COMP 304 PROJECT 1 REPORT

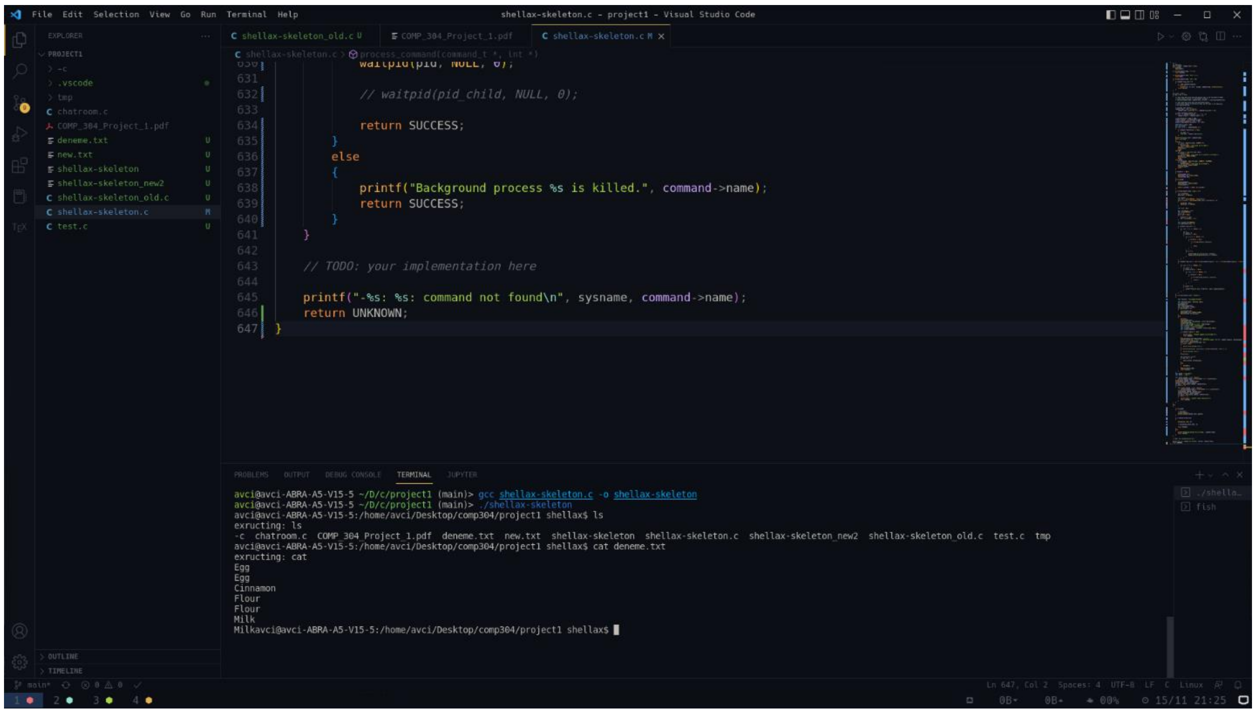
Ali AVCI 0072779

PART 1:

In this part of the project, using execv, common unix command is implemented for the shellax terminal. According to the execv page, using execv requires full path of the command as a first argument and as a second argument, a null terminated array which consists of elements that are, first one is command name, and the other ones are arguments for command.[1] The other part of the part 1, is implementation of the background process. Using & symbol at the end of the command means that the process executes in background and while it is executing in the background, I can type new commands to shell.

Screenshots of the running shellax terminal for the first part:

For executing common commands using execv: (ls and cat command)

