eman ta zabal zazu

Universidad del País Vasco Euskal Herriko Unibertsitatea

BANK-ROBBER-RUN Shell Escape Room

Oihan Irastorza Carrasco
irastorza001@ikasle.ehu.eus

Amaia Aristegui Iturraran
aaristegui005@ikasle.ehu.eus

Andoni Velasco Gardoki
avelasco035@ikasle.ehu.eus

Esteban Arroyo
earroyo015@ikasle.ehu.eus

Asier Arronategui Basurto
aarronategui001@ikasle.ehu.eus

—

Subject: Introduction to Operating System
Degree: Computer Science
Academic Year: 2021/2022
School: Informatics Faculty UPV/EHU

May 22, 2022

Contents

Chapter 1

Introduction

Through this practice proposal, what we are going to achieve is the reinforcement of knowledge about all the theoretical concepts we have been learning in IOS subject. This project consists of carrying out/programming the basic functions of the terminal so as not to use the ones that come by default, with the practical purpose of making an **escape room** on the shell from scratch.

The story of the project focuses on a robbery, where the player will have to search for the secret stored in a bank vault. Through the server's file system, the complete map of a bank and its surroundings is simulated by means of directories. The vault is precisely one of these directories.

As in real life, the banks have the safe kept under high security. therefore, access to the Vault will be a challenge that the player will take on by **obtaining objects, interacting with the environment and unlocking new areas**.

In the different sections of this document, we will emphasize the design of the game and the code implemented to get to it.

The documentation will be divided into two large blocks that will be:

- **Set-up of the project**. It will be the part of the report that is aimed at explaining what the game consists of, the steps that must be taken, and the situations that the player will encounter as the gang's infiltrator.
- **Development of the project**. It will be the part dedicated to all the things related to the implementation, such as the code itself, specificaction and verification.

Thanks to the GitHub[4] version manager, the reader will always have access to the different versions of the code being discussed at any time, as well as compare different versions of the project. This repository also contains a project with all the features implemented in the escape room.

Chapter 2

Setting up the project

An escape rooms consist of a series of questions/actions which must be completed successfully to progress through the scenario and reach the final objective. For us, the goal is to create known (and new) terminal programs to be able to use them as commands for the player.

2.1 The general idea of the game

The escape room is called BANK ROBBER RUN and, as the name indicates, consists of carrying out a **robbery in a bank**. Is the player the one who is going to carry out the robbery and he/she will have to obtain the necessary permissions and tools to access the vault. These **permissions will be progressively obtained as necessary tools are collected** in order to open the doors to other rooms. But along the way, the player will encounter many obstacles that will make it difficult to obtain these tools.

By the end of the game, the player will have acquired the necessary tools to access the vault room and steal what is inside it.

Characters in the game will have their **dialogues altered based on actions taken by the player**. Some characters will be able to raise the alarm, others will be able to incapacitate the player, and others will be able to offer you valuable items.

2.2 Project scaffolding

Taking into account that we were going to develop a project that requires a considerable amount of assets and routines, it is important to have a clear organization of the working directory. Thus, the structure under /IOS-Project (this is, the root of the project) looks as follows:

- ***BANK-ROBBER-RUN.c***. The launcher of the *myshell0.c* file.
- **Game/**. The root directory of the project.
  - ***Directories/***. The root directory of the game.
    - ***Van/***. The starting point of the game. From here on, the rest of the child directories make up the game map.

    * ***Inv/***. The inventory of the player.

- ***function/***. This directory contains the programs for executing all the commands implemented in the game, as well as several auxiliary functions.
- ***assets/***. The vast majority of objects, tools and npc's are stored in this directory.
- ***myshell0.c***. The main file of the shell. Displays the main menu and allows the player to start a new game or load it since the last checkpoint. This is the file that handles the calls to the rest of the commands.
- ***defines.h***. Contains constants that are needed throughout the majority of the project.
- ***Makefile***. The file used for compiling and building the project.

2.3 Project file tree and game map

The simplified map of the game looks as follows:

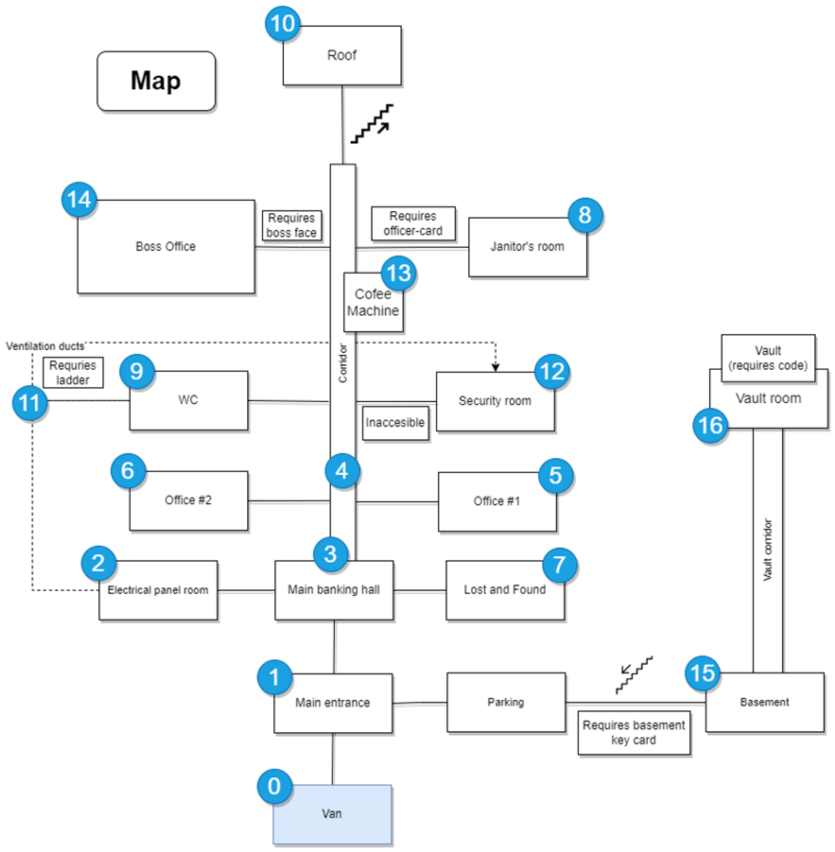
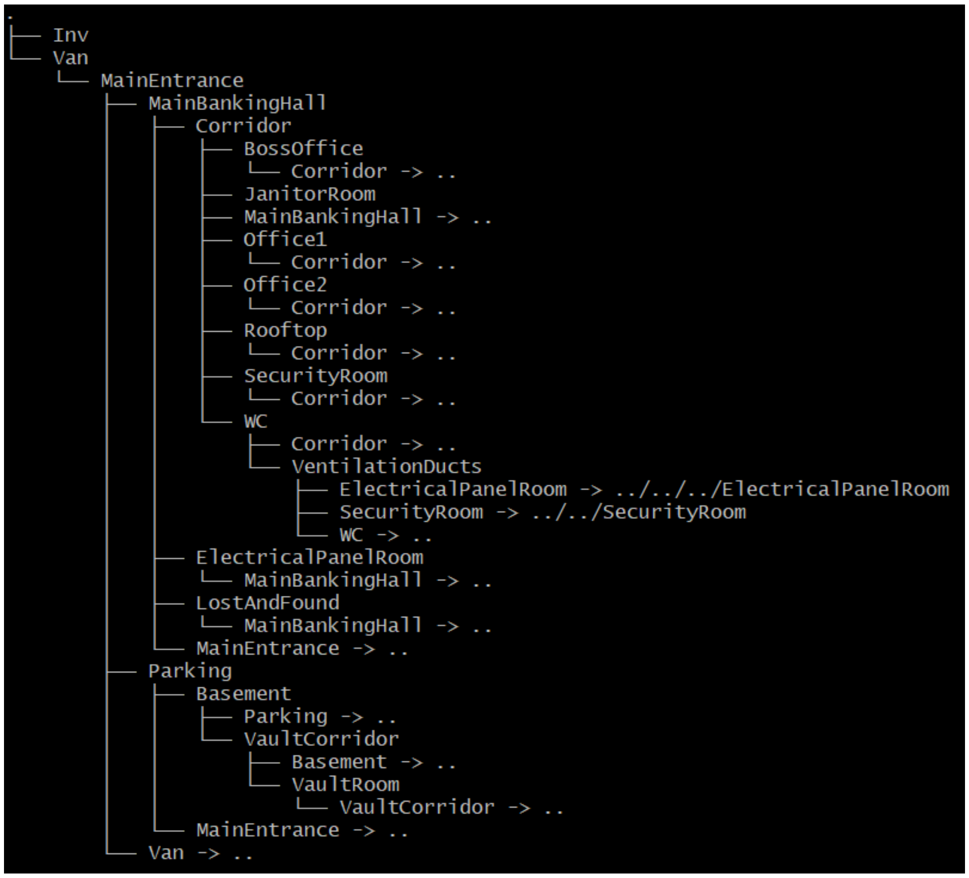


