HAL 9000 System

Operating Systems – Year 23/24– HAL 9000 System – Version 1.2

Introduction to Operating Systems projects

An operating system is composed basically of 4 modules or subsystems:

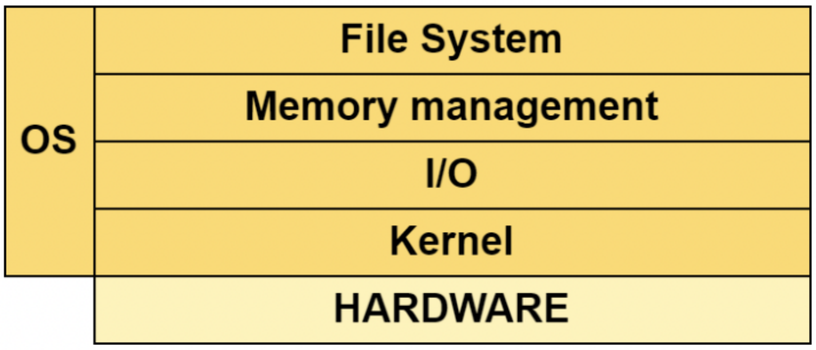

Figure 1. Architecture by layers of an Operating System.

KERNEL. It is the only layer that can inhibit interrupts. Its basic functions are the representation of the processes in execution, the control of the concurrence of processes, the management, and control of interrupts, and providing the system with mechanisms of communication between processes, synchronization, and mutual exclusion.

I/O SYSTEM. It is the only layer that can execute input and output instructions. Its function is to communicate with peripherals. Therefore, the upper layers must interact with the hardware through this layer.

MEMORY MANAGEMENT. Together with the hardware, it creates the possibility of having virtual memory. In addition, this layer is responsible for ensuring (also in conjunction with the hardware) the protection of memory data and sharing it.

FILE SYSTEM. It gives the system a structured view of the data stored on the disk. User applications are user-created programs. When these programs need some functionality of the operating system, they perform what is called a "system call", which consists of calls to functions offered by the different layers. That is, if the application wants to display a character on the screen, it will need to call an I/O layer function to achieve this.

The content of the project of the Operating Systems subject is oriented to the learning of a distributed architecture through the different communication mechanisms (sockets mainly), highlighting the problem of processes concurrency and multitasking. The main goal is to enable connectivity between the different computers and allow the transmission of information between the various nodes of a distributed system. In addition, the use of kernel mechanisms (shared memory, mutual exclusion methods, synchronization, and process creation) is required in order to solve the project.

NOTE 0. The four projects of the 2004-2005 academic year were named Atreides, Atreides++, Harkonnen, and Fremen. It was all in honor of Frank Herbert and his famous novel Dune. Many students remembered the origin and found out where the name came from.

NOTE 1. The four projects of the course 2005-2006 received the names Oedipus Rex, Sphinx, Antigone, and Teiresias. Like all SO projects, there is nothing random. The names came from the Oedipus tragedy from Sophocles and gave meaning to the project. The course began with a simple project 1, Oedipus the King. Oedipus lives happily as king while ignoring that he has killed his father and married his mother. Ignorance makes him happy. OS students live happily because project 1 is easy and they don't know what awaits them. Project 2 is the Sphinx. Oedipus must pass the Sphinx test, otherwise, it will kill him. Obviously, project 2 of the 2005-2006 year was the most difficult of all and if you did not pass it, you wouldn’t pass the subject. Project 3 was called Antigone, who helps Oedipus once he falls into disgrace and despair. Antigone was the salvation of students who wanted to take the June exam and needed a long project delivered. I save you the details of Teiresias. Although many students followed the names and references, none of them fully grasped the meaning of the tragedy...

NOTE 2. The projects of the academic year 2006-2007 focused on Dante Alighieri's masterpiece, "Divina Commedia". The projects were called "Inferno", "Purgatorio" and "Paradiso". I guess it doesn't take much comment to understand what's going on in Project 1. The second project is softened but you need it if you want to take the exam. Finally, the third is relatively short and simple, a paradise with everything you have done before.

NOTE 3. The projects of the academic year 2007-2008 focused on the masterpiece of the Wachowski brothers, "Matrix", where humans (students) have to fight against machines to survive (pass the subject). The projects were called "Matrix", "Trainman" and "Keymaker". Matrix was the central server where customers, such as Nebuchadnezzar, had to connect in order to be able to exchange files and communicate through a distributed chat. They could also connect with other demons, such as Oracle or Link, who gave them advice. Trainman was a project of process load simulation and memory management. If you know the role of the Trainman in the film Matrix you will see that the relationship is direct. Finally, file encoding in EXT2. We considered that a good name to decode was the Matrix character called Keymaker.

NOTE 4. The projects of the academic year 2008-2009 focused on Isaac Asimov's masterpiece, "The Foundation". The projects were called: "Trantor", the central planet of the galaxy, therefore having the same magnitude as project 1 of Operating Systems; "Trevize, the loader", named after the director of the first foundation, with an unusual intuition, but with dangerous intentions; and "Pelorat, the file reader", name of the history professor who helped Trevize find planet Earth, the home to rest after passing the three operating systems projects.

NOTE 5. The projects of the academic year 2009-2010 focused on the Australian hard rock group AC/DC, as a tribute to the departure of the course's intern teacher for the last three years, Hugo Meza. The first project, under the name Highway to shell (a small modification of the name of the most popular song in the group), led the students through three phases. The first, Welcome to the Shell, welcomed the students to hell with the implementation of a shell command

OPERATING SYSTEMS Project 2021-2022 3 interpreter, under the name Malcolm (singer of the group). In the second phase, in order to escape the darkness in which they were, the students agreed with the devil (Dealing with the Devil), where they had to follow a protocol to communicate with the demon Angus (guitarist of the group). Finally, when they thought it was all over, they met Ballbreaker (an album they recorded in 1995), a name that does not require any explanation. The second project of 2009- 2010 was called Back in Black, the name of the album released in 1980, where students had to identify the format of a given volume and extract information from it.