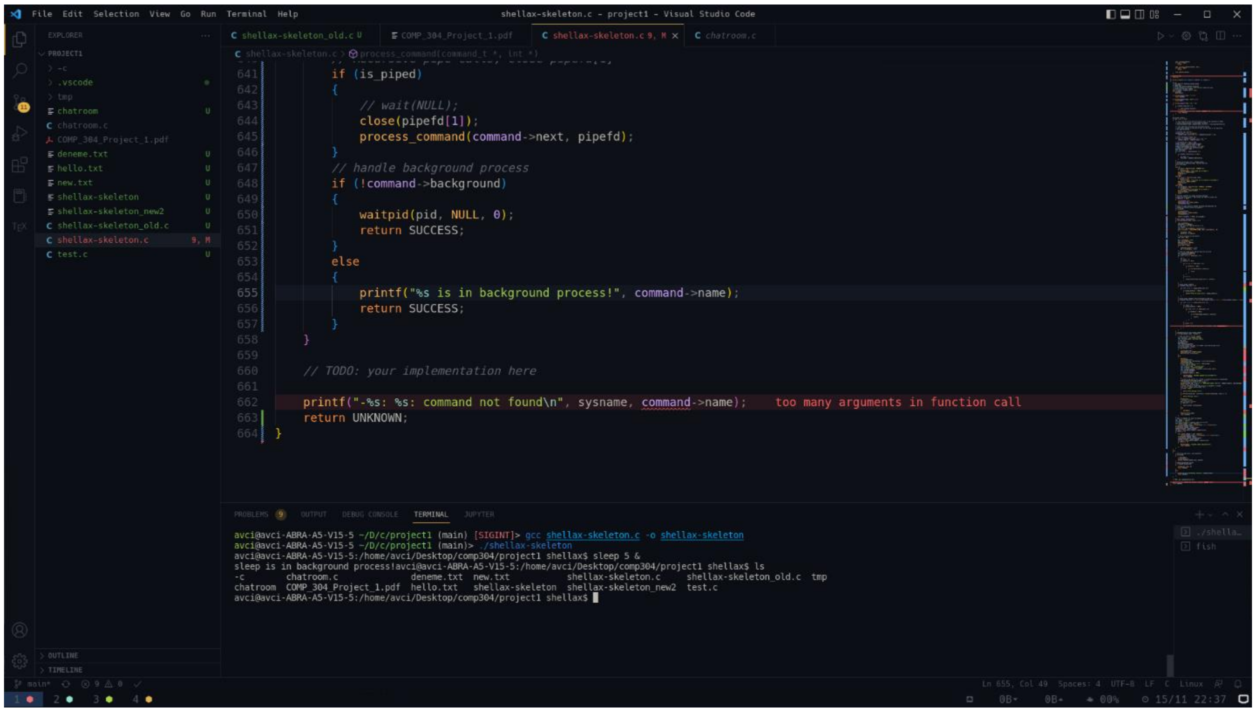


```c
            waitpid(pid, NULL, 0);

            // waitpid(pid_child, NULL, 0);

            return SUCCESS;
        }
        else
        {
            printf("Background process %s is killed.", command->name);
            return SUCCESS;
        }
    }

    // TODO: your implementation here

    printf("-%s: %s: command not found\n", sysname, command->name);
    return UNKNOWN;
}
```

```
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)> gcc shellax-skeleton.c -o shellax-skeleton
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)> ./shellax-skeleton
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ ls
exructing: ls
-c  chatroom.c  COMP_304_Project_1.pdf  deneme.txt  new.txt  shellax-skeleton  shellax-skeleton.c  shellax-skeleton_new2  shellax-skeleton_old.c  test.c  tmp
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ cat deneme.txt
exructing: cat
Egg
Egg
Cinnamon
Flour
Flour
Milk
Milkavci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$
```

Figure 1, Executing command commands in shellax terminal

For Background Process of commands:



```c
        if (is_piped)
        {
            // wait(NULL);
            close(pipefd[1]);
            process_command(command->next, pipefd);
        }
        // handle background process
        if (!command->background)
        {
            waitpid(pid, NULL, 0);
            return SUCCESS;
        }
        else
        {
            printf("%s is in background process!", command->name);
            return SUCCESS;
        }
    }

    // TODO: your implementation here

    printf("-%s: %s: command not found\n", sysname, command->name);     too many arguments in function call
    return UNKNOWN;
}
```

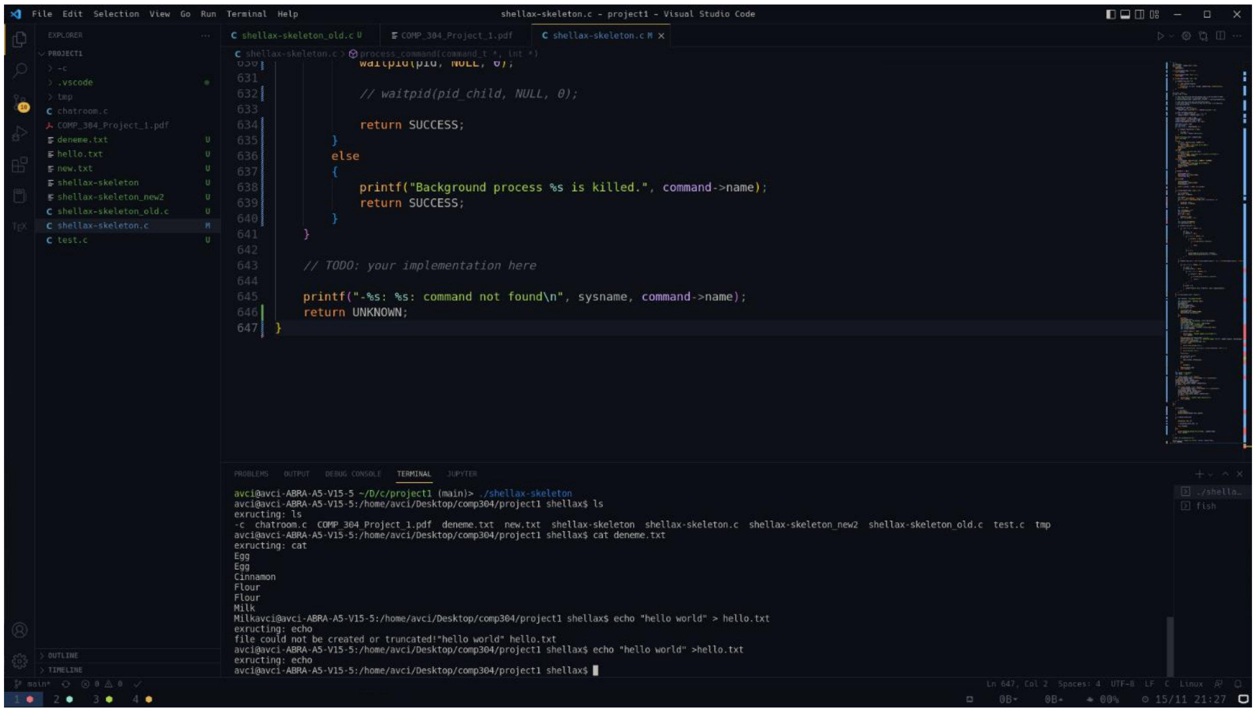
```
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main) [SIGINT]> gcc shellax-skeleton.c -o shellax-skeleton
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)> ./shellax-skeleton
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ sleep 5 &
sleep is in background process!avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ ls
-c        chatroom.c           deneme.txt  new.txt           shellax-skeleton.c     shellax-skeleton_old.c  tmp
chatroom  COMP_304_Project_1.pdf  hello.txt   shellax-skeleton  shellax-skeleton_new2  test.c
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ 
```

