École Internationale des Sciences du Traitement de l'Information (EISTI)

GSI Project specifications

Authors : Titouan Bion, Axel Martin et Thomas Viau
Second year engineering students at the EISTI (Pau).

Addressed to M. Juan Angel Lorenzo del Castillo

March 21, 2016

Contents

1 Introduction 2

2 Features description 3
2.1 What will be implemented? . . . 3
2.1.1 Job Queue management . . . 3
2.1.2 Multiple versions . . . 3
2.2 Gantt chart . . . 3

3 Job Queue implementation details 6
3.1 Job parameters . . . 6
3.2 Job Queue parameters . . . 6
3.3 Dynamic feed of the Job Queue using `schedclt` . . . 6

4 Scheduling strategy 8
4.1 Scheduling method description . . . 8
4.1.1 Overall description . . . 8
4.1.2 Time slice . . . 8
4.2 Example of scheduling with our strategy . . . 8

5 Conclusion 10

1 Introduction

As part of our Parallel Programming and Network courses, we are asked to develop a job scheduler. This document is the specifications of this project.
A scheduler is used to assign a job — processes, threads, etc. — to machine hardware resources such as processors. All Operating Systems works with its own scheduler in order to give the user the impression of fluidity even if tons of process are working *"at the same time"*.
First, we will present the features of our program. Then we will elaborate on the Queue management and finally we will develop on our chosen scheduling strategy.

2 Features description

2.1 What will be implemented?

We will implement a `C++` Job Scheduler.
To begin with, let's clarify the definition of a job. A job will be a program launch by a command line. The latter will be given to our application — along with some parameters[1] — by being added to the Job Queue.

2.1.1 Job Queue management

A path — to a file that contains all jobs information — will be provide to our scheduler at start-up. All jobs included in this file will be processed by our scheduler.
However, we will provide a tool to interact with the scheduler to add new job on the fly[2]. Jobs will be feed into the Job Queue regardless of the way it have been added and be processed by our program.