NOTE 6. The projects of the 2010-11 academic year focused on the works of the controversial Donatien Alphonse François de Sade, better known as "Marquis de Sade". Thus, the project was titled "Les 120 journées de Sodome". This was divided into three phases: "Le Château de Silling" was the first of them, where the students managed the behavior of the client of the application, which was named after the character in the novel "Blangis". Later the students entered the castle in a second phase called "Les quatre Madames", where Blangis connected to a demon called "Thérèse". Finally, the thing fainted in the third phase, where the students had to create the server "Libertinage", which, as the name suggests, is where the "festival" began.

NOTE 7. The projects of the 2011-12 academic year focused on the world-famous television series of the Simp(so)ns. This project was divided into five phases: "Homer's Shell", "Mou's Bar", "Clancy Wiggum", "Living in Springfield" and "Living in Springfield++", as a whole these phases formed a system that allowed to execute commands locally, some of their own on a remote server and the activation of various services that showed mythical phrases of the series, all following a client-server architecture.

NOTE 8. The project of the 2012-13 academic year was called LsBox. The reason was quite simple: it was about designing and implementing a very similar (in fact simplified) system of the well-known Dropbox. The only curious detail was the project logo bearing an undercover Map of Australia as it was a small tribute to an OS intern that left these tasks.

NOTE 9. The project of the 2013-14 academic year was called LsHangIn. The reason was that it was a simplified version of the well-known Google Hangouts: it was about designing and implementing a multi-room chat system for users. In addition, each of the phases had a relation with the film The Hangover.

NOTE 10. The project of the 2014-15 academic year was called Gekko. Gekko is a businessman and the main protagonist of the film Wall Street, a direct relationship with the stock exchange and the goal of the project. But there were also several other tributes: TumblingDice, the fluctuation generator, is also a Rolling Stones song, and Dozer, the stockbroker, is one of the Matrix characters.

NOTE 11. The project of the 2015-16 academic year was called LsTransfer, the force awakens. It was a clear tribute to the return of the Star Wars saga. In addition, the server was called Naboo (the planet that appears in different episodes of Star Wars) and Gungan customers (inhabitants of this planet).

NOTE 12. The project of the 2016-17 academic year was called LsTinder, may the Love be with you. It was because of the clear similarity with the social network to which the project referred, and the motto another tribute to the Star Wars saga. In addition, the different processes were called Rick and Morty in reference to an American adult animated television series.

NOTE 13. The project of the 2017-18 academic year was called LsEat, may the Food be with you. First, the name is in reference to Just Eat fashion. The motto is another tribute to the Star Wars saga, in its imminent premiere of Episode VIII, The Last Jedi. In addition, the different processes had named after Star Trek. Picard and Data characters and Enterprise, the ship.

NOTE 14. The project of the 2018-19 academic year was called the Cosgrove System, Stairway to heaven. First, Cosgrove is the last name of the family starring in the legendary Peter Jackson film Braindead, considered a classic of gore films of the time. The motto, Stairway to Heaven, was nothing related to the observatories of the project but a tribute to the Led Zeppelin song. In addition, the different processes had names of the protagonists of the Braindead film: McGruder, McTavish, Paquita, and Lionel.

NOTE 15. The project of the course 2019-20 called Cypher System was a tribute again to the Matrix, as it is being recorded for the 2022 Matrix IV, a myth in the world of science fiction. If you ran the cover script, it showed the green letters falling across the Matrix screen. Trinity is the main protagonist of Matrix and the examples are always names of characters in the saga. There's nothing random.

NOTE 16. The 2020-21 project was called Overlook System. This was a tribute to Stanley Kubrick's legendary film The Shining, which celebrated its 40th anniversary in 2020. The Hotel was called Overlook and the image on the cover of the project was one of its mythical corridors. The different processes of the system design were Jack, Wendy, Danny, and Lloyd; which are the names of the personages of the saga. There is nothing random.

NOTE 17. The 2021-22 project was called the Arrakis System. This was a tribute to Denis Villeneuve's legendary 2021 film Dune, but taking into account David Lynch's original from 1984. Arrakis is the planet where the majority of the film takes place. There were the Atreides (the good guys), the Harkonnen (the bad guys), and the Fremen (the native race of the planet Arrakis) who are important and key elements in the legendary film.

NOTE 18. The project of the academic year 2022-23 was called Eä System. This was a tribute to the series The Lord of the Rings: The Rings of Power. It is a television series based on the novel The Lord of the Rings and its appendices by J. R. R. Tolkien. It takes place thousands of years before Tolkien's The Hobbit and The Lord of the Rings in the Second Age of Middle-earth. It is produced by Amazon Studios.

From here, the interpretation of the names and concepts of this year's project (relating to the difficulty, content, and subject) is up to you... and we hope to see them in the final report!