Figure 2, Background Processes of Commands

PART 2:

In this part of the project, first I/O redirection is implemented then program piping is handled for Shellax. For I/O redirection, dup2 system call is used. Direction enum is defined and for all direction cases such as in, out and append, value is assigned. Then indexing the command->redirects, the filename is taken. Then according to the direction mode, system calls are used. In case IN, since the argument is taken from the STDIN, using dup2 system call the STDIN file descriptor is duplicated.[2] In case OUT, since I want to write result to filename instead of STDOUT, I used dup2 system call to duplicate STDOUT file descriptor. Also, since the filename doesn't exist, I create the file using creat system call.[3] For append case again I duplicate STDOUT file descriptor.

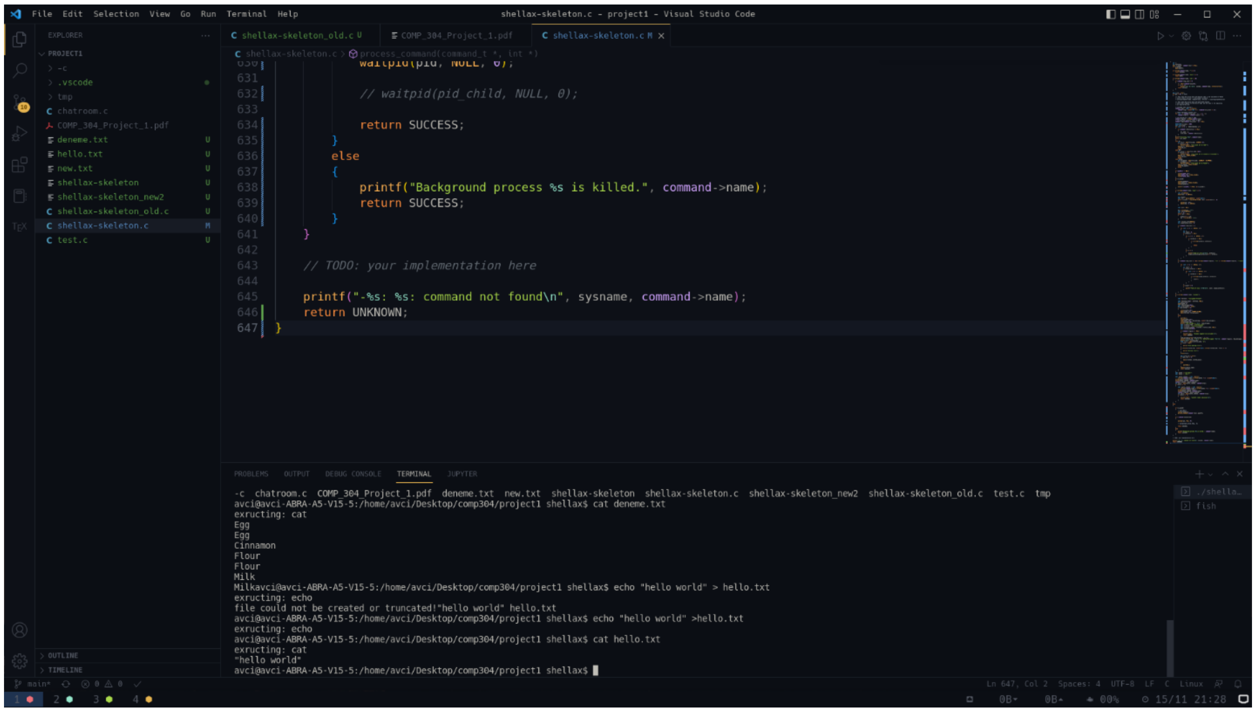
For program piping, I added int * pipefd_r parameter for pipe array, to process command function. I checked if the next command exists after the first command, so if it exists, which means that the commands are piped. I used pipefd_r as a reading pipe, which means command takes argument from STDIN, then I used pipefd for writing the result of previous command to pipefd[1] instead of STDOUT.[4] Then in the parent process, I used recursive calls.



```c
            waitpid(pid, NULL, 0);

            // waitpid(pid_child, NULL, 0);

            return SUCCESS;
        }
        else
        {
            printf("Background process %s is killed.", command->name);
            return SUCCESS;
        }
    }

    // TODO: your implementation here

    printf("-%s: %s: command not found\n", sysname, command->name);
    return UNKNOWN;
}
```

```
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)> ./shellax-skeleton
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ ls
exructing: ls
-c  chatroom.c  COMP_304_Project_1.pdf  deneme.txt  new.txt  shellax-skeleton  shellax-skeleton.c  shellax-skeleton_new2  shellax-skeleton_old.c  test.c  tmp
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ cat deneme.txt
exructing: cat
Egg
Egg
Cinnamon
Flour
Flour
Milk
Milkavci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ echo "hello world" > hello.txt
exructing: echo
file could not be created or truncated!"hello world" hello.txt
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ echo "hello world" >hello.txt
exructing: echo
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ █
```

