COMP304 PROJECT - 1

Part 1:

First we fork the process to execute the given command.

For the children:

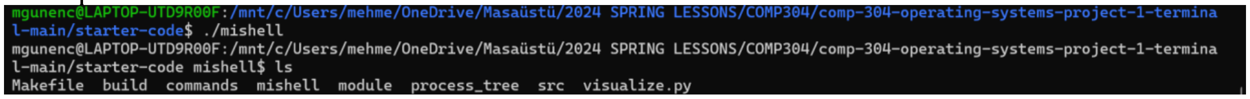
> We have used a method called resolve_path since not every command is under the same directory.
>
> Method ‘resolve_path’ takes a string and searches the path directory in the computer to see if there is an executable matching the name. If there is it returns the matching command if not it returns NULL.
>
> After receving the file path, we use execv with the filepath and the arguments for which extraction is handled for us with the given code.
>
> If there is an error with the execution perror catches and gives an error.

For the parent:

> It waits for the child to finish its execution.
>
> If ampersand operator is called it doesn’t wait for the child and immediately returns. Effectively showing the prompt.
>
> Therefore it returns SUCCESS as it should to show the prompt again.

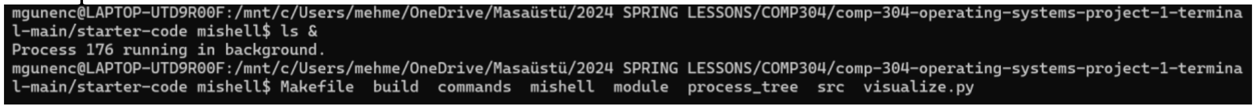
For the cd and exit commands only thing that we should have done is to correct the mistake in the if block that checks if the command name is ‘cd’ and writing return EXIT under the if block of the exit. Main method handles the exit by default.

Example of ls:



```
mgunenc@LAPTOP-UTD9R00F:/mnt/c/Users/mehme/OneDrive/Masaüstü/2024 SPRING LESSONS/COMP304/comp-304-operating-systems-project-1-termina
l-main/starter-code$ ./mishell
mgunenc@LAPTOP-UTD9R00F:/mnt/c/Users/mehme/OneDrive/Masaüstü/2024 SPRING LESSONS/COMP304/comp-304-operating-systems-project-1-termina
l-main/starter-code mishell$ ls
Makefile  build  commands  mishell  module  process_tree  src  visualize.py
```

Example of ls &:



```
mgunenc@LAPTOP-UTD9R00F:/mnt/c/Users/mehme/OneDrive/Masaüstü/2024 SPRING LESSONS/COMP304/comp-304-operating-systems-project-1-termina
l-main/starter-code mishell$ ls &
Process 176 running in background.
mgunenc@LAPTOP-UTD9R00F:/mnt/c/Users/mehme/OneDrive/Masaüstü/2024 SPRING LESSONS/COMP304/comp-304-operating-systems-project-1-termina
l-main/starter-code mishell$ Makefile  build  commands  mishell  module  process_tree  src  visualize.py
```

Part 2:

DISCLAIMER: There is a fault that occurs with the piping extraction code given to us.Tt sometimes does not separate from the ‘ | ’ as it should be and for example for ‘ ls | grep “a” ’ it assumes ‘ls | ‘ is the command to execute hence it doesn’t work expected. I want to state that we have sent multiple emails to all of the TA’s and the Hakan Ayral regarding this issue and never got a response and it is not our fault since this code was given to us to build upon our implementation.

Piping Implementation:

Under the process_command function before checking other functions we check if command->next != NULL to start the piping process since if there is a piped command it should be that the commands next is assigned to another command.

Then following the links, looping until the command->next is NULL the length of the pipe is deduced.

Then we create the pipes and the fd in order to handle the redirecting STDIN/STDOUT to an array so the next method can read from that array.

Then in a for loop that iterates for chain_length times, we fork the process and we set fd's that are piped to STDIN or STDOUT. Note that fd is an integer array so called file-descriptor and skipping indexes 2 by 2 we are piping the file descriptor, creating a number of pipes necessary for the chain_length number of commands.

After that to get the ith command we use c->next i times.

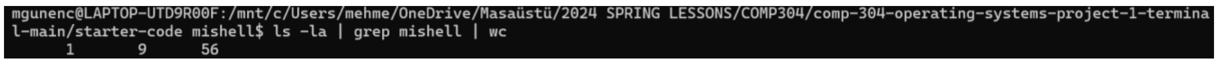
First with a separate method called process_built_ins we check if it is a built_in command,
If not we execute using execvp.