HAL 9000 System

At the beginning of the digital music revolution, the companies Napster and Spotify changed the way people listened to music. Napster pioneered the sharing of MP3 files between users over the Internet, while Spotify introduced the concept of legal music streaming.
Seeing Spotify's business success, different competitors also want to enter the market, offering better solutions and different architectures so that users can enjoy the same service without, for example, having to pay subscriptions or having to share accounts with friends , friends, partners or even ex-partners.
With the great reputation that Operating Systems students have, companies have contacted us to be able to develop a first version of Spotify's future competitor... SO Team to the rescue!
The system to be implemented has been called HAL 9000 by the company that hired us. Its focus is ease of use for the user, along with the speed and reliability of the system to easily attract new customers. This is why it will be necessary to implement an architecture and technological infrastructure, where it will be necessary to configure, design and implement a communications system to be able to manage music. We start from a set of devices that have a Linux operating system installed and the TCP/IP network.
The ecosystem to be designed is based on having different physical servers, where each will have music available for users to download. These servers have unknown physical locations, and we will only be able to access them via IP and port. Users will connect to these servers and can make requests to them such as downloading songs, playlists, or viewing all the music available on the server.
It will therefore be necessary for us to design and implement again all the software of the ecosystem according to the design specifications, requirements, and restrictions that we will receive. Of course, now with robustness, efficiency, and scalability. To make our work easier, the design will be given to us in incremental phases. In each of them, its functionalities and limitations are specified, and a set of execution and testing examples is given.

Figure 1. Logo of Napster, a pioneering program in file sharing

General overview

A first functional approximation of the system's process architecture would be the one presented in Figure 2, although, obviously, the internal design architecture will be much more complex.

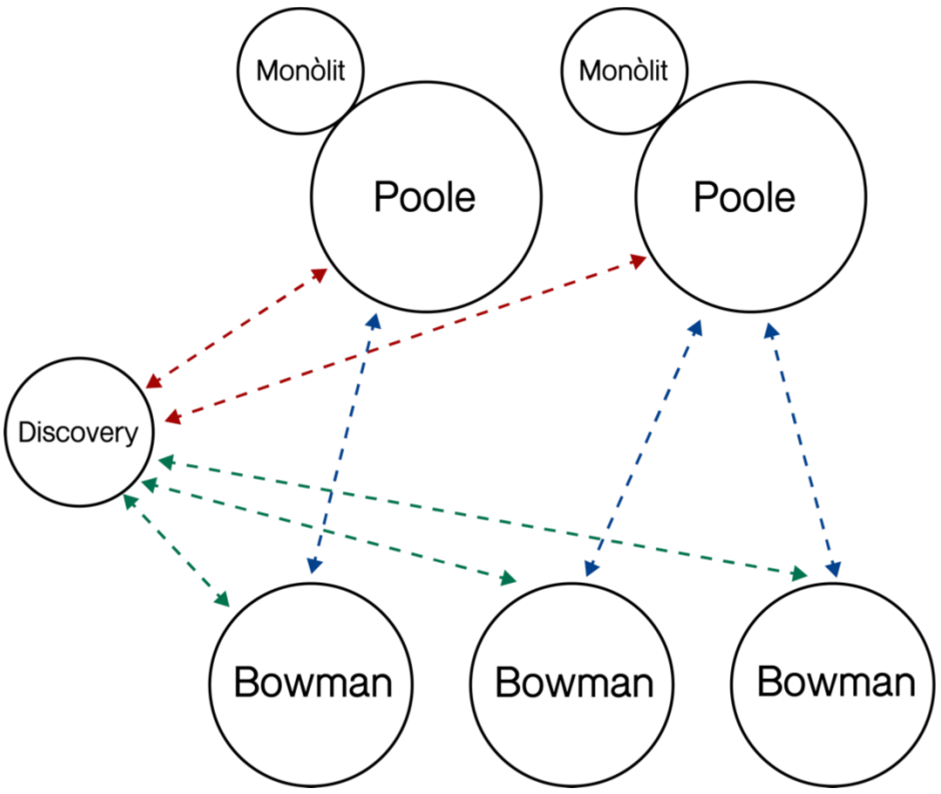

Figure 2. HAL 9000 System: functional diagram.

The **Poole processes** will act as servers that will store the music available for download. The **Bowman processes** will be the different users of the application, who will want to connect to see the available music and be able to get the songs or playlists they want. To avoid the possible saturation of one of the servers, an improvement of the architecture has been proposed with a third process, the **Discovery process**, which will act as a load balancer for the different **Poole** servers. This way, when a user wants to connect to the **HAL 9000** system, it will connect to the **Discovery** server and then, transparently to the user, connect to the **Poole** server with the fewest users connected at that time (the load balancer will decide which music server the user will connect to).

In short, the main goal of the system is that the different Bowman processes can communicate with a music server (Poole), where they can make requests and download MP3 files. This operation will be detailed later.

To be able to design and implement the project, it is recommended that you **read the statement in its entirety**. To facilitate its implementation, 4 phases have been planned:

1. **Phase 1** will create the Bowman process and the Poole process. From the Bowman process, the configuration file will be processed and a shell implementation will be performed with the corresponding extended commands. From the Poole process, only the configuration file will need to be processed.
2. In **phase 2,** the Discovery process will be implemented, together with the different connections between servers and the load balancer. The first frames between processes will also be made.
3. In **phase 3,** all the communication between the different processes will be carried out, including the download of music, playlists... and client disconnections.
4. In **phase 4,** a fifth process will be implemented, the Monolith, which will be responsible for computing different statistics of the music servers and maintaining these persistent statistics between executions.

Phase 1: Just warming up

In Phase 1, the design and programming of Bowman and Poole will begin. In both processes, the configuration file, whose name is received as a parameter, will first need to be processed.

In addition, the Bowman process will work in command line mode and will have a reduced set of functions programmed (See Table 1).

Configuration file - Bowman

This text file will have several fields:

- Name of the executing user.
- Name of the folder where the user's files are located.
- The IP and port (each on a line ending with a '\n') where the Discovery server is located.