Figure 2.1: Simplified map

By executing the *tree -d* command over *Directories/* directory, we obtain the following output:



```
.
├── Inv
└── Van
    └── MainEntrance
        ├── MainBankingHall
        │   ├── Corridor
        │   │   ├── BossOffice
        │   │   │   └── Corridor -> ..
        │   │   ├── JanitorRoom
        │   │   ├── MainBankingHall -> ..
        │   │   ├── Office1
        │   │   │   └── Corridor -> ..
        │   │   ├── Office2
        │   │   │   └── Corridor -> ..
        │   │   ├── Rooftop
        │   │   │   └── Corridor -> ..
        │   │   ├── SecurityRoom
        │   │   │   └── Corridor -> ..
        │   │   └── WC
        │   │       ├── Corridor -> ..
        │   │       └── VentilationDucts
        │   │           ├── ElectricalPanelRoom -> ../../../ElectricalPanelRoom
        │   │           ├── SecurityRoom -> ../../SecurityRoom
        │   │           └── WC -> ..
        │   ├── ElectricalPanelRoom
        │   │   └── MainBankingHall -> ..
        │   ├── LostAndFound
        │   │   └── MainBankingHall -> ..
        │   └── MainEntrance -> ..
        ├── Parking
        │   ├── Basement
        │   │   ├── Parking -> ..
        │   │   └── VaultCorridor
        │   │       ├── Basement -> ..
        │   │       └── VaultRoom
        │   │           └── VaultCorridor -> ..
        │   └── MainEntrance -> ..
        └── Van -> ..
```

Figure 2.2: Directory tree

And while the user is playing the game, the map takes a more eye-catching design:

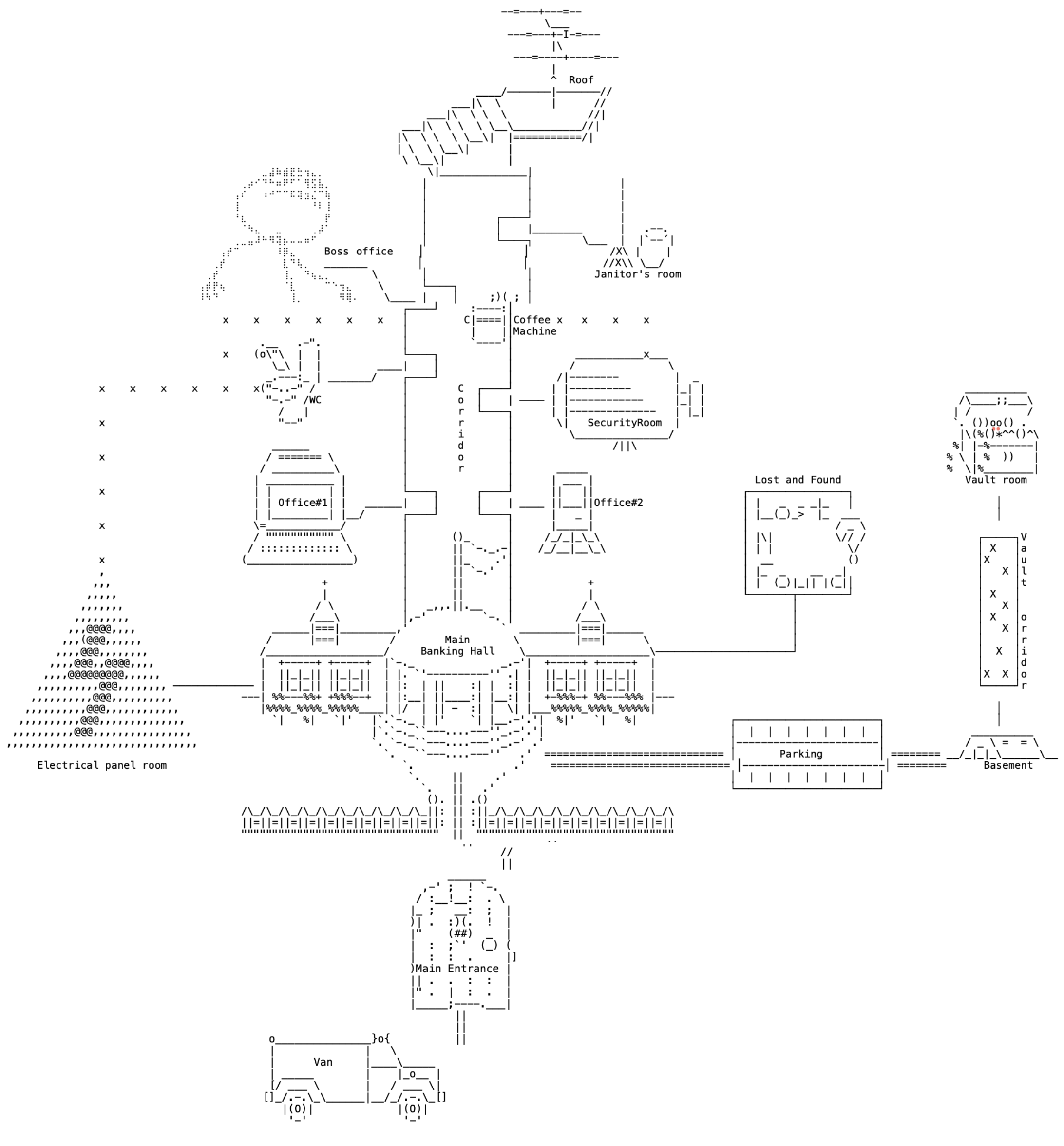


Figure 2.3: In-game map

To see the original picture, please refer to appendix E.