After this loop we loop again for 2*(chain_length-1) times and close the file descriptors.

Every parent that did not execute the aforementioned code, were waiting here for the children.

Then return SUCCESS to give control back to the user.

Example piping:



```
mgunenc@LAPTOP-UTD9R00F:/mnt/c/Users/mehme/OneDrive/Masaüstü/2024 SPRING LESSONS/COMP304/comp-304-operating-systems-project-1-termina
l-main/starter-code mishell$ ls -la | grep mishell | wc
      1       9      56
```

Part 3 section 1:

For this part we first used the key-handling of the given code, when the tab is pressed it is recorded as auto-completable.

Moreover we changed the main method a bit so when process_command return AUTO_COMPLETE it shows the prompt once again, with the changes applied by the process_command.

Under the process_command the first thing it checks is if command->auto-complete is true.

If it is our auto-completion code handles it,
Shortly line by line execution is as follows:

It creates a matches list to store the found matches. Then it copies the command name to a commd string and adds terminator to the end and calls the autocorrect with commd and matches as its arguments.

Method autocorrect():

It checks the path one by one to see if any command matches the given uncompleted command. Hence adds the found ones to the matches. It then adds NULL to the end of the list so we don't need to keep track of the size of the matches. It searches and compares the directories using the helper method search_directory.

Then we deduce the how many matches are there and store it under the n varible.

For the the single match(n=1):

We complete the method by printing the found command using the putchar method. Then we replace command->name with the found one and return AUTO_COMPLETE

For the multiple match:

We print every possible match and show the terminal prompt once again with the ex-input and return AUTO_COMPLETE

For the zero match(n=0):

We run the ls command and return SUCCESS to return the terminal to the user and list the directory as it is the behaviour requested by the instructions.

Part 3 section 2:

For the hdiff command we created a different hdiff.h file and imported it to our shell-skeleton.c.

Under the process_command, we check if command->name is "hdiff" and if so,
it passes the processing to the hdiff.h file.

Hdiff simply first checks if expected number of arguments are given.

Flag is optional and set to -a by default.

If there is no problem with the arg count, it extracts the absolute paths of the 2 files that are given.

Then it attempts opening them with read-only privileges.

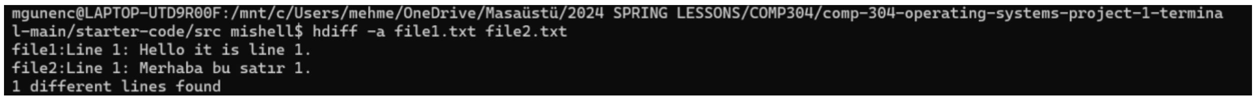
It checks if opening is successful and if not it prints an error.

It then, depending on the flag either uses compareLines or compareBytes methods.

compareLines(file1, file2):
    it uses fgets to read the files line by line in a while loop and compares two lines,
    If they differ it prints the 2 lines and increases the difference counter.