Example configuration text file in Figure 4:

```
Floyd
/floyd
192.168.50.51
8010
```

Figure 3. Configuration file of a Bowman process.

Configuration File - Poole

This text file will have several fields:

- Name of the music server.
- Name of the folder where the server files are located.
- The IP and port (each on a line ending with a '\n') where the Discovery server is located.
- The IP and port (each on a line ending with a '\n') from which the Poole server will be opened.

Example configuration text file in Figure 4:

```
Smyslov
/smyslov
192.168.50.51
8011
192.168.50.51
8015
```

Figure 4. Configuration file of a Poole process.

Then you will need to design and implement all the program logic that interprets all the commands, which allow Bowman to function. These commands are case insensitive.

COMANDA	**DESCRIPCIÓ**
`CONNECT`	Connect the user to the system. It will first connect to the load balancer (Discovery) and, automatically and transparently to the user, to a Poole server.
`LOGOUT`	Disconnects the user from the system
`LIST SONGS`	Lists all available songs from the Poole server the user is connected to.
`LIST PLAYLISTS`	Lists all available playlists from the Poole server the user is connected to.
`DOWNLOAD <SONG/PLAYLIST>`	Download a song or a list of songs.
`CHECK DOWNLOADS`	Check the status of ongoing downloads, in the form of progress bars
`CLEAR DOWNLOADS`	Clear completed downloads from the download list.

Table 1. List of extended orders.

```
$matagalls:> Bowman config.dat

Floyd user initialized
$ DOWNLOAD MACARENA.MP3
Cannot download, you are not connected to HAL 9000
$ CONNECT PLEASE
Unknown command
$ CONNECT
Floyd connected to HAL 9000 system, welcome music lover!
$ CHECK DOWNLOADS
You have no ongoing or finished downloads
$ LIST SONGS
There are 6 songs available for download:
1. Macarena.mp3
2. Walk_of_life.mp3
3. Levels.mp3
4. Less_is_more.mp3
5. Stand_up.mp3
6. Isla_nostalgia.mp3
$ DOWNLOAD Isla_nostalgia.mp3
Download started!
$ something else
ERROR: Please input a valid command.
$ LIST PLAYLISTS
There are 2 lists available for download:
1. Pim_pam_trucu_trucu
      a. Levels.mp3
      b. Stand_up.mp3
      c. Macarena.mp3
2. Copeo_pre_costa
      a. Macarena.mp3
      b. Isla_nostalgia.mp3
$ DOWNLOAD Pim_pam_trucu_trucu
Download started!
$ CHECK DOWNLOADS
Isla_nostalgia.mp3                    96% |====================% |
Pim_pam_trucu_trucu - Levels.mp3      61% |=============%        |
Pim_pam_trucu_trucu - Stand_up.mp3    45% |=========%            |
Pim_pam_trucu_trucu - Macarena.mp3    13% |==%                   |
$ LOGOUT
Thanks for using HAL 9000, see you soon, music lover!

$matagalls:>
```

Figure 5. Execution of a Bowman process.

It is requested:

- That you design and program Bowman.c (name of the program that will be used by clients).
- That you program Poole.c (name of the program that will be used by the music servers).
- Bowman first processes the configuration file and stores its information in an appropriate structure.
- Bowman will have to recognize (but not implement at this stage) the orders described above.
- Poole will process the configuration file and save its information in a suitable structure.

Considerations Phase 1:

- Once the execution of the Bowman and Poole processes has been completed, all types of significant dynamic memory must be freed, if any.
- Users name cannot contain '&'. If they contain any, they must be removed. This is due to the system's communication protocol, which is in the Annex. This will be case sensitive.
- Orders are case insensitive.
- We can consider that the format of the configuration file is correct.
- Printf, scanf, gets, puts, etc. functions cannot be used. You can only interact with the screen and files with the read and write functions. Yes, it is allowed to make use of the sprintf function and similar functions like asprintf, etc.
- The functions system, popen or any variant cannot be used.
- The stability of the application and its correct operation must be guaranteed. Under no circumstances can infinite loops, core dumps, active waits, compile warnings, etc. occur. It is also necessary to control all those aspects likely to give an error and, if they occur, inform the user properly and, if possible, continue with the normal operation of the application.
- From now until the end of the project it is necessary to consider that the Bowman process can terminate its execution by using the appropriate command or by pressing CTRL+C. You must take this into account and try to make the whole system as stable as possible in case it happens.
- It is mandatory to use a makefile to generate the executable.
- It is not enough that the file you upload to the well has a .tar extension, but it must be able to be unpacked with the tar command. Any project or checkpoint that cannot be "unzipped" in this way will not be corrected.
- Remember that the code must be properly modulated, that is to say: everything cannot be located in a single .c file, and a makefile must be present. Note that if when trying to correct a project it does not compile with the make command (for whatever reason), the delivery will be qualified as not suitable (2).

Phase 2: Feeling the connection

In this second phase it will be necessary to implement the connections of the Bowmans with the Poole servers. As explained briefly in previous sections, a third process (Discovery) will act as a load balancer so that clients always connect to the least saturated server at that time.

Since the various Bowman, Poole, and Discovery processes can obviously be on different physical servers, this connectivity will need to be designed using ***sockets***. Review the **communication protocol** in the Appendices.

