3.3.3. Sprint 3 (Nov 13 -> Nov 24)

Id Spot	**Stains**	**Priority**	**Estimate**
3.1	Implementation of FIFO, RoundRobin, SRT and preemptive prority algorithms	High	4 days
3.2	Algorithm command line display	High	2 days
3.3	Validation of FIFO, RoundRobin, SRT, Preemptive Prority	Average	1 day
3.4	Metrics	Average	1 day
3.5	Shared libraries/dynamic links	Average	1 day
3.6	Creation of the first graphical interface with GTK	Average	2 days

3.3.3. Sprint 4 - (Nov 25 -> Dec 5)

Id Spot	**Stain**	**Priority**	**Estimate**
4.1	Improved console output	Average	1 day
4.2	Implementing the Multilevel algorithm	High	4 days
4.3	File configuration with user input	Average	2 days
4.4	Finalizing metrics for algorithms	Average	1 day
4.5	Add comparisons between algorithms	Average	1 day

- These are rough estimates, which can be adjusted according to the specific needs of the team and the project. It is important to take into account the complexity of the tasks, the experience of the team and the resources available.

Sprints backlogs can be made even more detailed by adding additional information, such as :

- Dependencies between tasks
- Potential risks

- Expected deliverables

4. Installation and usage guide

4.1. Introduction

4.1.1 Purpose

This guide provides instructions on installing, configuring, and using the process scheduler application.

4.1.2 System Requirements

Before proceeding, ensure that your system meets the following requirements:

- **Operating System**: The application is compatible with any Linux environment (e.g Ubuntu, kali) or a Windows Subsystem for Linux (WSL).
- **C Compiler**: A C compiler like **GCC** is required to compile the source code.
- **Build System**: A build system like **make** is needed to automate the compilation process.
- **Graphics GTK libraries** should be available on the system to render GUIs.
- **Other Development Tools**: Tools like **pkg-config** are often used to query information about installed libraries.

4.2. Installing the application

4.2.1. Required packages:

To install the **gcc** compiler

```
$ sudo apt-get install gcc
```

To install the **make** build system

```
$ sudo apt-get install make
```

To install **GTK** libraries to render graphics and **pkg-config**

```
$ sudo apt-get install libgtk-3-0
$ sudo apt-get install libgtk-3-dev
$ sudo apt-get install pkg-config
```

4.2.2. Installing the application:

- Make sure your system is up to date

```
$ sudo apt-get update
$ sudo apt-get install update
```

- In order to install the application, you have to first use the **make** build system.
- This step provides the dependencies between the different source files with the help of compiler flags, and compilation options already specified in the Makefile.
- The first make rule generates shared object files from the source files located in the `/algos` directory.

```
your@machine:/path/to/process-scheduler$ make
```

- To generate the executable `process-scheduler` executable and run your application, type the following command:

```
your@machine:/path/to/process-scheduler$ make run
```

NB: this step runs the program.

- To install the application on your machine, type the following command.

```
your@machine:/path/to/process-scheduler$ make install
```

The expected output for the installation command should look like this:

```
sudo mkdir -p /usr/local/lib/algos
[sudo] password for your-username:
sudo cp prototype.glade /usr/local/lib/
```

NB: The installation requires copying files into system directories and would, thus require, administrator privileges.
Make sure to provide a correct password to carry on with the installation.

4.2.3. Using the Application

- After installation, you can run the code from any directory with the following command.

```
your@machine:/random/path$ process-scheduler
```

4.2.3.1. Homepage:

- A GUI is opened. The user decides whether to configure the PCB file , choose an existing file or generate the PCB file randomly.