The map provides an overview of the bank and its surroundings. The player starts in the Van, together with his gang partner.The main part of the game resides inside the bank, in the locations between the main banking hall and the rooftop. Towards the end of the game, the player will access the basement, and from there eventually the vault room. When the player gets the secret of the game and exits the basement, the game ends[1].

[1]This does not mean that the game can only end at that point, as the player can lose the game due to certain negative actions that we will see later on.

2.4 Dialogues of the game

The dialogues of each character vary based on the decisions and actions taken by the user. This can lead directly from getting new items and opening new paths, to losing the game.

In the initial dialogue, your band mate will explain everything you need to know to start the game:

- The **main objective** of the robbery: to find the vault and get the secret it holds.
- The **initial situation** and a bit of background. The bank is expecting an electrician in 40 minutes, as the basement lights have broken down. It will be up to the player to impersonate the electrician to sneak into the bank.
- The **objects** with which the player starts the game.
- The **commands** that the player can use.
- The **first steps to be taken** by the player. Talk to the bank manager at the main entrance, go to the electrical panel with her, and then try to find an officer's card to move easily through the rest of the rooms.

To see all the dialogues in the game, please refer to Appendix B.

2.5 Storyline

For all possible paths and actions that the user can take, please refer to Appendix E.

The game starts in the van, along with your band mate, Robert. He will explain everything previously mentioned[2]. Next, the user will access the next location, the main entrance, where the Bank Manager Edurne will accompany him/her to the electrical panel room.

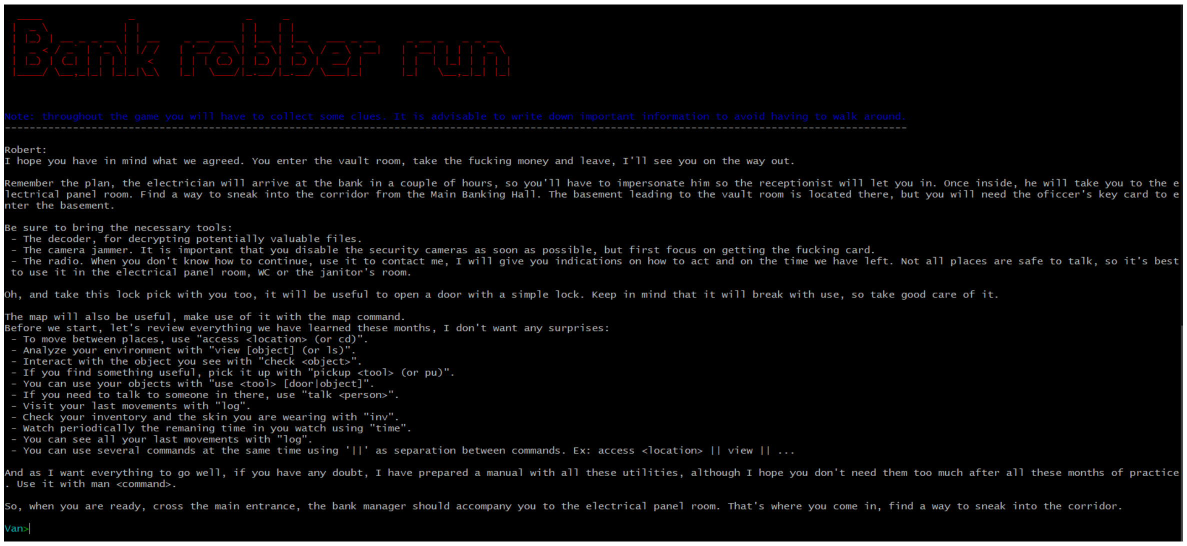
The prologue ends here. At this point, the player has to find a way to sneak into the Corridor, and can choose between two options to distract the guard in the main banking hall.

- Using the radio, Robert will offer to distract the guard, without telling the player exactly "how". Right after the conversation, the player will hear glass breaking in the main banking hall, and the guard will move to the main entrance to see what has happened.

  This action produces a handicap, and that is that the player will not be able to use the radio again to talk to his band mate, as he will be chased by the guard until then.
- Turn off the main banking hall lights using the electrical panel. This will make the guard unable to detect the player, and then the player will be able to enter the Corridor without any problems.

Upon first entering the corridor, the player will encounter an office worker, Matt. He will ask how he got into the corridor. Unless the answer selected by the player is very aggressive (in which case Matt will raise the alarm), the clerk will go to the restroom, carelessly leaving his office door open.

[2]The user can use the radio in the secure areas to talk to Robert. His dialogues will be updated as the player progresses through the game.



```
Bank robber run

Note: throughout the game you will have to collect some clues. It is advisable to write down important information to avoid having to walk around.
----------------------------------------------------------------------------------------------------------------------------------------------

Robert:
I hope you have in mind what we agreed. You enter the vault room, take the fucking money and leave, I'll see you on the way out.

Remember the plan, the electrician will arrive at the bank in a couple of hours, so you'll have to impersonate him so the receptionist will let you in. Once inside, he will take you to the e
lectrical panel room. Find a way to sneak into the corridor from the Main Banking Hall. The basement leading to the vault room is located there, but you will need the oficcer's key card to e
nter the basement.

Be sure to bring the necessary tools:
 - The decoder, for decrypting potentially valuable files.
 - The camera jammer. It is important that you disable the security cameras as soon as possible, but first focus on getting the fucking card.
 - The radio. When you don't know how to continue, use it to contact me, I will give you indications on how to act and on the time we have left. Not all places are safe to talk, so it's best
 to use it in the electrical panel room, WC or the janitor's room.

Oh, and take this lock pick with you too, it will be useful to open a door with a simple lock. Keep in mind that it will break with use, so take good care of it.

The map will also be useful, make use of it with the map command.
Before we start, let's review everything we have learned these months, I don't want any surprises:
 - To move between places, use "access <location> (or cd)".
 - Analyze your environment with "view [object] (or ls)".
 - Interact with the object you see with "check <object>".
 - If you find something useful, pick it up with "pickup <tool> (or pu)".
 - You can use your objects with "use <tool> [door|object]".
 - If you need to talk to someone in there, use "talk <person>".
 - Visit your last movements with "log".
 - Check your inventory and the skin you are wearing with "inv".
 - Watch periodically the remaning time in you watch using "time".
 - You can see all your last movements with "log".
 - You can use several commands at the same time using '||' as separation between commands. Ex: access <location> || view || ...

And as I want everything to go well, if you have any doubt, I have prepared a manual with all these utilities, although I hope you don't need them too much after all these months of practice
. Use it with man <command>.

So, when you are ready, cross the main entrance, the bank manager should accompany you to the electrical panel room. That's where you come in, find a way to sneak into the corridor.

Van>
```