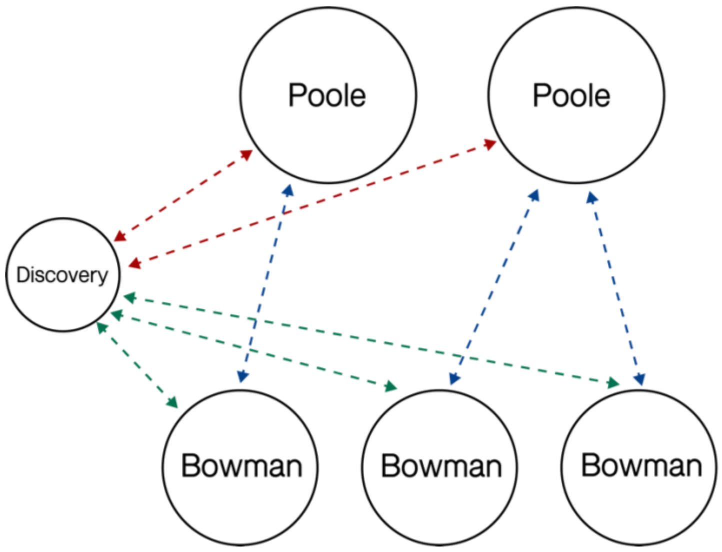

Figure 6. Scheme to be designed and implemented.

Poole servers will connect to Discovery to indicate that they are ready to receive client connections. This connection will be momentary and will only be made at the beginning of the Poole processes connection.

Similarly, Bowman processes will also connect to Discovery to notify that they want to enter the system. Discovery will respond to this connection with a new server to connect to (the Poole process already registered on the system with fewer connected users), and the Bowman process will make the new connection (automatically). Only this last connection will be stable throughout its execution since, as we will see, the two processes will exchange frames during the execution of the system.

The Discovery server will always be up and running. This will never be closed or receive any interrupts that could interrupt its execution. On the other hand, the possible disconnections of Bowman and Poole will have to be managed, but not at this stage.

Having said that, the following will need to be implemented:

1. The Discovery process, which will process a configuration file and start receiving connections from Poole and Bowman processes.
2. Poole's connectivity to Discovery (not stable).
3. Bowman's connectivity with Discovery (not stable).
4. Bowman's connectivity to Poole (stable), along with commands to list files available on Poole servers.

Configuration file - Discovery

This text file will have several fields:

- The IP and port (each on a line ending with a '\n') from which Poole processes will listen for connections.
- The IP and port (each on a line ending with a '\n') from which Bowman processes will listen for connections.

Example configuration text file in Figure 4:

```
192.168.50.51
8010
192.168.50.51
8011
```

Figure 7. Configuration file of a Discovery process.

System connectivity

Below is an example of connections with an environment with 1 Discovery process (there will always be only 1, and it will always be on), 2 Poole processes, and 3 Bowman processes.

1. Discovery starts its execution and listens for requests from Poole and Bowman processes.
2. The two Poole processes connect to the system, making requests to Discovery.
3. A Bowman process is connected to Discovery. It responds with the IP address and port (see frame protocol annex) of any of the two Poole servers (both have 0 users connected). Discovery updates a list of connected users, noting that a client is connected to Poole_1.
4. A second Bowman process is connected to Discovery. It responds with the IP address and port of the Poole server that does not yet have a user connected. Discovery updates a list of connected users, noting that a client is connected to Poole_2.

5. A third Bowman process is connected to Discovery. It responds with the IP address and port of either of the two Poole servers, since they both have a client connected each. Discovery updates a list of connected users, noting that a client is connected to Poole_2.

Once Bowman is connected to Poole, he can start throwing requests. In this phase it is necessary to implement the orders of:

1. SONGS LIST
2. LIST PLAYLISTS
3. LOG OUT

LOGOUT

It is important to note that, when facing a Logout, Bowman will have to notify Discovery, which is what always maintains the updated list of users and which Poole servers they are connected to. In the same way, in this phase it will be necessary to manage when the Bowman process exits the system with CTRL+C.

That said, you can handle Bowman's opt-out however you like for Discovery to update its list:

1. Bowman notifies Poole, who notifies Discovery. Discovery then removes Bowman from the list and sends an ACK to Poole. Poole sends an ACK to Bowman, who disconnects.
2. Bowman notifies Discovery and Poole, and waits for confirmation from both to disconnect.

Once Bowman goes offline, you have two implementation options again:

1. Finish the process
2. Make the Shell available again, in case you want to connect again

However, if a Bowman logs out without being logged in, it will terminate its execution.

The design decisions of the logout will have to be explained in a suitable way in the memory of the practice, while making a diagram of the communication implemented.

At this stage, the disconnection of Poole processes will not need to be handled. You can consider that Poole and Discovery will be servers that will always be connected and will not crash during execution. To see a simple example of behavior you can look at the following figure:

```
$matagalls:> Poole config.dat
Reading configuration file
Connecting Smyslov Server to the system..
Connected to HAL 9000 System, ready to listen to Bowmans petitions

Waiting for connections...

New user connected: Floyd.

New request - Floyd requires the list of songs.
Sending song list to Floyd

New user connected: Elena.

New request - Floyd wants to download Isla_Nostalgia.mp3
Sending Isla_Nostalgia.mp3 to Floyd

New request - Elena requires the list of playlists.
Sending playlist list to Elena

New request - Elena wants to download the playlist Copeo_pre_costa
Sending Copeo_pre_costa to Elena. A total of 2 songs will be sent.
…
```

Figure 8. Poole execution example

Considerations Phase 2:

- Those of Phase 1 are still valid.
- In this Phase file transfer must NOT be implemented (DOWNLOAD SONG AND DOWNLOAD PLAYLIST) but LIST SONGS, LIST PLAYLIST and LOGOUT.
- It will be necessary to manage the falls of the Bowman processes, and maintain the stability of the system.
- It can be considered that Poole and Discovery will not be able to fall in the face of this phase.

Phase 3: Pirates of the Caribbean

In this phase, the connectivity between the different processes of the HAL 9000 system will be implemented. This phase is responsible for managing the sending of frames and files between the Bowman and Poole processes.