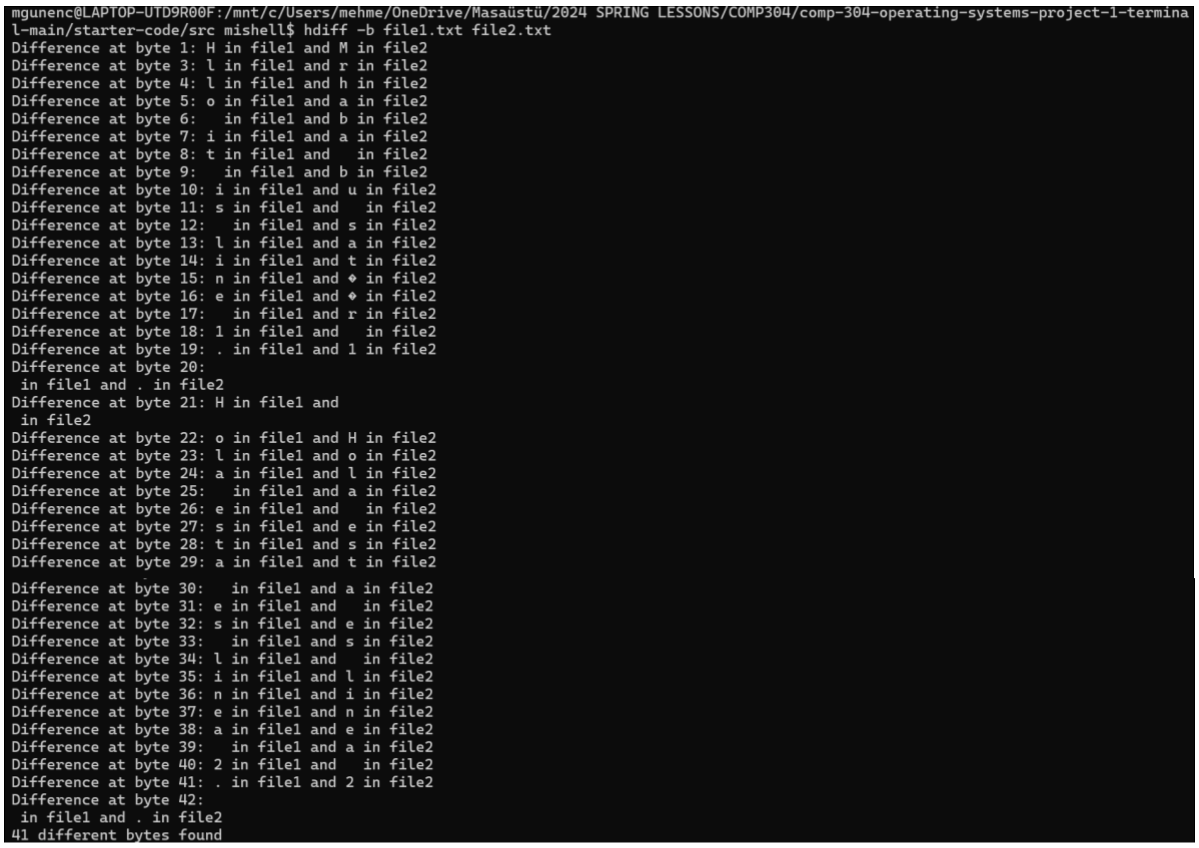
compareBytes(file1, file2):
    It uses getc method to get the char and compares this times char by char(aka byte by byte). Only significant difference between compareLines and compareBytes is one uses fgets while other uses getc.

Example hdiff with -a flag:



```
mgunenc@LAPTOP-UTD9R00F:/mnt/c/Users/mehme/OneDrive/Masaüstü/2024 SPRING LESSONS/COMP304/comp-304-operating-systems-project-1-termina
l-main/starter-code/src mishell$ hdiff -a file1.txt file2.txt
file1:Line 1: Hello it is line 1.
file2:Line 1: Merhaba bu satır 1.
1 different lines found
```

Example hdiff with -b flag:



```
mgunenc@LAPTOP-UTD9R00F:/mnt/c/Users/mehme/OneDrive/Masaüstü/2024 SPRING LESSONS/COMP304/comp-304-operating-systems-project-1-termina
l-main/starter-code/src mishell$ hdiff -b file1.txt file2.txt
Difference at byte 1: H in file1 and M in file2
Difference at byte 3: l in file1 and r in file2
Difference at byte 4: l in file1 and h in file2
Difference at byte 5: o in file1 and a in file2
Difference at byte 6:   in file1 and b in file2
Difference at byte 7: i in file1 and a in file2
Difference at byte 8: t in file1 and   in file2
Difference at byte 9:   in file1 and b in file2
Difference at byte 10: i in file1 and u in file2
Difference at byte 11: s in file1 and   in file2
Difference at byte 12:   in file1 and s in file2
Difference at byte 13: l in file1 and a in file2
Difference at byte 14: i in file1 and t in file2
Difference at byte 15: n in file1 and � in file2
Difference at byte 16: e in file1 and � in file2
Difference at byte 17:   in file1 and r in file2
Difference at byte 18: 1 in file1 and   in file2
Difference at byte 19: . in file1 and 1 in file2
Difference at byte 20:
 in file1 and . in file2
Difference at byte 21: H in file1 and
 in file2
Difference at byte 22: o in file1 and H in file2
Difference at byte 23: l in file1 and o in file2
Difference at byte 24: a in file1 and l in file2
Difference at byte 25:   in file1 and a in file2
Difference at byte 26: e in file1 and   in file2
Difference at byte 27: s in file1 and e in file2
Difference at byte 28: t in file1 and s in file2
Difference at byte 29: a in file1 and t in file2
Difference at byte 30:   in file1 and a in file2
Difference at byte 31: e in file1 and   in file2
Difference at byte 32: s in file1 and e in file2
Difference at byte 33:   in file1 and s in file2
Difference at byte 34: l in file1 and   in file2
Difference at byte 35: i in file1 and l in file2
Difference at byte 36: n in file1 and i in file2
Difference at byte 37: e in file1 and n in file2
Difference at byte 38: a in file1 and e in file2
Difference at byte 39:   in file1 and a in file2
Difference at byte 40: 2 in file1 and   in file2
Difference at byte 41: . in file1 and 2 in file2
Difference at byte 42:
 in file1 and . in file2
41 different bytes found
```