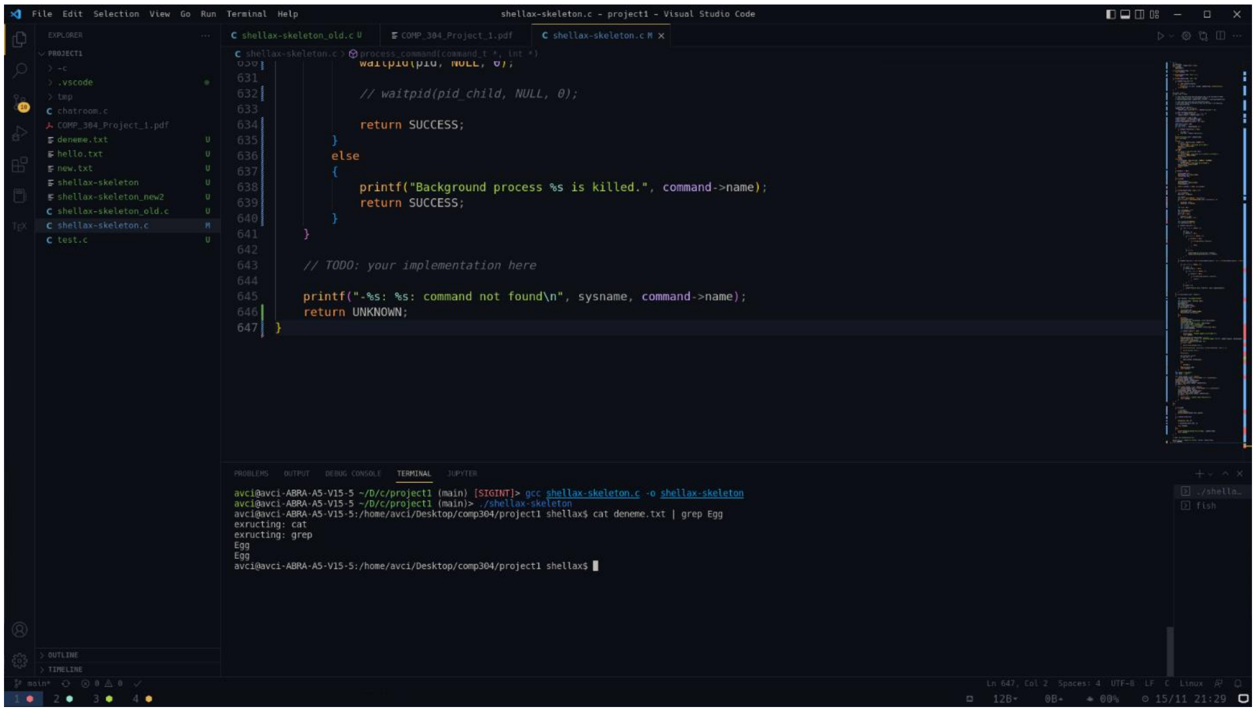
Figure 3, I/O redirection



```c
            waitpid(pid, NULL, 0);

            // waitpid(pid_child, NULL, 0);

            return SUCCESS;
        }
        else
        {
            printf("Background process %s is killed.", command->name);
            return SUCCESS;
        }
    }

    // TODO: your implementation here

    printf("-%s: %s: command not found\n", sysname, command->name);
    return UNKNOWN;
}
```

```
-c  chatroom.c  COMP_304_Project_1.pdf  deneme.txt  new.txt  shellax-skeleton  shellax-skeleton.c  shellax-skeleton_new2  shellax-skeleton_old.c  test.c  tmp
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ cat deneme.txt
exructing: cat
Egg
Egg
Cinnamon
Flour
Flour
Milk
Milkavci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ echo "hello world" > hello.txt
exructing: echo
file could not be created or truncated!"hello world" hello.txt
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ echo "hello world" >hello.txt
exructing: echo
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ cat hello.txt
exructing: cat
"hello world"
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ █
```

Figure 4, I/O redirection result in shellax terminal



```
                waitpid(pid, NULL, 0);

                // waitpid(pid_child, NULL, 0);

                return SUCCESS;
            }
            else
            {
                printf("Background process %s is killed.", command->name);
                return SUCCESS;
            }
        }

        // TODO: your implementation here

        printf("-%s: %s: command not found\n", sysname, command->name);
        return UNKNOWN;
}
```

```
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main) [SIGINT]> gcc shellax-skeleton.c -o shellax-skeleton
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)> ./shellax-skeleton
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ cat deneme.txt | grep Egg
exructing: cat
exructing: grep
Egg
Egg
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$
```