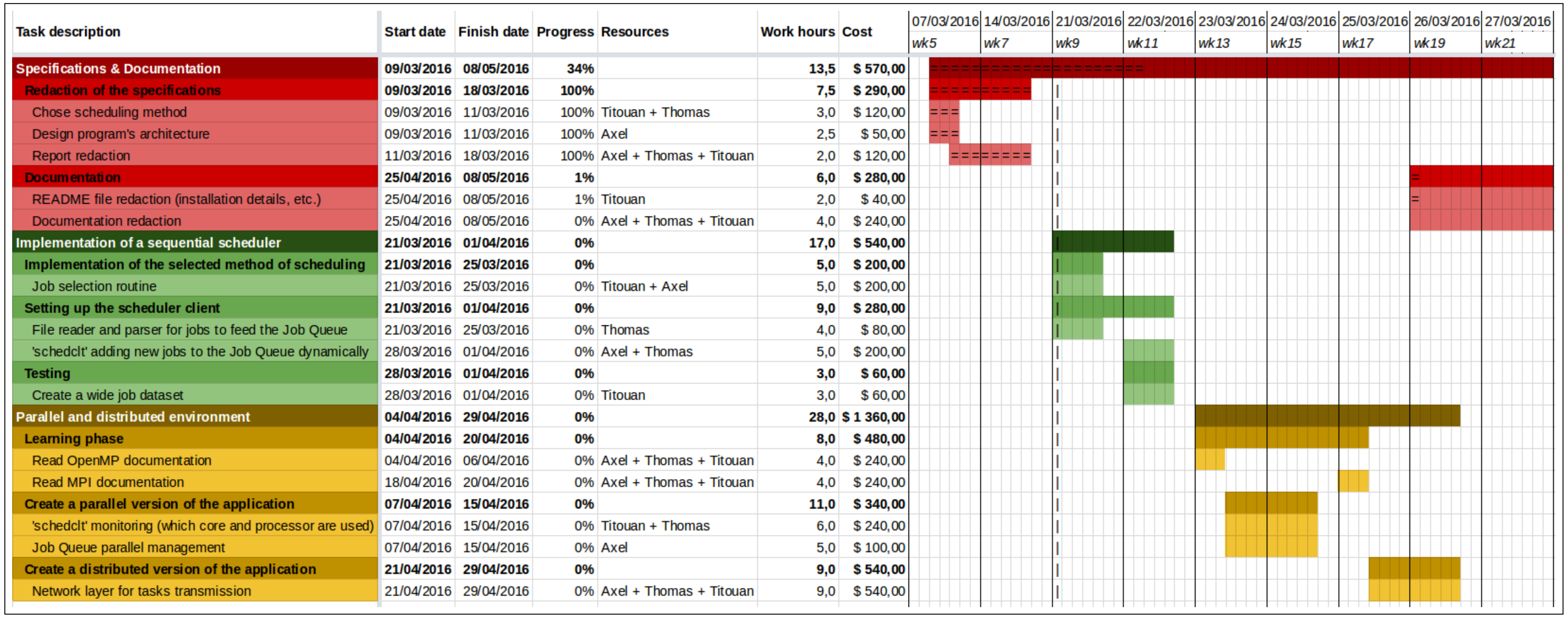
2.1.2 Multiple versions

First, we will develop a sequential management of our Job Queue, each job will be processed one by one and assigned to available resources according to their priorities[3].
Then, we will provided a parallel version of our program to improve our Queue management and processing onto multiple cores of our processor with the "OpenMP" library.
Finally, we will distribute the last version to a cluster of machine in order to make our program scalable with the "MPI" library.

2.2 Gantt chart

We created a Gantt Diagram for our project to organize our work, you can see it on Figure 1.

[1] See 3.1 for more information on job parameters.
[2] See our Job Queue implementation details for more information: 3.3
[3] See our Scheduling Strategy for more information: 4



Task description	Start date	Finish date	Progress	Resources	Work hours	Cost
Specifications & Documentation	**09/03/2016**	**08/05/2016**	**34%**		**13,5**	**$ 570,00**
Redaction of the specifications	**09/03/2016**	**18/03/2016**	**100%**		**7,5**	**$ 290,00**
Chose scheduling method	09/03/2016	11/03/2016	100%	Titouan + Thomas	3,0	$ 120,00
Design program's architecture	09/03/2016	11/03/2016	100%	Axel	2,5	$ 50,00
Report redaction	11/03/2016	18/03/2016	100%	Axel + Thomas + Titouan	2,0	$ 120,00
Documentation	**25/04/2016**	**08/05/2016**	**1%**		**6,0**	**$ 280,00**
README file redaction (installation details, etc.)	25/04/2016	08/05/2016	1%	Titouan	2,0	$ 40,00
Documentation redaction	25/04/2016	08/05/2016	0%	Axel + Thomas + Titouan	4,0	$ 240,00
Implementation of a sequential scheduler	**21/03/2016**	**01/04/2016**	**0%**		**17,0**	**$ 540,00**
Implementation of the selected method of scheduling	**21/03/2016**	**25/03/2016**	**0%**		**5,0**	**$ 200,00**
Job selection routine	21/03/2016	25/03/2016	0%	Titouan + Axel	5,0	$ 200,00
Setting up the scheduler client	**21/03/2016**	**01/04/2016**	**0%**		**9,0**	**$ 280,00**
File reader and parser for jobs to feed the Job Queue	21/03/2016	25/03/2016	0%	Thomas	4,0	$ 80,00
'schedclt' adding new jobs to the Job Queue dynamically	28/03/2016	01/04/2016	0%	Axel + Thomas	5,0	$ 200,00
Testing	**28/03/2016**	**01/04/2016**	**0%**		**3,0**	**$ 60,00**
Create a wide job dataset	28/03/2016	01/04/2016	0%	Titouan	3,0	$ 60,00
Parallel and distributed environment	**04/04/2016**	**29/04/2016**	**0%**		**28,0**	**$ 1 360,00**
Learning phase	**04/04/2016**	**20/04/2016**	**0%**		**8,0**	**$ 480,00**
Read OpenMP documentation	04/04/2016	06/04/2016	0%	Axel + Thomas + Titouan	4,0	$ 240,00
Read MPI documentation	18/04/2016	20/04/2016	0%	Axel + Thomas + Titouan	4,0	$ 240,00
Create a parallel version of the application	**07/04/2016**	**15/04/2016**	**0%**		**11,0**	**$ 340,00**
'schedclt' monitoring (which core and processor are used)	07/04/2016	15/04/2016	0%	Titouan + Thomas	6,0	$ 240,00
Job Queue parallel management	07/04/2016	15/04/2016	0%	Axel	5,0	$ 100,00
Create a distributed version of the application	**21/04/2016**	**29/04/2016**	**0%**		**9,0**	**$ 540,00**
Network layer for tasks transmission	21/04/2016	29/04/2016	0%	Axel + Thomas + Titouan	9,0	$ 540,00

Figure 1: The project Gantt Diagram.

We share working time equally between all group members as you can see on Figure 2.

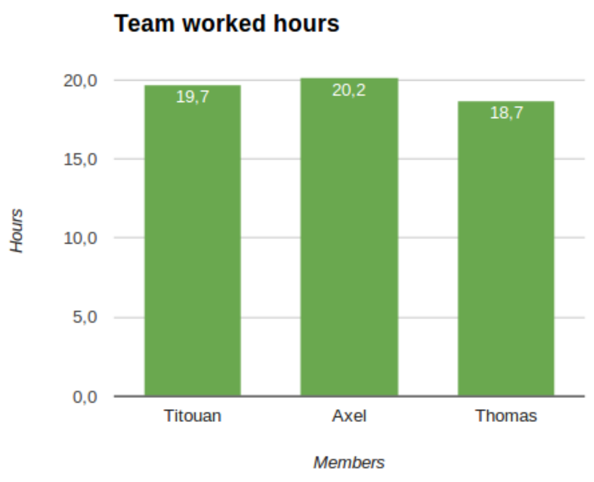


Figure 2: Hours repartition between group members.