Figure 2.4: Starting dialogue with Robert

This is when the office#1 will be accessible. Here inside there are several actions that the player can perform:

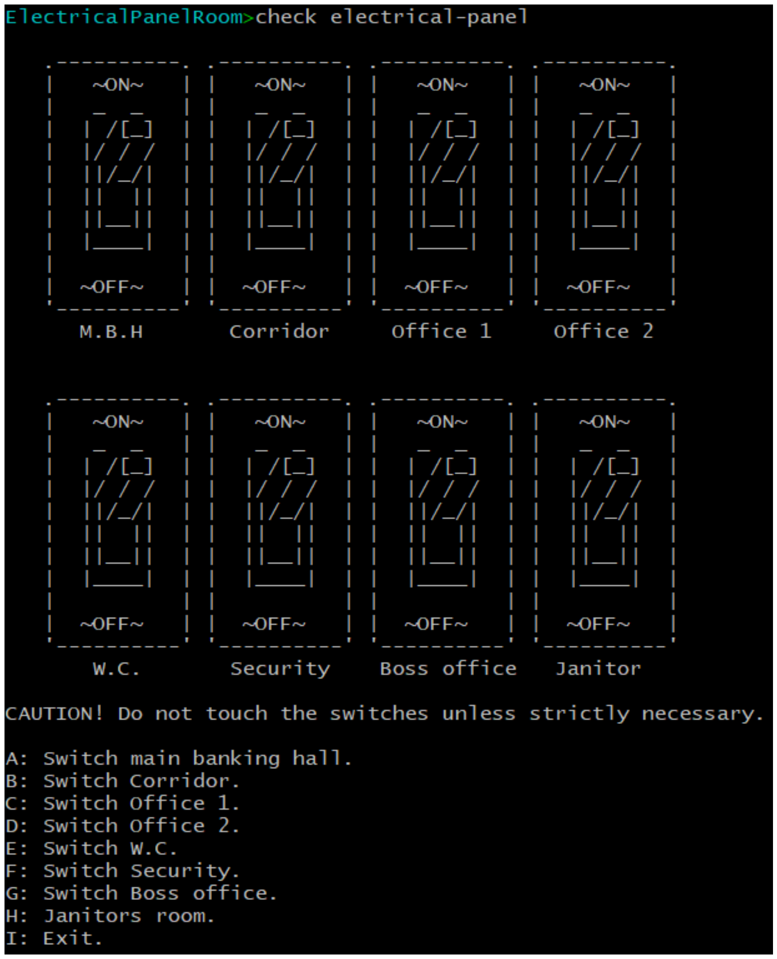
- Check cabinet. By interacting with the cabinet, the player can switch (indefinitely) his current costume between electrician and executive.
- Check desk drawers. Laxatives are found.
- Check computer. The email open, with several emails in the inbox. One of them is from the boss, and hints that he will be in the corridor in a while having a coffee.

When the player is about to leave the office, he will hear footsteps in the corridor. It is Matt, he is returning to the office. The player will then be presented with 3 different actions:

- Hide inside the cabinet. Matt will come in and sit for a while. He will see that he has the mail open, but he will not make a big deal about it. After a while he will go out for a cigarette. The player can then go back to talk to Matt at the Rooftop, but it will be of no further importance to the story.
- Hide behind the curtains. Matt the will see player's feet as he enters and will hit the alarm. Game Over.
- Wait behind the door and knockout Matt. Matt will fall to the office floor unconscious.

Back in the Corridor, the user can try to access Office2. A woman will respond from inside, and the story will not progress until the following requirements are completed:

- The office#2 lights are on. They can be turned on and off from electrical panel room.
- The player wears an executive skin.
- Laxatives have been used in the coffee machine.



```
ElectricalPanelRoom>check electrical-panel

   .----------. .----------. .----------. .----------.
   |   ~ON~   | |   ~ON~   | |   ~ON~   | |   ~ON~   |
   |    _ _   | |    _ _   | |    _ _   | |    _ _   |
   |   | /[_] | |   | /[_] | |   | /[_] | |   | /[_] |
   |   |/ / / | |   |/ / / | |   |/ / / | |   |/ / / |
   |   ||/_/| | |   ||/_/| | |   ||/_/| | |   ||/_/| |
   |   ||  || | |   ||  || | |   ||  || | |   ||  || |
   |   ||__|| | |   ||__|| | |   ||__|| | |   ||__|| |
   |   |____| | |   |____| | |   |____| | |   |____| |
   |          | |          | |          | |          |
   |  ~OFF~   | |  ~OFF~   | |  ~OFF~   | |  ~OFF~   |
   '----------' '----------' '----------' '----------'
      M.B.H       Corridor     Office 1     Office 2


   .----------. .----------. .----------. .----------.
   |   ~ON~   | |   ~ON~   | |   ~ON~   | |   ~ON~   |
   |    _ _   | |    _ _   | |    _ _   | |    _ _   |
   |   | /[_] | |   | /[_] | |   | /[_] | |   | /[_] |
   |   |/ / / | |   |/ / / | |   |/ / / | |   |/ / / |
   |   ||/_/| | |   ||/_/| | |   ||/_/| | |   ||/_/| |
   |   ||  || | |   ||  || | |   ||  || | |   ||  || |
   |   ||__|| | |   ||__|| | |   ||__|| | |   ||__|| |
   |   |____| | |   |____| | |   |____| | |   |____| |
   |          | |          | |          | |          |
   |  ~OFF~   | |  ~OFF~   | |  ~OFF~   | |  ~OFF~   |
   '----------' '----------' '----------' '----------'
       W.C.       Security    Boss office   Janitor

CAUTION! Do not touch the switches unless strictly necessary.

A: Switch main banking hall.
B: Switch Corridor.
C: Switch Office 1.
D: Switch Office 2.
E: Switch W.C.
F: Switch Security.
G: Switch Boss office.
H: Janitors room.
I: Exit.
```

Figure 2.5: Electrical panel

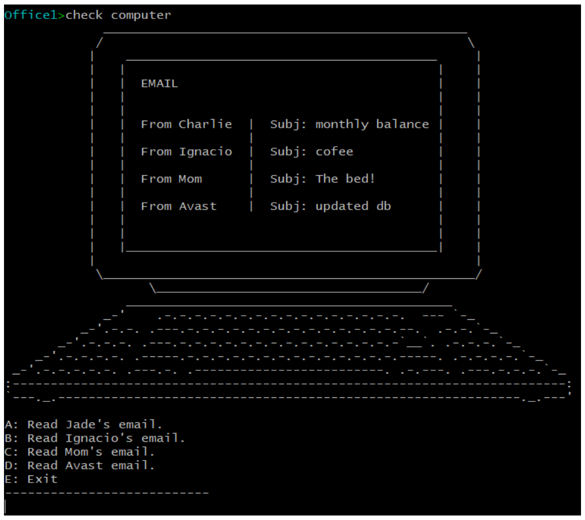
Once the requirements have been met, the next time you try to access office#2, Jade, the officer, will come out to assist you. The player can then invite her for coffee, and Jade will accept. Jade will then run off to the restroom, leaving the office door open. The player can then access office#2.