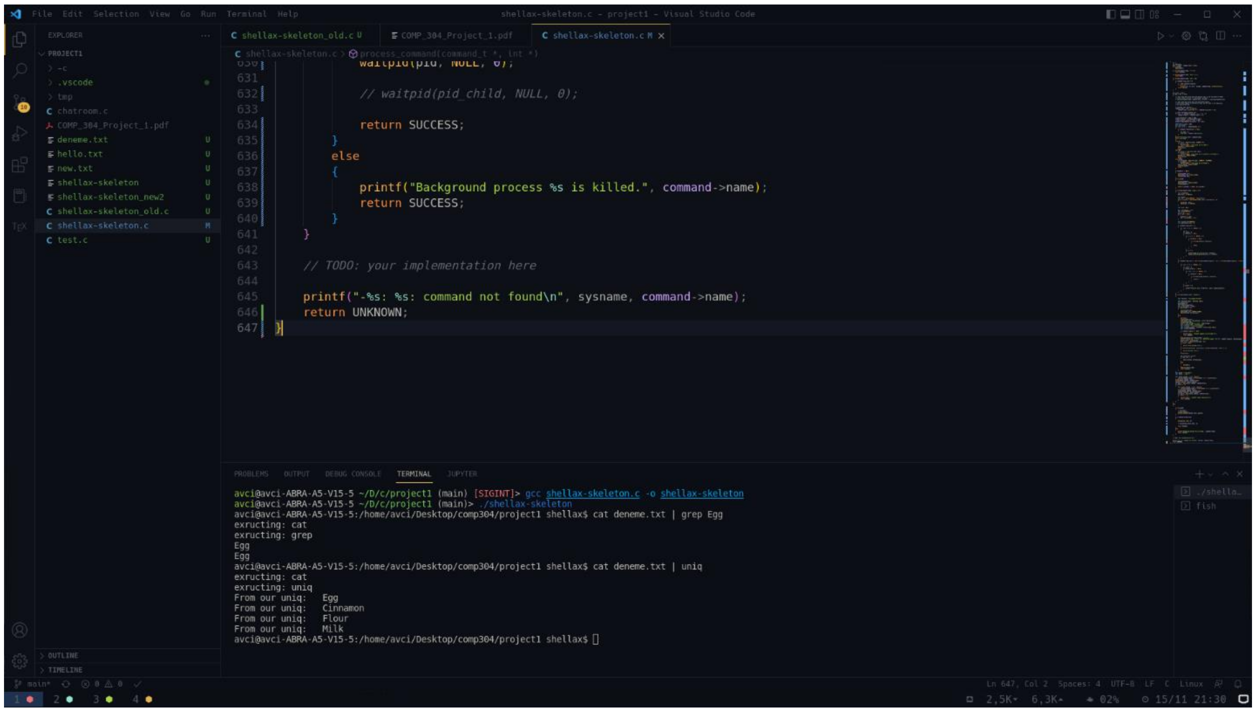
Figure 5, piping program

PART 3:

In the part 3, first uniq command is implemented. I implemented the uniq command in the shellax terminal, by checking the command name. I read from STDIN, using buffers. Then I parsed the str (which I created for reading from STDIN) according to newlines. Then I created char * words array and added all elements that are parsed according to the newline to this array. Then I created char * unique_words array and I added words of one occurrences to this array and count the number of elements in this array. Then according to the given arguments I checked and I printed the results of uniq or (uniq-c or uniq --count).

Then I implemented the chatroom part in the chatroom.c file. First I checked for argument counts. Then I created chatroom under the /tmp/ directory, using argv[1] and mkdir system call.[5] Then I created named pipe file using mkfifo and argv[1] and argv[2].[6] I used fork system call and create a child process for writing.[7] Using fgets, I took the message from STDIN, and then using struct dirent * dir, I opened the directory of chatroom, and I iterated the directory for writing the all named pipes in the directory[8]. Then using buffer, I opened the named pipes for writing and using their file descriptor in the write system call. [9] Then I closed the directory. In the parent process, I opened the piped name and using the file descriptor, I read the message.