3 Job Queue implementation details

3.1 Job parameters

A job will eventually have different properties — this list is not exhaustive and may differ at the end — provided by the user:

- Burst time: the time the job will use for the CPU to compute the job **[required]**
- Priority: for example it could be implemented as an integer, highest priority to the lowest integer
- RAM and CPU load

3.2 Job Queue parameters

Despite the fact that our program could work perfectly with default values and use the best for the system, the Job Queue itself can have optional properties. These properties will be provided by the user:

- `Timeout`: the time after which the application will kill the job, if set to 0 there would be no timeout and no job would be killed
- `Number of cores`: the number of processors the scheduler can send jobs to, by default it uses different values, depending on the version of the scheduler. In the sequential version it could not be modified and will be set to 1. In the parallel and distributed version it will match the number of available processors.
- `Time Slice`: if set to `0`, there would be no time slice and the scheduler would be non-preemptive[4].

3.3 Dynamic feed of the Job Queue using `schedclt`

A tool `schedclt` — for scheduler client — will be developed in order to interact with the scheduler. This tool will take as parameters the same information required to launch a job. Then, it will send a `SIGUSR1` signal to the scheduler, announcing the latter that it needs to reload its job queue. You can see an illustrated version of this routine on Figure 3. `schedclt` will also allow the user to see the current state of the processors — i.e. which job run on this processor.

Implementation The first implementation will use a swap file in order to add jobs to the scheduler's Job Queue in real time. The swap file used for the job description and the communication between the client and the scheduler **should not** be edited by the user.
We may lately design a complete API in the scheduler daemon for retrieving or sending other information such as the cores numbers in the cluster, parametrize the cluster or even remove process.

[4]See our Scheduling Strategy for more information: 4

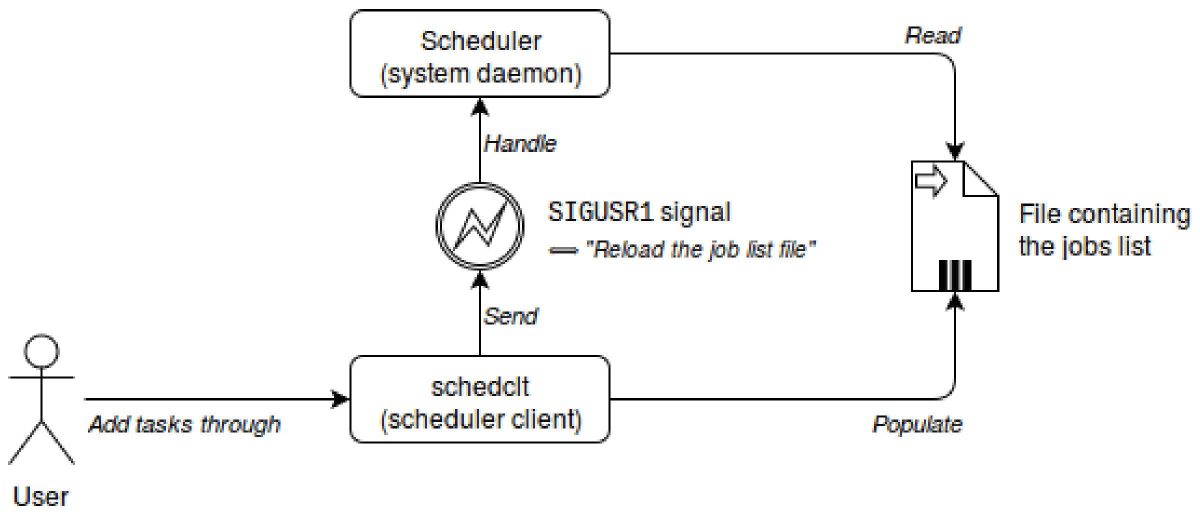


Figure 3: Chart of our Queue implementation.

4 Scheduling strategy

4.1 Scheduling method description

There are many methods and as many algorithms to achieve scheduling. We chose the method that prioritize the shortest burst time programs to make the Operating System more dynamic and provide the best user experience. This method is used by the majority of Operating Systems, as Windows and UNIX systems (Mac, Linux distributions).