Part 3 section 3:

Custom Command of Tuna CIMEN - BACKUP & REVERT:

This command includes 2 commands:

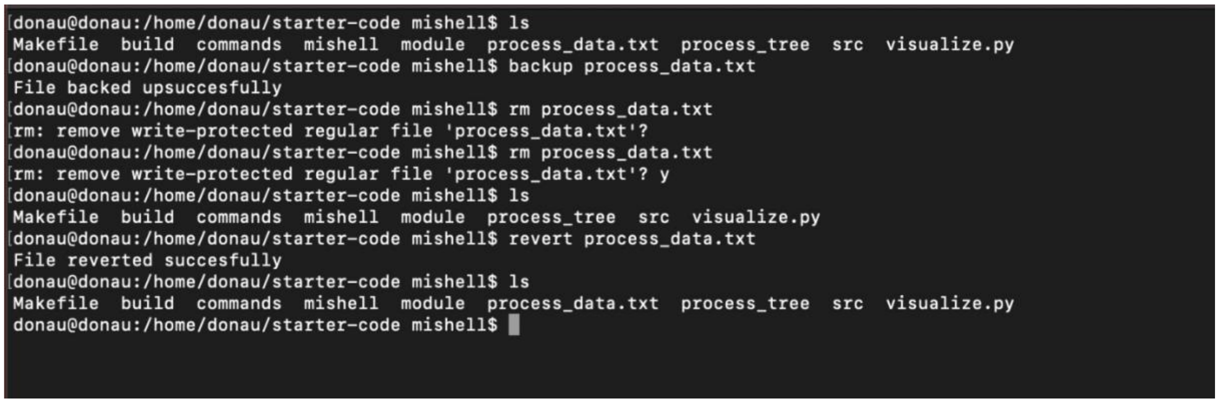
backup <filename>: it saves the current file to the memory so even though it is deleted or changed, it can be reverted.

revert <filename>: it checks if there is a save with the given name, and if there is, it reverts/ brings back the file to the current working directory.

How They Work?

Both commands work similarly, they get the current working directory and they copy the current file to the top directory received by getenv("HOME") command or copy from the getenv("Home") to the current working directory.

An example of backup & revert:



```
[donau@donau:/home/donau/starter-code mishell$ ls
Makefile  build  commands  mishell  module  process_data.txt  process_tree  src  visualize.py
[donau@donau:/home/donau/starter-code mishell$ backup process_data.txt
File backed upsuccesfully
[donau@donau:/home/donau/starter-code mishell$ rm process_data.txt
[rm: remove write-protected regular file 'process_data.txt'?
[donau@donau:/home/donau/starter-code mishell$ rm process_data.txt
[rm: remove write-protected regular file 'process_data.txt'? y
[donau@donau:/home/donau/starter-code mishell$ ls
Makefile  build  commands  mishell  module  process_tree  src  visualize.py
[donau@donau:/home/donau/starter-code mishell$ revert process_data.txt
File reverted succesfully
[donau@donau:/home/donau/starter-code mishell$ ls
Makefile  build  commands  mishell  module  process_data.txt  process_tree  src  visualize.py
donau@donau:/home/donau/starter-code mishell$
```

Firstly, process_data.txt is backed up.
After that, it was removed with rm (can be seen after that with the ls command) .
Then, using revert, process_data.txt was successfully reverted (can be seen after that with the ls command).