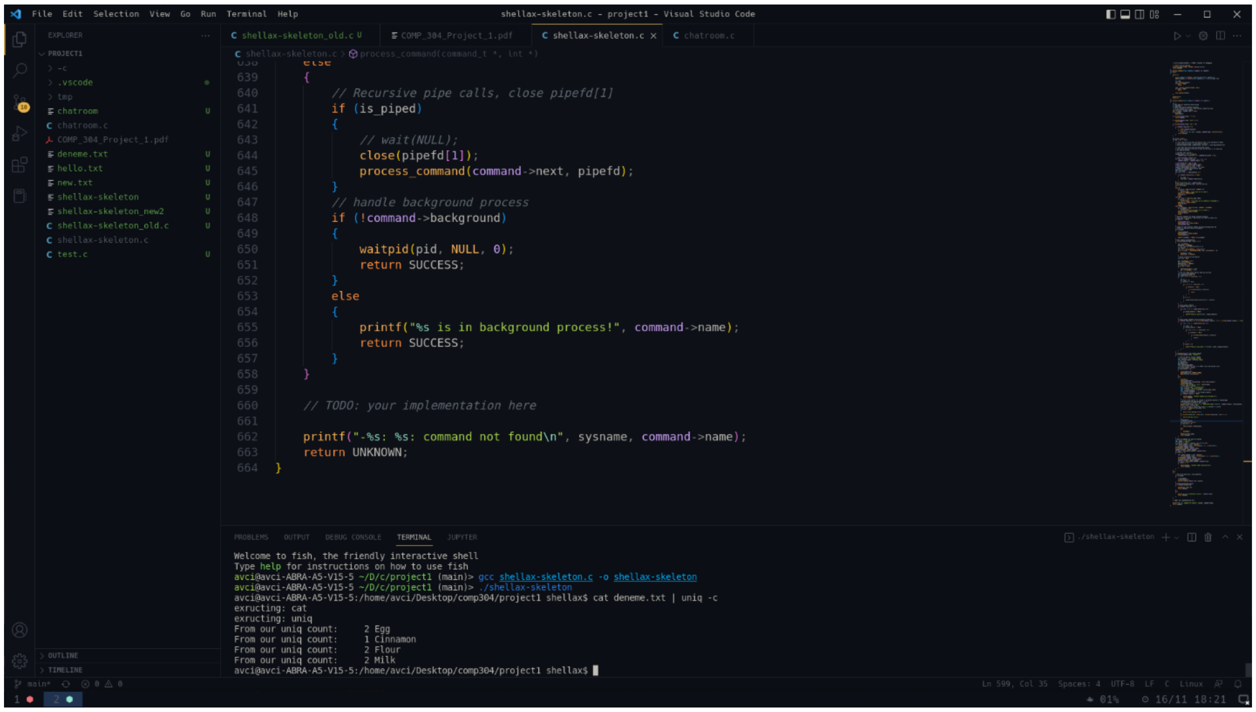
Then I implemented wiseman command in the shellax terminal. I checked the command name and used if statement for wiseman command execution. I created fortune_arg array which includes full path of fortune command. Then I created pipefd_e, then using this pipe and read_message char array, I read STDOUT of fortune command. During the reading, again I used dup2 system call for duplication of STDOUT file descriptor. I executed fortune command in the child process, then I read message in the parent process. Since crontab takes an argument as file, I created file in writing mode then using sprintf I created crontab argument and wrote to file using fwrite function.[10] Then again I used fork system call then executed the crontab and after the crontab function execution, I checked crontab jobs using crontab -l argument.[11]



```c
                waitpid(pid, NULL, 0);

                // waitpid(pid_child, NULL, 0);

                return SUCCESS;
            }
            else
            {
                printf("Background process %s is killed.", command->name);
                return SUCCESS;
            }
        }

    // TODO: your implementation here

    printf("-%s: %s: command not found\n", sysname, command->name);
    return UNKNOWN;
}
```

```
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main) [SIGINT]> gcc shellax-skeleton.c -o shellax-skeleton
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)> ./shellax-skeleton
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ cat deneme.txt | grep Egg
exructing: cat
exructing: grep
Egg
Egg
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ cat deneme.txt | uniq
exructing: cat
exructing: uniq
From our uniq:   Egg
From our uniq:   Cinnamon
From our uniq:   Flour
From our uniq:   Milk
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ 
```

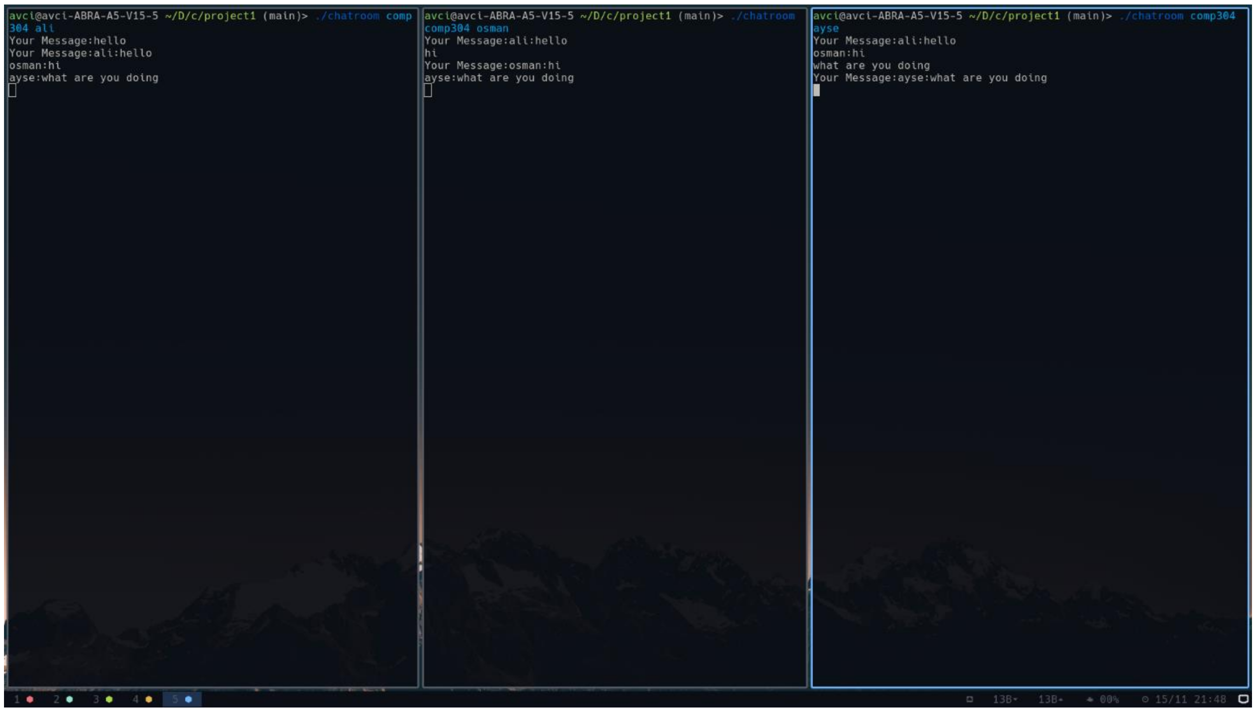
Figure 6, uniq command running



```c
        else
        {
            // Recursive pipe calls, close pipefd[1]
            if (is_piped)
            {
                // wait(NULL);
                close(pipefd[1]);
                process_command(command->next, pipefd);
            }
            // handle background process
            if (!command->background)
            {
                waitpid(pid, NULL, 0);
                return SUCCESS;
            }
            else
            {
                printf("%s is in background process!", command->name);
                return SUCCESS;
            }
        }

        // TODO: your implementation here

        printf("-%s: %s: command not found\n", sysname, command->name);
        return UNKNOWN;
    }
```

```
Welcome to fish, the friendly interactive shell
Type help for instructions on how to use fish
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)> gcc shellax-skeleton.c -o shellax-skeleton
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)> ./shellax-skeleton
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ cat deneme.txt | uniq -c
exructing: cat
exructing: uniq
From our uniq count:      2 Egg
From our uniq count:      1 Cinnamon
From our uniq count:      2 Flour
From our uniq count:      2 Milk
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$
```