4.1.1 Overall description

This method named **Fixed-priority pre-emptive scheduling** will always execute the job with the highest priority to ensure that the user can have what he wants first. A job not being prioritized by the user will have a *default priority*. In the case where there are no prioritized jobs and several default priority jobs, the first job that will be executed is the one with the shortest *burst time* — the time the job will use for the CPU to compute the job — to ensure the velocity of the method. We will also implement the concept of *aging* priority too. A process will not wait forever in an infinite loop. Indeed, the time elapsed since the job's declaration will raise its priority. For instance, a job with default priority, a burst time of 600ms which was declared for 10s, will be run **before** a job with default priority, a burst time of 550ms which was declared for 4s.

4.1.2 Time slice

The *time slice* concept is used in this method too. It means that a job will not hog the processor for a time longer than a time slice. A job can be suspended if a job with higher priority comes in and will be resumed after the termination of the latter. Actually, this *priority race* is checked each time slice. It can be turned off to become a Fixed-priority non-preemptive scheduling, but the pre-emptive method is the one we will use to ensure different jobs can be executed *"simultaneously"*.

4.2 Example of scheduling with our strategy

The following chart (Figure 4) shows how processes are assigned to cores of a processor. Here we have a duo-core processor with four processes:

- A → Burst time: 300ms ; declared for 3s ; priority: high
- B → Burst time: 100ms ; declared for 2s ; priority: high
- C → Burst time: 200ms ; declared for 2s ; priority: normal
- D → Burst time: 100ms ; declared for 2s ; priority: normal

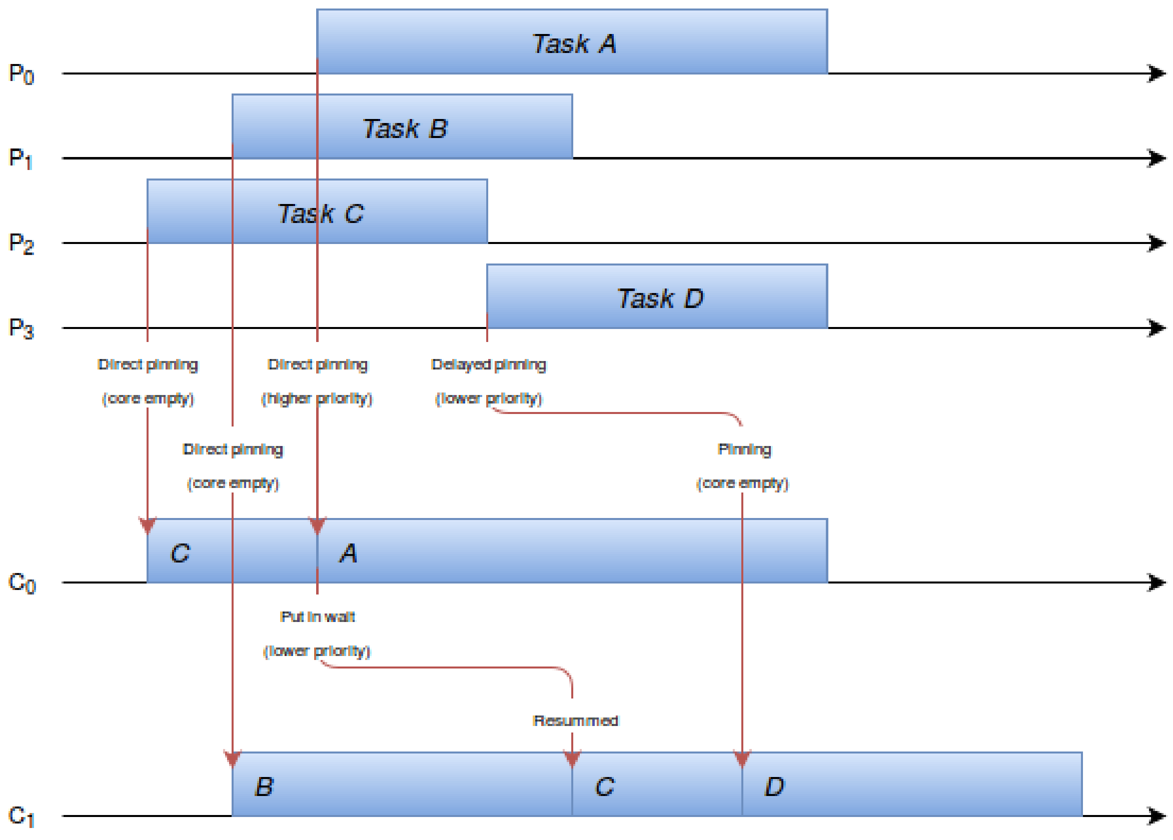


Figure 4: Example of scheduling following our strategy.

5 Conclusion

Among all strategy, we prefer to focus our strategy on the user needs and experience as we empirically know that an good application is the one loved by its users. We hope this approach suit you and will provide all our knowledge to produce the best result we could achieve.