You will also need to implement the CHECK DOWNLOADS and CLEAR DOWNLOADS commands, to be able to track ongoing and completed downloads from Bowman processes. It will also be necessary to manage the controlled crashes of the Poole servers, by Control + C.

Remember that the communication protocol to be implemented can be found in the Annex.

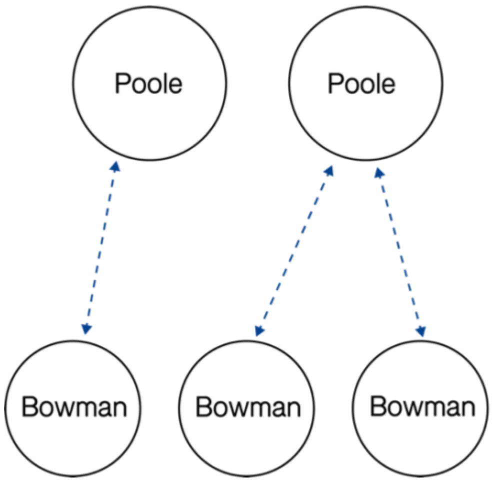

Figure 9. Communication to be implemented in phase 3, including all the file sending frames

Let's remember some details already explained previously that are relevant:

- With the DOWNLOAD SONG_NAME command, Bowman asks Poole to send him a song that Poole has. It will be necessary to manage that the user requests a song that does not exist.
- With the command DOWNLOAD PLAYLIST_NAME, Bowman asks Poole to send him ALL the songs in a certain playlist that Poole has. It will be necessary to manage that the user requests a playlist that does not exist.
- With the CHECK DOWNLOADS command, Bowman will be able to see his downloads in progress, along with a progress bar indicating where they are going, or if they are already complete.
- With the CLEAR DOWNLOADS command, Bowman will be able to clear the downloaded songs from the download list.
- If you receive a file with the same name as a previously received file, it will be overwritten.

```
$matagalls:> Bowman config.dat

Floyd user initialized

$ CONNECT
Floyd  connected  to  HAL  9000
system, welcome music lover!

$ LIST SONGS
There  are  6  songs  available  for
download:
1. Macarena.mp3
2. Levels.mp3
3. Stand_up.mp3
4. Isla_nostalgia.mp3

$ LIST PLAYLISTS
There  are  2  lists  available  for
download:
1. Pim_pam_trucu_trucu
      a. Levels.mp3
      b. Stand_up.mp3
2. Copeo_pre_costa
      a. Macarena.mp3
      b. Isla_nostalgia.mp3

$ DOWNLOAD Isla_Nostalgia.mp3
Download started!

$ DOWNLOAD Pim_pam_trucu_trucu
Download started!

$ CHECK DOWNLOADS
Isla_nostalgia.mp3
      100% |====================%|
Pim_pam_trucu_trucu - Levels.mp3
      61% |=============%        |
Pim_pam_trucu_trucu - Stand_up.mp3
      45% |=========%            |

$ CLEAR DOWNLOADS
Pim_pam_trucu_trucu - Levels.mp3
      73% |===============%      |
Pim_pam_trucu_trucu - Stand_up.mp3
      65% |=========%            |

...
```

```
$montserrat:> Poole config1.dat

Reading configuration file
Connecting  Smyslov  Server  to  the
system..
Connected to HAL 9000 System, ready
to listen to Bowmans petitions

Waiting for connections...

New user connected: Floyd.

...

New  request  -  Floyd  requires  the
list of songs.
Sending song list to Floyd

New  request  -  Floyd  requires  the
list of playlists.
Sending playlist list to Floyd

New   request   -   Floyd   wants   to
download Isla_Nostalgia.mp3
Sending    Isla_Nostalgia.mp3    to
Floyd

New   request   -   Floyd   wants   to
download       the        playlist
Pim_pam_trucu_trucu.
Sending   Pim_pam_trucu_trucu   to
Floyd. A total of 2 songs will be
sent.

...
```

Figure 10. Example of communication between a Bowman and a Poole process

Considerations Phase 3:

- Those of the previous phases are still valid.
- **The correct sending of files must be verified with the MD5SUM tool.**
- It is not allowed to use programmed code to do the MD5SUM. You need to run the md5sum command provided by the operating system bash itself (md5sum).
- When a Poole server goes offline, the system should detect this and update the load balancer accordingly.

Optional Phase 3:

- When a Poole server crashes, reconnect all the Bowmans that were connected to the server one by one, while asking Discovery where they should connect.
- Handle Poole and Bowman crashes due to unhandled interrupts (SIGKILL).

Phase 4: Teaching statistics to my children

After implementing all system communication, we have been asked that the HAL 9000 system be able to calculate certain system statistics. That is why we will implement a fourth process, the Monolith process.

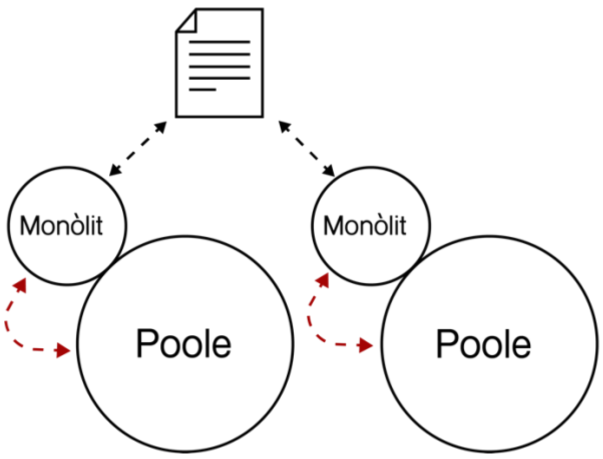

Figure 11. Implementation of the Monolith processes, in charge of computing statistics with the information they receive from the Poole servers

The Monolith processes must be created at the beginning of the Poole execution. In other words, the Poole.c programs will have to create a subprocess (Monolithic) and will have to communicate with it to be able to send it the relevant data to make the statistics.