Figure 7, uniq --count running

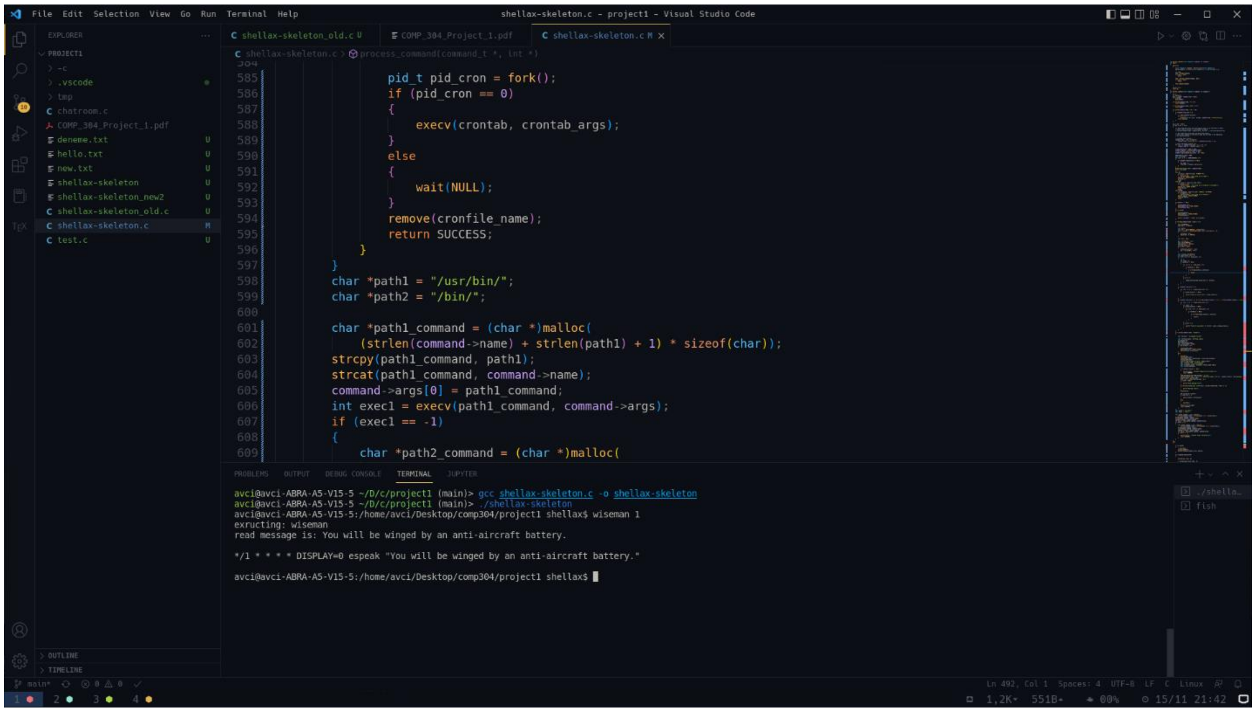


```
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)> ./chatroom comp
304 ali
Your Message:hello
Your Message:ali:hello
osman:hi
ayse:what are you doing
```

```
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)> ./chatroom
comp304 osman
Your Message:ali:hello
hi
Your Message:osman:hi
ayse:what are you doing
```

```
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)> ./chatroom comp304
ayse
Your Message:ali:hello
osman:hi
what are you doing
Your Message:ayse:what are you doing
```