Part 3 section 3:
Custom Command of Mehmet Tuna GUNENC – SIZEDIR:

My custom command "sizedir" provides a view of the size of directory contents along with the total size of a specified directory, allowing users to specify the unit of measurement through various flags. "sizedir" command supports the following flags:

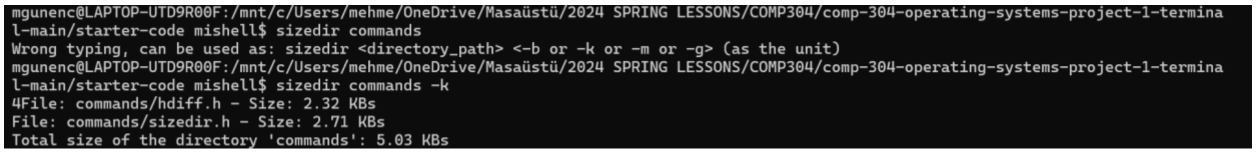
-b: Displays the size in bytes.
-k: Displays the size in kilobytes.

-m: Displays the size in megabytes.
-g: Displays the size in gigabytes.

Each flag allows the user to choose a different unit for displaying file sizes which offers flexibility based on the user's needs or the situation.

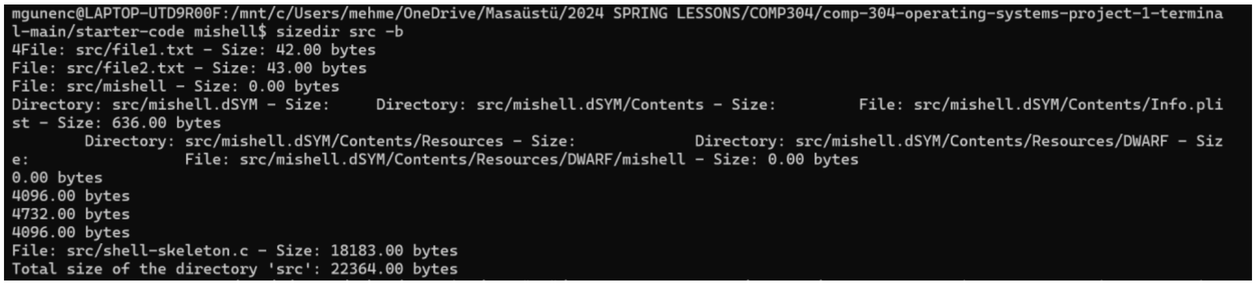
“sizedir” commands purpose is to enable users to efficiently determine the sizes of all files within a directory by the desired unit (flag put), as well as the total size of the directory itself.

An example for sizedir using -k flag:



```
mgunenc@LAPTOP-UTD9R00F:/mnt/c/Users/mehme/OneDrive/Masaüstü/2024 SPRING LESSONS/COMP304/comp-304-operating-systems-project-1-termina
l-main/starter-code mishell$ sizedir commands
Wrong typing, can be used as: sizedir <directory_path> <-b or -k or -m or -g> (as the unit)
mgunenc@LAPTOP-UTD9R00F:/mnt/c/Users/mehme/OneDrive/Masaüstü/2024 SPRING LESSONS/COMP304/comp-304-operating-systems-project-1-termina
l-main/starter-code mishell$ sizedir commands -k
4File: commands/hdiff.h - Size: 2.32 KBs
File: commands/sizedir.h - Size: 2.71 KBs
Total size of the directory 'commands': 5.03 KBs
```

An example for sizedir using -b flag:



```
mgunenc@LAPTOP-UTD9R00F:/mnt/c/Users/mehme/OneDrive/Masaüstü/2024 SPRING LESSONS/COMP304/comp-304-operating-systems-project-1-termina
l-main/starter-code mishell$ sizedir src -b
4File: src/file1.txt - Size: 42.00 bytes
File: src/file2.txt - Size: 43.00 bytes
File: src/mishell - Size: 0.00 bytes
Directory: src/mishell.dSYM - Size:     Directory: src/mishell.dSYM/Contents - Size:         File: src/mishell.dSYM/Contents/Info.pli
st - Size: 636.00 bytes
        Directory: src/mishell.dSYM/Contents/Resources - Size:             Directory: src/mishell.dSYM/Contents/Resources/DWARF - Siz
e:                File: src/mishell.dSYM/Contents/Resources/DWARF/mishell - Size: 0.00 bytes
0.00 bytes
4096.00 bytes
4732.00 bytes
4096.00 bytes
File: src/shell-skeleton.c - Size: 18183.00 bytes
Total size of the directory 'src': 22364.00 bytes
```

Part 4:

We have implemented psvis command in a different file and made it recognizable by the mishell.