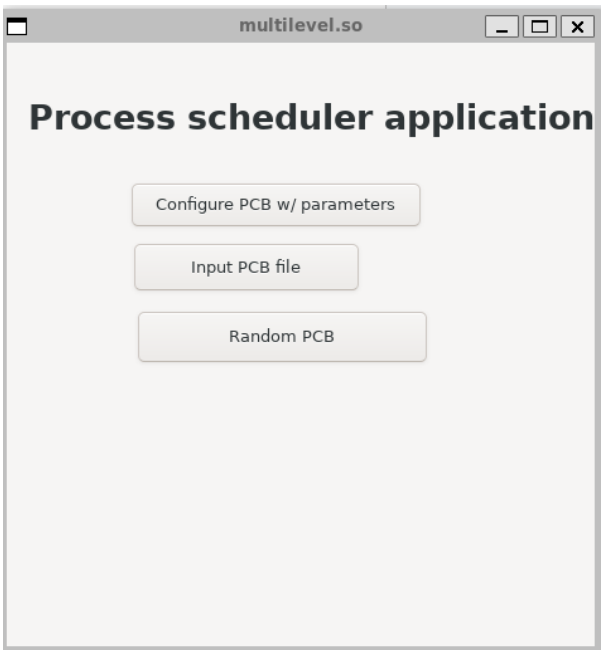

Figure: GUI for Homepage

A- Submitting a PCB text file

Upon clicking on on the “Input PCB file”, the user is redirected to the GUI for PCB file input

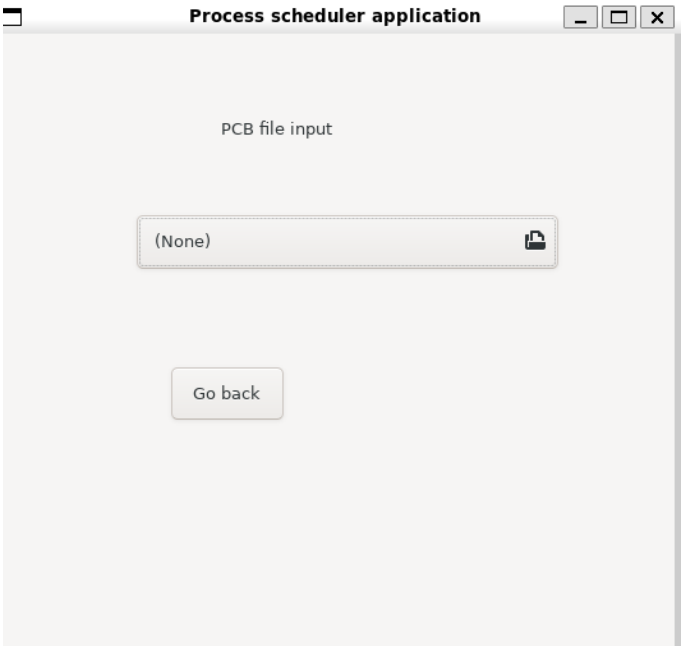

Figure: GUI for PCB file input

The user can use the file chooser widget and choose any text file.

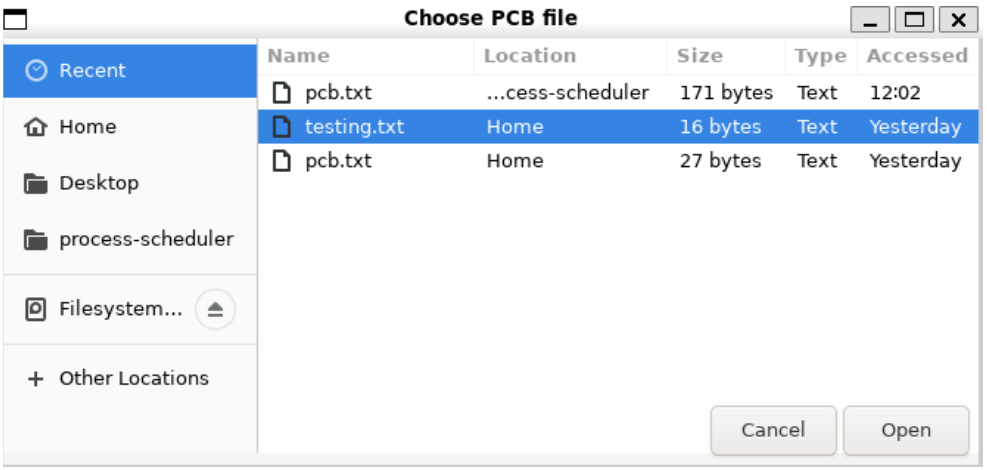

Figure: Text file selection

If the file is not empty, the user is redirected to the algorithm selection GUI.