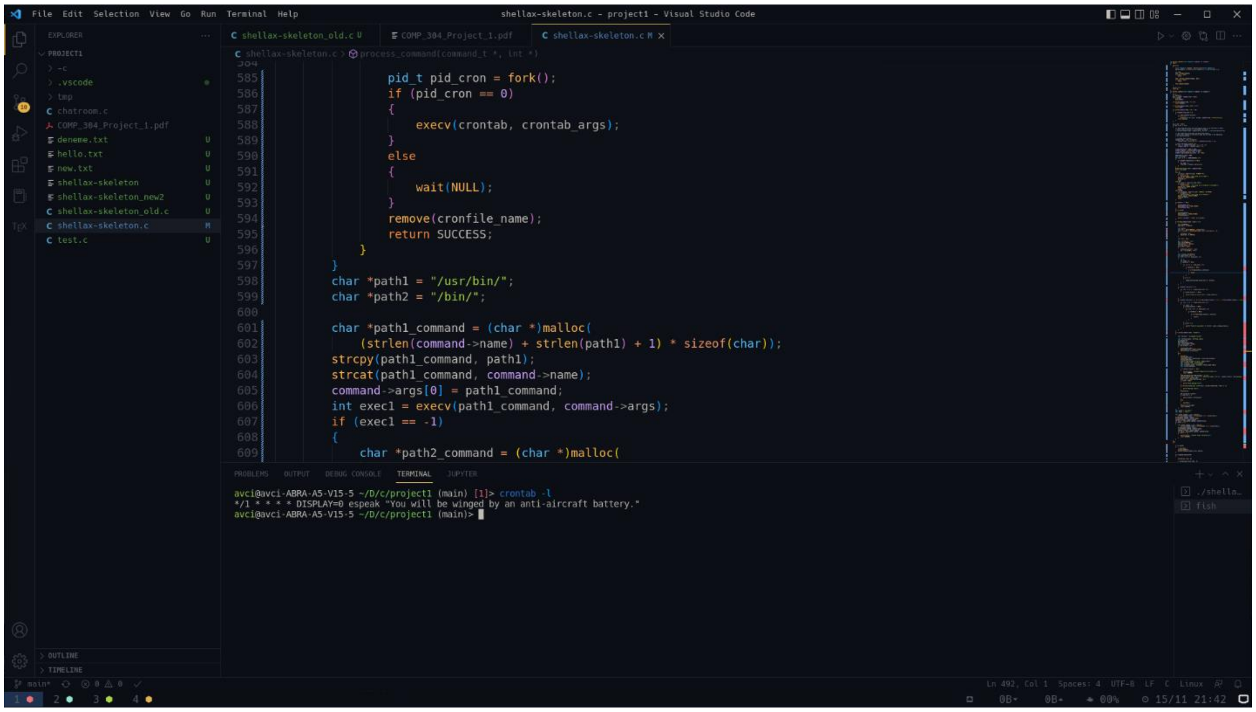
Figure 8, chatroom.c running



```c
        pid_t pid_cron = fork();
        if (pid_cron == 0)
        {
            execv(crontab, crontab_args);
        }
        else
        {
            wait(NULL);
        }
        remove(cronfile_name);
        return SUCCESS;
    }
}
char *path1 = "/usr/bin/";
char *path2 = "/bin/";

char *path1_command = (char *)malloc(
    (strlen(command->name) + strlen(path1) + 1) * sizeof(char));
strcpy(path1_command, path1);
strcat(path1_command, command->name);
command->args[0] = path1_command;
int exec1 = execv(path1_command, command->args);
if (exec1 == -1)
{
    char *path2_command = (char *)malloc(
```

```
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)> gcc shellax-skeleton.c -o shellax-skeleton
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)> ./shellax-skeleton
avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$ wiseman 1
exructing: wiseman
read message is: You will be winged by an anti-aircraft battery.

*/1 * * * * DISPLAY=0 espeak "You will be winged by an anti-aircraft battery."

avci@avci-ABRA-A5-V15-5:/home/avci/Desktop/comp304/project1 shellax$
```

Figure 9, Wiseman <time> running



```c
        pid_t pid_cron = fork();
        if (pid_cron == 0)
        {
            execv(crontab, crontab_args);
        }
        else
        {
            wait(NULL);
        }
        remove(cronfile_name);
        return SUCCESS;
    }
}
char *path1 = "/usr/bin/";
char *path2 = "/bin/";

char *path1_command = (char *)malloc(
    (strlen(command->name) + strlen(path1) + 1) * sizeof(char));
strcpy(path1_command, path1);
strcat(path1_command, command->name);
command->args[0] = path1_command;
int exec1 = execv(path1_command, command->args);
if (exec1 == -1)
{
    char *path2_command = (char *)malloc(
```

```
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main) [1]> crontab -l
*/1 * * * * DISPLAY=0 espeak "You will be winged by an anti-aircraft battery."
avci@avci-ABRA-A5-V15-5 ~/D/c/project1 (main)>
```

Figure 10, after crontab is executed

References

1. https://man7.org/linux/man-pages/man3/execv.3.html
2. https://man7.org/linux/man-pages/man2/dup2.2.html
3. https://man7.org/linux/man-pages/man2/creat.2.html
4. https://man7.org/linux/man-pages/man2/pipe.2.html
5. https://man7.org/linux/man-pages/man2/mkdir.2.html
6. https://man7.org/linux/man-pages/man3/mkfifo.3.html
7. https://man7.org/linux/man-pages/man2/fork.2.html
8. https://man7.org/linux/man-pages/man3/opendir.3.html
9. https://man7.org/linux/man-pages/man2/write.2.html
10. https://man7.org/linux/man-pages/man3/fgets.3.html
11. https://www.computerhope.com/unix/ucrontab.htm