Once inside the office, the player can interact with the laptop on the table, and the following options will be presented:

- Consult the database. Contains information on all employees. Requires password.
- A mail from the lost and found department. This mail informs that the employee named Anton has lost his access card, and he can come and pick it up if he identifies himself with his ID number.
- An encrypted file named "pass.txt", that contains the password for the database.

If the user uses the decoder on the laptop, he can try to decode the message. He will be offered to use 3 different decoders:

- Base64



```
Office1>check computer

    EMAIL

    From Charlie  |  Subj: monthly balance
    From Ignacio  |  Subj: cofee
    From Mom      |  Subj: The bed!
    From Avast    |  Subj: updated db

A: Read Jade's email.
B: Read Ignacio's email.
C: Read Mom's email.
D: Read Avast email.
E: Exit
--------------------------
```

Figure 2.6: Matt's computer

- Rot13
- Morse

If Base64 is selected, the password of the database will be displayed. The user can then access the database and get Anton's employee information, and subsequently obtain the access card in lost and found department.

In order to inquire about the access card at the lost and found department, two requirements must be met:

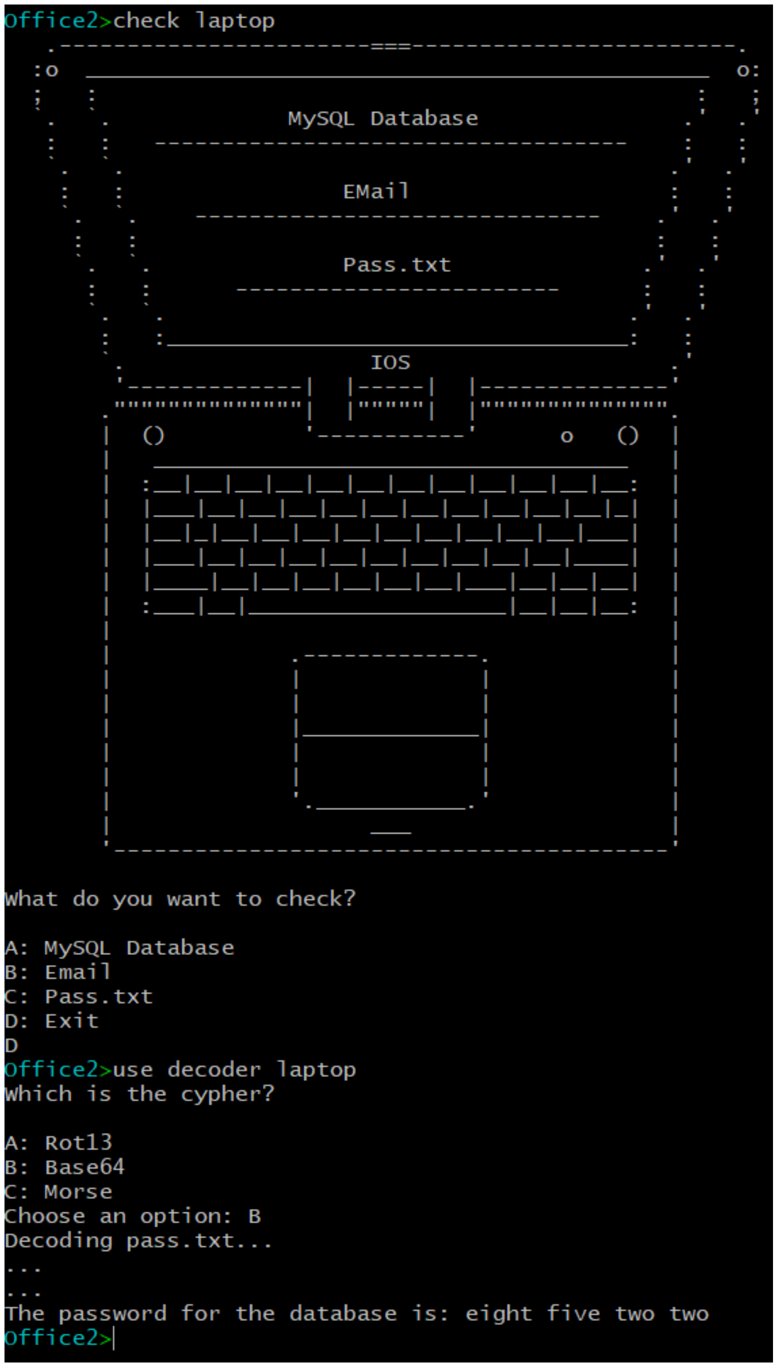
- Be dressed with executive skin.
- Have seen the lost and found mail on the laptop.

When the user arrives at this department, he/she can talk to Veronica to retrieve the access card. She has three chances to give the correct identifier. In case of success, the user will receive the access card. Otherwise, the game ends, as the player will have lost the way to get an access card.

After obtaining the card, the next objective is to find a way to get into the security room and hack the cameras. For that, firstly the access to the janitor's room will be unlocked, thanks to the card obtained in lost and found department. Inside we can find two things:

- Bleach. Useful for sabotaging the coffee machine.
- Ladder.