Psvis command loads the kernel module named ‘mymodule’ to the kernel, it request a sudo password.

Mymodule:

We have used the template from the problem session 4 to set a character device, so we can read from the user space.

Upon being uploaded to the kernel, it creates a character device and gets the pid that is given to the module, if no pid given, it assumes it 1.

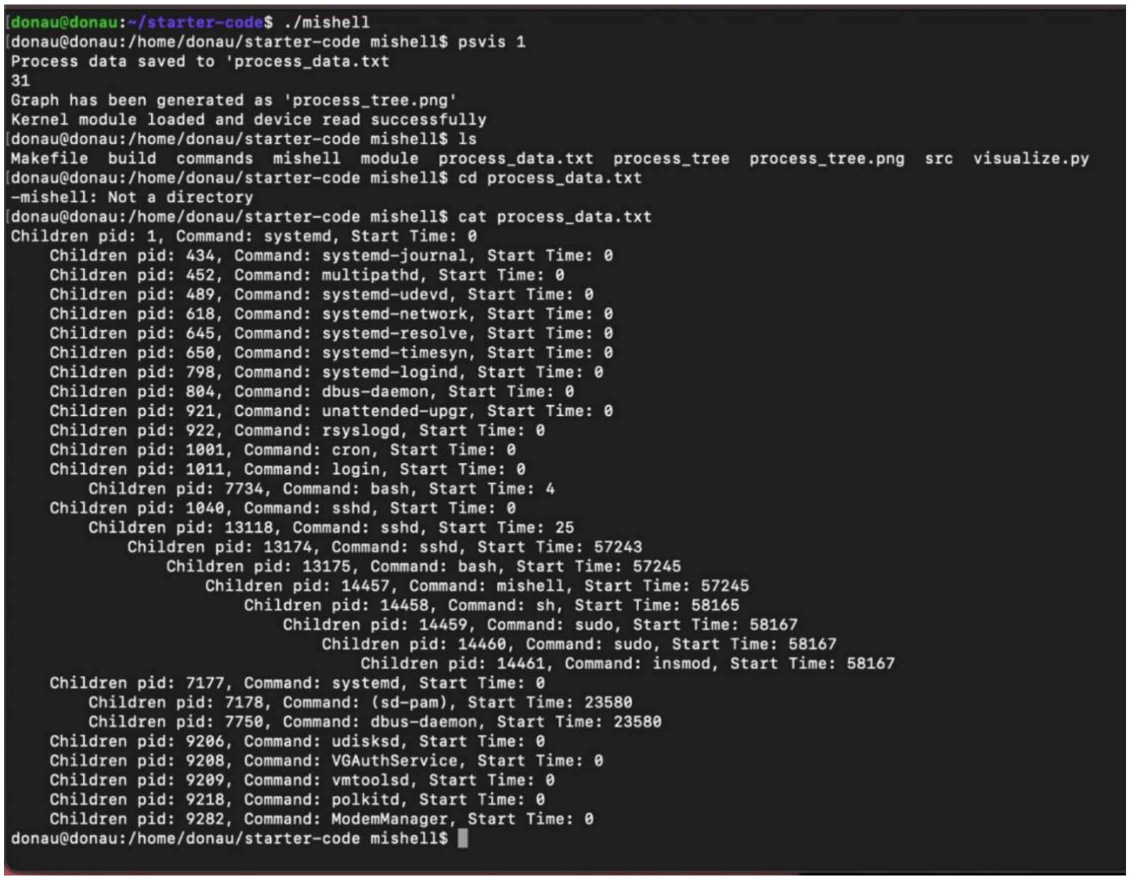
Then by using print_process_tree command, it fills the process_tree_buffer by adding a formatted string which includes a name, pid and the start_time , each line consists of a tab which indicates the child hierarchy and recursively does the same for the children nodes.

Under the device_read this buffer is copied to the buffer and the size to read is returned accordingly.

Later the psvis opens and reads the character device created by the mymodule and dumps this buffer to a file named “process_data.txt”.