Communication between Poole and Monolith CANNOT BE DONE WITH SOCKETS OR MESSAGE QUEUES. You will need to think of an alternative communication.

Specifically, you will always need to have a list of all songs downloaded, along with the total number of downloads per song.

Considerations Phase 4:

- Those of the previous Phases remain valid.
- Sockets or message queues cannot be used to communicate between Poole and Monolith.
- The file to be updated will be unique, and will be called "stats.txt".
- Mutual exclusion must be guaranteed when writing the file.
- Communication cannot be performed using secondary memory.

Delivery and planning requirements

This project will have different **control points or *checkpoints*** to be able to monitor it carefully and guarantee the consolidation of each of the phases incrementally. Specifically, the following calendar of deadlines will be followed:

CONCEPT	DEADLINE
Phase 1	23/10/2023
Phase 2	20/11/2023
Phase 3	13/12/2023
Final delivery 1 (out of 10)	08/01/2024
Final delivery 2 (out of 10)	02/02/2024
Final delivery 3 (out of 8)	20/05/2024
Final delivery 4 (out of 6)	21/06/2024

Checkpoints are not mandatory, although they are highly recommended in order to guarantee the robustness of the project. These *checkpoints* will only serve to linearly increase the qualification of the project. In no case will your grade be penalized.

It is also advisable to **validate the overall design of the project** with the project monitors before starting phases 2, 3 and 4. This way you can guarantee that the implementation will not suffer from irreparable inconsistencies due to later phases. That's why you need to take advantage of the doubt times and bring the **printed designs** to be validated. It must be remembered that a project works is no guarantee that it is valid, because the whole design may not meet the requirements of the statement. That is why it is very important to have guarantees about the design to be implemented.

Deliveries will be made to the eStudy in a well with a file that includes the source code (.c, .h files and the makefile) of the project, running entirely on Montserrat, as well as the data and configuration files used. The file must be in .tar format. You can generate it with the following command:

> *tar cf Gx_Fn.tar *.c *.h makefile *.ext_files*
>
> where Fn is the Phase number to be delivered. For example, group 12 would submit G12_F1.tar for Phase 1 checkpoint delivery.

In the final deliveries, it will also be necessary to deposit the report in PDF format. The report must contain, obligatorily, the points indicated below in the corresponding Annex.

Annex I: Report Contents

The report must be properly formatted documentation. It must contain the following sections:

1. **Cover**
2. **Index**
3. **Design**: an explanation of how the project has been designed and structured. It can be explained phase-by-phase or the project as a whole. It must include clearly and understandably:
    a. Diagrams explaining what processes have been created, the different communications between processes, etc.
    b. Data structures used and their justification.
    c. System resources used (signals, sockets, semaphores, pipes, etc.) with their justification. d. Optional phases implemented.
4. **Observed problems and how they have been solved.**
5. **Temporal estimation:** time spent on the whole development of the project for each of the students. The categories to consider are:
    a. Research: time spent looking for tools and/or learning components to implement (without counting the time spent on the lab sessions).
    b. Design: time spent designing the project.
    c. Implementation: coding time.
    d. Testing: time spent testing.
    e. Documentation: time spent writing the report and internal documentation of the code (comments, etc.).
6. **Conclusions and proposals for improvement.**
7. **(Optional) Explanation of ALL the names of the project processes. What theme did we base this year on to make the statement? �**
8. **Bibliography used** (in a correct bibliographic format IEEE, APA, etc.): both bibliographic resources and links.

The writing and the grammatical and orthographic correctness will be assessed. The report must have **numbered pages**.

The report must be delivered in **PDF format** in the final delivery but it is recommended to elaborate it during the implementation of the project.

Annex II: general communication protocol

To carry out communication between processes on different machines, a specific protocol for sending frames will be used, which will be explained below. It should be noted that messages will be sent via sockets using connection-oriented communication.

A single frame type will be used in this protocol. These are of **fixed size (256 Bytes)** and always made up of the following 4 fields:

Type: Describes the frame type in 1-character hexadecimal format that will contain a frame identifier.

Header_Length: 2-byte field in numeric format used to indicate the length of the Header.

Header: This field is used to identify the header of the frame. The length of this field is specified in the previous HEADER_LENGTH field

Date: This field is used to store values or data to be sent by the frame. The length of this field depends directly on the total length of the frame and the size of the header, specified in the HEADER_LENGTH field. Depending on the case, it can contain different values as specified in each functionality.

TYPE	HEADER_LENGTH	HEADER	DATA
(1 Byte)	(2 Bytes)	(X Bytes)	(256 – 3 - X Bytes)

It is very important in order to guarantee good communication with the process that the definition of the structure of the communication frame is strictly, in order and format, the one described previously. This means that any change in the type of the fields or their order within the structure to be designed will cause the frames received or sent to the process to not comply with the communication protocol and, therefore, this communication will not work correctly.

If the size of the data to be sent is less than 256, the frame must be filled with the corresponding *padding*.

It is completely normative to send **entire frames** in a single write to the socket.

NEW CONNECTIONS TO DISCOVERY - POOLE

Poole ➔ Discovery

Frame to ask for a new connection to Discovery server.