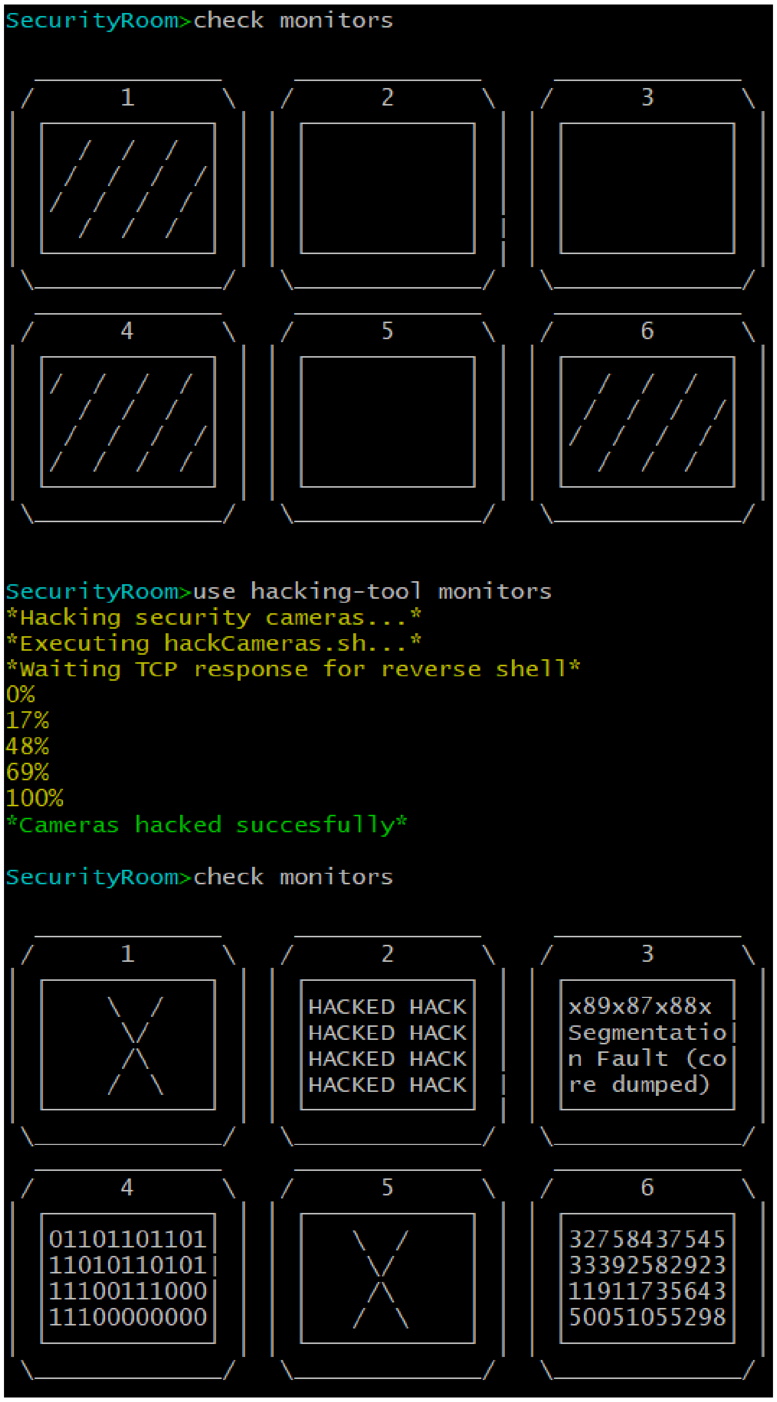
If the ladder is used in the WC's ventilation ducts, it will unlock a bidirectional path between the electrical panel room and the WC, and an unidirectional path from the WC to the security room. This is the only path to access to security until the player accesses for first time this room and opens the door from inside.



```
Office2>check laptop

What do you want to check?

A: MySQL Database
B: Email
C: Pass.txt
D: Exit
D
Office2>use decoder laptop
Which is the cypher?

A: Rot13
B: Base64
C: Morse
Choose an option: B
Decoding pass.txt...
...
...
The password for the database is: eight five two two
Office2>
```

Figure 2.7: Jade's laptop

When the player sneaks into the security room, he will encounter Javier, the sleeping guard. It will remain asleep even if the player tries to talk to him. The player will also find the security camera monitors, which can be hacked by using the hacking-tool. Finally, there is a picture of the boss that can be collected for using it as a bio metrical key for entering the boss office, and also a night-vision-goggles, which will be necessary in the basement, taking into account that the electrical problem takes place there.



```
SecurityRoom>check monitors

      1              2              3

      4              5              6

SecurityRoom>use hacking-tool monitors
*Hacking security cameras...*
*Executing hackCameras.sh...*
*Waiting TCP response for reverse shell*
0%
17%
48%
69%
100%
*Cameras hacked succesfully*

SecurityRoom>check monitors

      1              2              3
                HACKED HACK    x89x87x88x
                HACKED HACK    Segmentatio
                HACKED HACK    n Fault (co
                HACKED HACK    re dumped)

      4              5              6
 01101101101                   32758437545
 11010110101                   33392582923
 11100111000                   11911735643
 11100000000                   50051055298
```

Figure 2.8: Hacked security cameras

The game will not advance until the following two conditions are met:

- The player has used bleach in the coffee machine.

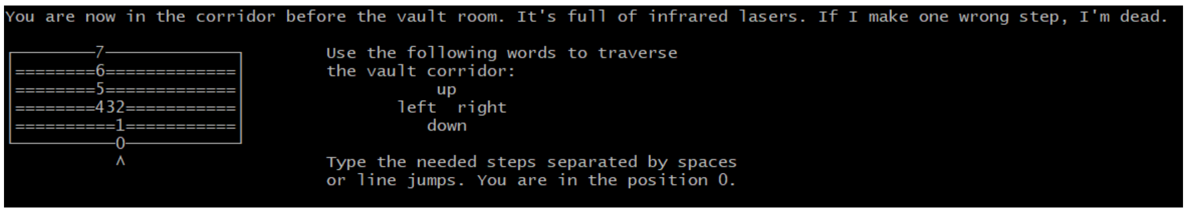
- The cameras have been hacked.

Once both requirements are met, the next time the corridor is accessed, the director will be coming out of his office towards the coffee machine, and will ask the player to come forward to talk. Halfway through the conversation, the director will start to feel sick, and will run to the WC.

If the player enters the bathroom, player will find the director dead on the floor and the basement card next to him.

The last part of the game is about accessing the vault room and getting hold of the secret. To do this, the player first has to access the director's office and examine the picture on the desk. On it are his wife and newborn daughter. The date on the photo is the password to the safe.

The player can now leave the bank and go to the basement, and then to the vault corridor. To enter it is necessary to have used the night vision goggles. Once inside the vault corridor, an image will be displayed with a room full of lasers and the path to take. The user will have to write down the steps to take one by one. If a mistake is made, the user will automatically return to the previous room.



```
You are now in the corridor before the vault room. It's full of infrared lasers. If I make one wrong step, I'm dead.

┌────────7───────────────┐          Use the following words to traverse
│========6=============  │          the vault corridor:
│========5=============  │                     up
│========432===========  │                 left  right
│==========1===========  │                    down
└──────────0─────────────┘
           ^                        Type the needed steps separated by spaces
                                    or line jumps. You are in the position 0.
```

Figure 2.9: Entering to the vault corridor

When the vault room is finally accessed, the battery of the night vision goggles will be depleted. This means that the player will not be able to see the way back, and will have to look for an alternative way to retrace his steps (the steps taken can be checked with the *log* command).

Finally, in the vault room, the player will be able to interact with the vault, and will be asked for a password. If the player enters the numbers of the date of the image found in the boss's office, he will get the secret.

The game ends when the player manages to retrace his steps in the vault corridor.

Chapter 3

Development of the project

3.1 Compile and build. Makefile

Commands are individually tested for correct operation and error proofing. However, we have built a Makefile to facilitate the construction of our project, so we can compile our routines in a faster and more efficient way.