If the file is empty, the following warning is shown.

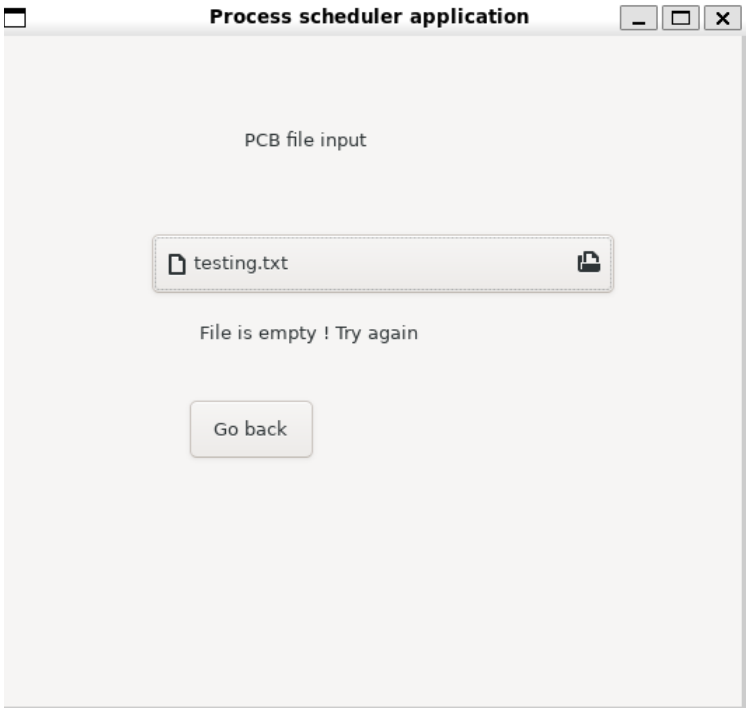

Figure: Empty text file warning

B- Configuring the PCB text file

Upon clicking on on the “Configure PCB w/ parameters”, the user is redirected to the GUI for configuring the PCB.

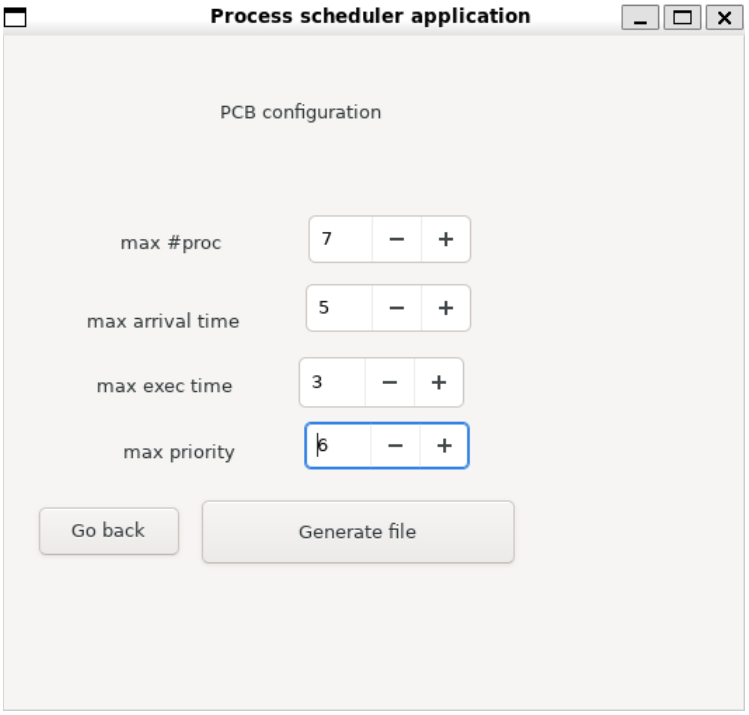

Figure: GUI for Configuring the PCB

Using the spin buttons, the user can define the maximum number of processes to be generated, the maximum arrival time, the maximum execution time and the maximum priority.

After clicking on the “Generate file button” , the shell outputs the generated PCB as follows.