- TYPE: 0x01
- HEADER_LENGTH: Length of the header
- HEADER: NEW_POOLE
- DATA: [*userName&IP&Port*]

Discovery ➔ Poole

Frame OK connection.

- TYPE: 0x01
- HEADER_LENGTH: Length of the header
- HEADER: CON_OK
- DATA: Empty

Frame KO connection. It is sent if the connection could not be established.

- TYPE: 0x01
- HEADER_LENGTH: Length of the header
- HEADER: [CON_KO]
- DATA: Empty

NEW CONNECTIONS TO DISCOVERY - BOWMAN

Bowman ➔ Discovery

Frame to ask for a connection to the Discovery server.

- TYPE: 0x01
- HEADER_LENGTH: Length of the header
- HEADER: NEW_BOWMAN
- DATA: [*userName*]

Discovery ➔ Bowman

Frame OK connection

- TYPE: 0x01
- HEADER_LENGTH: Length of the header
- HEADER: CON_OK
- DATA: [serverName&*IP*&*Port*]

Frame KO connection. It is sent if the connection could not be established.

- TYPE: 0x01
- HEADER_LENGTH: Length of the header
- HEADER: CON_KO
- DATA: Empty

<u>**NEW CONNECTIONS TO POOLE - BOWMAN**</u>

Bowman ➔ Poole

Frame to ask for a connection to the Poole server.

- TYPE: 0x01
- HEADER_LENGTH: Length of the header
- HEADER: NEW_BOWMAN
- DATA: [*userName*]

Poole ➔ Bowman

Frame OK connection

- TYPE: 0x01
- HEADER_LENGTH: Length of the header
- HEADER: CON_OK
- DATA: Empty

Frame KO connection. It is sent if the connection could not be established.

- TYPE: 0x01
- HEADER_LENGTH: Length of the header
- HEADER: CON_KO
- DATA: Empty

LIST SONGS

Bowman ➔ Poole

Frame to ask a list of songs to the Poole server.

- TYPE: 0x02
- HEADER_LENGTH: Length of the header
- HEADER: LIST_SONGS
- DATA: Empty

Poole ➔ Bowman

Frame that responds to a user's LIST SONGS request.

Please note that the track list may not fit in a single frame.

You can freely modify the data field of the frame to report how many songs there are in total, or modify the header to inform Bowman of how many frames to expect to receive.

- TYPE: 0x02
- HEADER_LENGTH: Length of the header
- HEADER: SONGS_RESPONSE
- DATA: *[song1&song2&song3&song4&song5...&songN]*

<u>LIST PLAYLISTS</u>

Bowman ➔ Poole

Frame to ask for the list of playlists to the Poole server.

- TYPE: 0x02
- HEADER_LENGTH: Length of the header
- HEADER: LIST_PLAYLISTS
- DATA: Empty

Poole ➔ Bowman

Frame that responds to a user's LIST PLAYLISTS request.

<u>Please note that the list of playlists and songs may not fit in a single frame.</u>

You can freely modify the data field of the frame to report how many lists and songs there are in total, or modify the header to inform Bowman of how many frames to expect to receive.

- TYPE: 0x02
- HEADER_LENGTH: Length of the header
- HEADER: SONGS_RESPONSE
- DATA: *[list1&song1&song2&song3#list2&song1&song2...#listN&song1&song2]*

DOWNLOAD SONG/PLAYLIST

Bowman ➔ Poole

Frame to ask Poole for a song.

- TYPE: 0x03
- HEADER_LENGTH: Length of the header
- HEADER: DOWNLOAD_SONG
- DATA: [song_name]

Bowman ➔ Poole

Frame to ask Poole for a playlist

- TYPE: 0x03
- HEADER_LENGTH Length of the header
- HEADER: DOWNLOAD_LIST
- DATA: [playlist_name]

<u>SEND FILE</u>

Poole ➔ Bowman

Frame to send a music file from a Poole server to a Bowman process.

First, the file information is sent.

- TYPE: 0x04
- HEADER_LENGTH: Length of the header
- HEADER: NEW_FILE
- DATA: *[FileName&FileSize&MD5SUM&ID]*

Where:

- FileSize: in bytes expressed in ASCII format
- MD5SUM: 32 characters of MD5SUM
- ID: Song identifier. It must be a random number from 0-999

Afterwards, the file will be sent.

- TYPE: 0x04
- HEADER_LENGTH: Length of the header
- HEADER: FILE_DATA]
- DATA: [ID&file_data]

Note that N frames of 256 bytes will need to be sent for each music file.

<u>Bowman ➔ Poole</u>

Successful MD5SUM check.

- TYPE: 0x05
- HEADER_LENGTH: Length of the header
- HEADER: CHECK_OK
- DATA: Empty

Incorrect MD5SUM check.

- TYPE: 0x05
- HEADER_LENGTH: Length of the header
- HEADER: CHECK_KO]
- DATA: Empty

LOGOUT

Bowman ➔ Poole

Frame to notify the server of a disconnection.

- TYPE: 0x06
- HEADER_LENGTH: Length of the header
- HEADER: EXIT
- DATA: [*userName*]

Poole ➔ Bowman

Frame OK disconnection.

- TYPE: 0x06
- HEADER_LENGTH: Length of the header
- HEADER: CONOK
- DATA: Empty

Frame KO disconnection.

- TYPE: 0x06
- HEADER_LENGTH: Length of the header
- HEADER: CONKO
- DATA: Empty

Procedure for detecting erroneous frames

If any of the processes in the HAL 9000 system receives a frame that does not correspond to any of the defined formats, it should send a return frame to be able to detect the error with the following frame:

- TYPE: 0x07
- HEADER_LENGTH: Length of the header
- HEADER: UNKNOWN
- DATA: Empty