Makefile

```
# Makefile

# Compiler flags
CC = gcc -DFUNCTION -rdynamic
CC1 = gcc -rdynamic
CPPFLAGS = -MMD
CFLAGS = -Wall -Wextra -std=gnu99 -g

# Functions
SRC = myshell0.c function/cd.c function/chmod.c function/resetGame.c function/storeMoves.c function/Leave.c
OBJ = ${SRC:.c=.o}
DEP = ${SRC:.c=.d}

# Indepependent processes
SRC1 = function/view.c
OBJ1 = ${SRC1:.c=.o}
DEP1 = ${SRC1:.c=.d}

SRC3 = function/inv.c
OBJ3 = ${SRC3:.c=.o}
DEP3 = ${SRC3:.c=.d}

SRC4 = function/Leave.c
OBJ4 = ${SRC4:.c=.o}
DEP4 = ${SRC4:.c=.d}

SRC5 = function/pickup.c
OBJ5 = ${SRC5:.c=.o}
DEP5 = ${SRC5:.c=.d}

SRC6 = function/talk.c
```

```
OBJ6 = ${SRC6:.c=.o}
DEP6 = ${SRC6:.c=.d}

SRC7 = function/pwd.c
OBJ7 = ${SRC7:.c=.o}
DEP7 = ${SRC7:.c=.d}

SRC8= function/chmod.c
OBJ8 = ${SRC8:.c=.o}
DEP8 = ${SRC8:.c=.d}

SRC9= function/use.c
OBJ9 = ${SRC9:.c=.o}
DEP9 = ${SRC9:.c=.d}

SRC10= ../BANK-ROBBER-RUN.c
OBJ10 = ${SRC10:.c=.o}
DEP10 = ${SRC10:.c=.d}

SRC11= function/check.c
OBJ11 = ${SRC11:.c=.o}
DEP11 = ${SRC11:.c=.d}

SRC12= function/man.c
OBJ12 = ${SRC12:.c=.o}
DEP12 = ${SRC12:.c=.d}


all: view myshell0 inv leave pickup talk pwd chmod use ../BANK-ROBBER-RUN check man

-include ${DEP1}
view: ${SRC1}
$(CC1) -o $@ $^ $(CFLAGS)

-include ${DEP}
myshell0: ${OBJ}
	$(CC) -o $@ $^ $(CFLAGS) -pthread -lpthread -fPIE

-include ${DEP3}
inv: ${SRC3}
	$(CC1) -o $@ $^ $(CFLAGS)

-include ${DEP4}
leave: ${SRC4}
	$(CC1) -o $@ $^ $(CFLAGS)

-include ${DEP5}
pickup: ${SRC5}
	$(CC1) -o $@ $^ $(CFLAGS)

-include ${DEP6}
talk: ${SRC6}
	$(CC1) -o $@ $^ $(CFLAGS)

-include ${DEP7}
pwd: ${SRC7}
```

```
        $(CC1) -o $@ $^ $(CFLAGS)

    -include ${DEP8}
    chmod: ${SRC8}
        $(CC1) -o $@ $^ $(CFLAGS)

    -include ${DEP9}
    use: ${SRC9}
        $(CC1) -o $@ $^ $(CFLAGS) -D_DEFAULT_SOURCE -D_BSD_SOURCE

    -include ${DEP10}
    BANK-ROBBER-RUN: ${SRC10}
        $(CC1) -o $@ $^ $(CFLAGS)

    -include ${DEP11}
    check: ${SRC11}
        $(CC1) -o $@ $^ $(CFLAGS)

    -include ${DEP12}
    man: ${SRC12}
        $(CC1) -o $@ $^ $(CFLAGS)

    clean:
    @rm -f ${OBJ} ${DEP} myshell0 ${DEP1} ${OBJ1} view ${DEP2} ${OBJ2} cd ${DEP3} ${OBJ3} ↙
        ↳ inv ${DEP4} ${OBJ4} Leave ${DEP5} ${OBJ5} pickup ${DEP6} ${OBJ6} talk ${DEP7} ${↙
        ↳ OBJ7} pwd ${DEP8} ${OBJ8} chmod ${DEP9} ${OBJ9} use ${DEP10} ${OBJ10} ../BANK-↙
        ↳ ROBBER-RUN ${DEP11} ${OBJ11} check ${DEP12} ${OBJ12} man
# END
```

3.2 Applied theoretical concepts

As this is a project for the IOS course, the objective is to apply the concepts worked on theoretically in the previous months. Throughout the different routines you can see all kinds of applications to these concepts. We could divide the most important concepts and their respective applications in the implementation as follows:

- **I/O and files API**[3]. Working with files is present in a large part of the implemented commands. A very clear example would be in the "talk" command, where the file of a game character is opened and its first byte is read, which is the current state of the character.

```
    // Get npc path
    strncpy(current, argv[2], PATH_MAX);
    strcat(current, "/assets/npc/");
    strcat(current, npc);
    strcat(current, ".npc");

    // Check if npc is the current room
    if (access(current, W_OK) == -1) {
            write(2, "\033[31mThere is no one by that name in this room.\n\033[37m", ↙
                ↳ strlen("\033[31mThere is no one by that name in this room.\n\033[37m↙
                ↳ "));
            return -1;
    }
```

```c

    fd = open(current, O_RDWR);
    if (fd == -1) return 1;

    // Read initial state of the npc
    read(fd, &state[0], 1);
```

Another function related to file processing would be *searchTalk*, an auxiliar function of *talk* command which takes as input parameter a two-character word and a file-descriptor, and returns the offset of the first occurrence of that word using *LSEEK* system call.

```c
/**
* Function: searchTalk
* --------------------
* Returns the byte offset of the provided dialogue
* @param word the code to be searched in the file (e.g 2C)
* @param file the file where to search
* @return 0 => npc keeps talking. 1 => stop conversation. 2 => game over
*/
int searchTalk(char *word, int file){
        char c;
        int readed;

        while ((readed = read(file, &c, 1)) == 1) {
                if (c == word[0]) {
                        read(file, &c, 1);
                        if (c == word[1]) return lseek(file, 0, SEEK_CUR);
                }
        }

        return -1;
}
```

- **User management API**.Permission management in our game is strictly controlled from the start of each game. In fact, in the **resetGame** function we make sure that each directory has its corresponding permissions (with the *chmod* commands we have implemented ourselves). A fragment of this function would be:

```c
        // Set readonly permissions for player to those directories that need a key or tool to be opened
        if (fork() == 0)
        {
                execlp("./chmod","./chmod","./Directories/Van/MainEntrance/Parking/Basement","0066", NULL);
                printf("\033[31mError changing Basement permissions: %s.\n\033[37m", strerror(errno));
                exit(0);
        }
        else wait(NULL);

        if (fork() == 0)
        {
                execlp("./chmod","./chmod","./Directories/Van/MainEntrance/MainBankingHall/Corridor/BossOffice","0066", NULL);
                printf("\033[31mError changing BossOffice permissions: %s.\n\033[37m", strerror(errno));
                exit(0);
        }
        else wait(NULL);

        if (fork() == 0)
        {
                execlp("./chmod","./chmod","./Directories/Van/MainEntrance/
                    MainBankingHall/Corridor/JanitorRoom","0066", NULL);
                printf("\033[31mError changing JanitorRoom permissions: %s.\n\033[37m", 
                    strerror(errno));
                exit(0);
        }
```

Another interesting example is in the *use* commands. We read a struct from a file composed by door-¿key_that_opens_the_door, and we grant read, write and execute permissions to the player with out *chmod* command.

```
if (found) {
// Check if key used by player is the one that opens the target door
if (strcmp(tool, keyDoor[keyDoorI].key) == 0) {
// Change door permissions
strncpy(commandPath, argv[0], PATH_MAX);
strncat(commandPath, "/../chmod", PATH_MAX);
realpath(dir->d_name, roomPath);

if (fork() == 0) {
        execlp(commandPath, commandPath, roomPath, "0755", (char *) NULL);
        if (errno != 0) {
                printf("\033[31mProblem unlocking the door: %s.\n\033[37m", strerror
                    (errno));
                free(commandPath);
                return 1;
        }
} else wait(NULL);

write(1, "\x1b[32mDoor unlocked.\x1b[0m\n", strlen("\x1b[32mDoor unlocked.\x1b[0m\
    n"));
} else write(1, "\033[31mWrong key.\n\033[37m", strlen("\033[31mWrong key.\n
    \033[37m"));

} else write(1, "This door doesn't require any key to be opened.\n", strlen("This 
    door doesn't require any key to be opened.\n"));
```

Unfortunately we have no permissions to create and manipulate users and group in the current environments, so we have limited ourselves to play with file and directory permissions.