8 processes generated

Process	Date Arr	Execution Time	Priority
p6	0	23	4
p2	0	16	5
p0	2	29	1
p1	4	4	4
p3	4	22	5
p4	4	8	1
p5	5	17	1
p7	7	24	2

Figure: Generated PCB after configuration.

The user is then redirected to the Algorithm choice menu.

C- Randomly generating a PCB text file

After clicking on the “Random PCB” , a PCB file is generated randomly and its content is displayed in the shell. Then, the user is redirected to the scheduling algorithm choice menu.

4.2.3.2. Algorithm choice menu

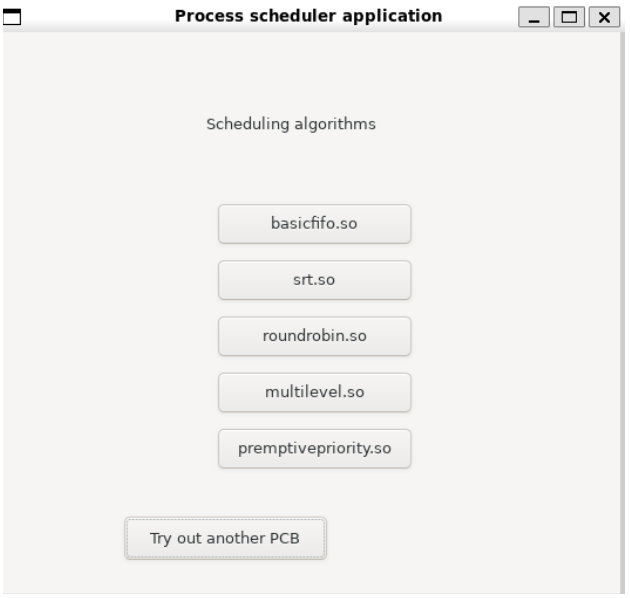

Figure: GUI for scheduling algorithm choice menu

Upon clicking on an algorithm button, the shell outputs the scheduling results.

Below is an example of the FIFO scheduling results pertaining to the PCB shown below.

Process	Date Arr	Execution Time	Priority
p3	1	3	4
p0	3	2	2
p2	3	2	3
p4	3	1	4
p1	5	3	4

Figure: Example PCB

Shell output	Walkthrough
FIFO Scheduling: t=0: CPU idle t=1 stateOfQueue: | p3 | t=1: executing process p3 for 3 units t=3 stateOfQueue: | p0 | t=3 stateOfQueue: | p0 | p2 | t=3 stateOfQueue: | p0 | p2 | p4 | t=4: executing process p0 for 2 units t=5 stateOfQueue: | p2 | p4 | p1 | t=6: executing process p2 for 2 units t=8: executing process p4 for 1 units t=9: executing process p1 for 3 units +----------------------+--------+--------+ | Process | Start | End | +----------------------+--------+--------+ | p3 | 1 | 4 | | p0 | 4 | 6 | | p2 | 6 | 8 | | p4 | 8 | 9 | | p1 | 9 | 12 | +----------------------+--------+--------+ p1: rotation time=7 ,waiting time=4 p4: rotation time=6 , waiting time=5 p2: rotation time=5 , waiting time=3 p0: rotation time=3 , waiting time=1 p3: rotation time=3 , waiting time=0 Average rotation time=4.80 Average waiting time=2.60	- When no process is using the CPU it is considered "`idle`" - `stateOfQueue` displays the ready queue upon each arrival of a new process. - The CPU chooses the process in the ready queue with the criteria that meet the scheduling algorithm - To recap the results, a table displays the start and end times for each process, as well as the rotation (total time spent in the CPU) and waiting (total time spent in the ready queue) times. - Further metrics include the average rotation time and average waiting time.

To test out other scheduling algorithms simply click on the corresponding button and the results will be displayed in the shell.

To try out another PCB, click on the "Try out another PCB" button.

4.2.4. Uninstalling the application

To uninstall the application, type the following command from inside the `process-scheduler` directory.

```
your@machine:/path/to/process-scheduler$ make uninstall
```

This removes the executable and shared object files from the system directories. (`usr/local/lib` and `usr/local/bin`).