- **Memory management API**. Most of the time, as programmers of the game, we know how much memory we are going to use the variables (or have an approximation), so we have not made too much use of dynamic memory[5]. An example might be when dealing with the stat command, where we have to store space for its runtime buffer. The following piece of code is taken from the view command, and it is used to check the contents of a directory.

```c
struct stat *buf = malloc(sizeof(struct stat));

...

DIR *d =opendir(".");
while((dir =readdir(d)) !=NULL)
{
        stat(dir->d_name,buf);
        if (S_ISDIR(buf->st_mode))
        {
                if (dir->d_name[0] != '.' && dir->d_name[strlen(dir->d_name)-1] != '~')
                {
                        strcat(room," ");
                        strcat(room,dir->d_name);
                        strcat(room,"\n");
                }
        }

...
```

A good point to take into account is the use of fixed-size string functions. The library *strings* is vulnerable to buffer overflow by default. For example, when using strcpy, you can copy more bytes than expected by the buffer argument, making this possible to change the return value of the routines and inject malware. Fortunately, the same library introduced the fixed-size functions (like *strncpy* or *strncat*). We have decided to use this functions whenever it is necessary (this is, almost always). The following piece of code is extracted from *myshell0.c* and is the code used when executing *log* command.

```c
else if(strcmp(argv[0],"log")==0){
    char *logFile = malloc(PATH_MAX);

    ...

    strncpy(logFile, root, PATH_MAX);
    strcat(logFile, "/../moves.txt");

    ...
```

- **Process and thread[1] management API** Considering that we have to simulate a shell, it is clear that we have to apply the *fork-exec*[2] structure to almost all commands. As we have already seen some examples above, we will not talk much more about it.

  As for the threads, we have implemented one that interprets the commands entered by the user, and another one that controls the game timer.

  First, the main function creates a single thread which runs the main program of the shell.

```c
void *beginning() {
    begin();
    pthread_exit(NULL);
}

int main() {
    pthread_t ptid;
    pthread_create(&ptid, NULL, &beginning, NULL);
    pthread_exit(NULL);
}
```

Then, the main routine will call *Time* function, which will attach a new thread to *Time1* function.

```
int begin() {
    pthread_detach(pthread_self());

    ...

    startTime=clock(); // start clock
    time_left=count_down_time_in_secs-seconds; // update timer
    Time();

    ...
}

void Time(){
    pthread_t ptid;
    pthread_create(&ptid, NULL, &Time1, NULL);
}

void* Time1(){
    while(cond){
            pthread_detach(pthread_self());
            countTime=clock();
            milliseconds=countTime-startTime;
            seconds=(milliseconds/(CLOCKS_PER_SEC))-(minutes*60);
            tempseconds=(milliseconds/(CLOCKS_PER_SEC));
            minutes=(milliseconds/(CLOCKS_PER_SEC))/60;
            hours=minutes/60;
            time_left=count_down_time_in_secs-tempseconds;
    }
    pthread_exit(NULL);
}
```

- **Pipes management API**. In our game we give the option to concatenate commands with the symbols "||". The *myshell0*'s routine *coutpipe* is in charge of separating each command and executing them one by one. You can see the complete code of the routine in the source code of the game.

3.3 Command specification and verification

3.3.1 Specification document

The specification made is the same as the one shown with the in-game man command. If any command has different ways of use, these will be clearly specified.

You can find all specifications in appendix C.

3.3.2 Code

The complete code with the dialogues and the history of the game is all implemented in the following link.

This code has gone through an exhaustive control of errors and warnings so that the user enjoys the game in its entirety and without getting lost.

As mentioned before, the code is available in the repository of the game, but also in the appendix A.

3.3.3 Verification document

In this section there will be three sub sections that will help us to validate the functions performed. Using these tests, what we want is to force the functions and take them to the extremes to validate them safely:

- ERROR, we will look for the programmed error to occur. That is, the one that has been treated when making the code of the function. This error when jumping in the program will be shown in red to be able to differentiate from the normal dialog without error.
- VALIDATION, in this validation what we will look for is the dreamed execution, the ideal one so that the program works correctly.
- UNCONTROLLED ERROR, is the error that jumps without having it controlled in the code. That is, the unscheduled failure. This error will not be shown in red since it is not treated, it is a terminal error. The aim is to reduce as much as possible this kind of errors.

All the verification's are documented under the appendix D.

Chapter 4

Conclusions

In conclusion, we can say without any doubt that this project has introduced us to many aspects of an operating system. In particular say that we learn about:

- Manage user permissions on files, directories and symbolic links.
- File system and I/O.
- Memory and thread management.
- Device independence.
- C language basics, such as pointer managing, string formatting...
- The use of an vast amount of system calls (open, close, link unlink, symlin, opendir, opendir, closedir, rmdir, read, write, fork...)
- Interprocess communication, with named and unnamed pipes.
- ...

The work has been equally distributed, and the attendance of the participants has been active and constant. We have been able to manage the implementation and documentation in a synchronized manner, which has allowed us to organize ourselves more efficiently. Even if we had to simplify the storyline a bit, most of the proposed ideas have been successfully implemented (the story, the dialogues, the ASCII-ART, the commands).

The project as a whole has been an enriching practice to work on all theory given in the first months of class.

Bibliography

[1] Eric C Cooper and Richard P Draves. “C threads”. In: (1988).

[2] GeeksForGeeks. *Difference between fork() and exec()*. Apr. 26, 2022. URL: `https://www.geeksforgeeks.org/difference-fork-exec/`.

[3] GNU. *File System Interface*. URL: `https://www.gnu.org/software/libc/manual/html_node/File-System-Interface.html`.

[4] IOS group. *Bank Robber Run*. May 20, 2022. URL: `https://github.com/Botxan/Bank-Robber-Run`.

[5] UC3M. *System call for memory management in C*. Feb. 8, 2012. URL: `http://www.it.uc3m.es/pbasanta/asng/course_notes/dynamic_memory_en.html`.

.1 Appendix A. Source code

Includes all the code developed throughout the project

.2 Appendix B. Dialogues

Includes all the dialogues of all the characters in the game.

.3 Appendix C. Specification documents

Definition of each command implemented.

.4 Appendix D. Verification documents

Tests of all the commands implemented.

.5 Appendix E. Bank map and storyline

All the possible actions that may take place inside the game.