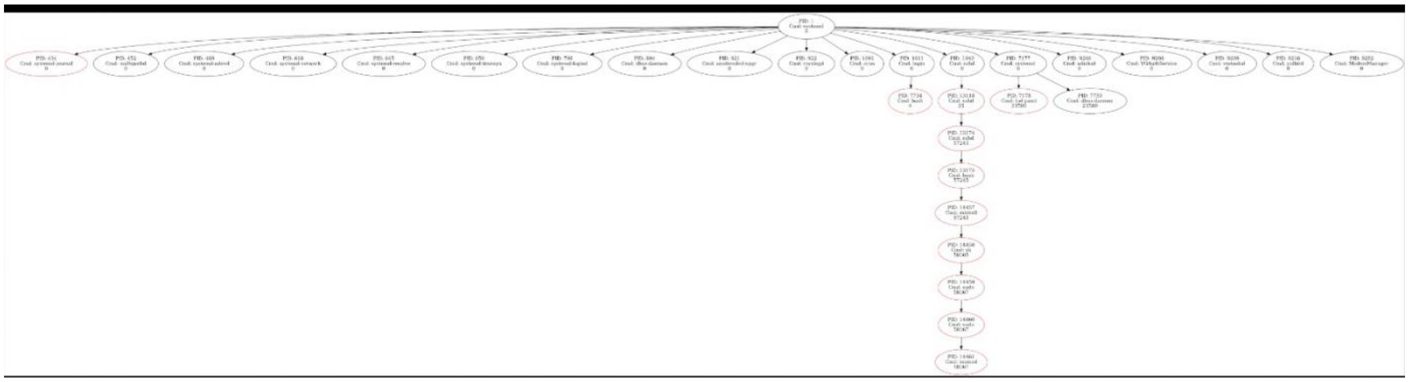
Then we use a python script to read from the “process_data.txt” and create the process tree image. Since there is a formatting including with the tabs it increases the depth any time there is a new tab, and keeps record of the last parent using the depth_map.
It then colors the heir by looping in the children and find the eldest process.

Psvis example:



```
[donau@donau:~/starter-code$ ./mishell
[donau@donau:/home/donau/starter-code mishell$ psvis 1
Process data saved to 'process_data.txt
31
Graph has been generated as 'process_tree.png'
Kernel module loaded and device read successfully
[donau@donau:/home/donau/starter-code mishell$ ls
Makefile  build  commands  mishell  module  process_data.txt  process_tree  process_tree.png  src  visualize.py
[donau@donau:/home/donau/starter-code mishell$ cd process_data.txt
-mishell: Not a directory
[donau@donau:/home/donau/starter-code mishell$ cat process_data.txt
Children pid: 1, Command: systemd, Start Time: 0
    Children pid: 434, Command: systemd-journal, Start Time: 0
    Children pid: 452, Command: multipathd, Start Time: 0
    Children pid: 489, Command: systemd-udevd, Start Time: 0
    Children pid: 618, Command: systemd-network, Start Time: 0
    Children pid: 645, Command: systemd-resolve, Start Time: 0
    Children pid: 650, Command: systemd-timesyn, Start Time: 0
    Children pid: 798, Command: systemd-logind, Start Time: 0
    Children pid: 804, Command: dbus-daemon, Start Time: 0
    Children pid: 921, Command: unattended-upgr, Start Time: 0
    Children pid: 922, Command: rsyslogd, Start Time: 0
    Children pid: 1001, Command: cron, Start Time: 0
    Children pid: 1011, Command: login, Start Time: 0
        Children pid: 7734, Command: bash, Start Time: 4
    Children pid: 1040, Command: sshd, Start Time: 0
        Children pid: 13118, Command: sshd, Start Time: 25
            Children pid: 13174, Command: sshd, Start Time: 57243
                Children pid: 13175, Command: bash, Start Time: 57245
                    Children pid: 14457, Command: mishell, Start Time: 57245
                        Children pid: 14458, Command: sh, Start Time: 58165
                            Children pid: 14459, Command: sudo, Start Time: 58167
                                Children pid: 14460, Command: sudo, Start Time: 58167
                                    Children pid: 14461, Command: insmod, Start Time: 58167
    Children pid: 7177, Command: systemd, Start Time: 0
        Children pid: 7178, Command: (sd-pam), Start Time: 23580
        Children pid: 7750, Command: dbus-daemon, Start Time: 23580
    Children pid: 9206, Command: udisksd, Start Time: 0
    Children pid: 9208, Command: VGAuthService, Start Time: 0
    Children pid: 9209, Command: vmtoolsd, Start Time: 0
    Children pid: 9218, Command: polkitd, Start Time: 0
    Children pid: 9282, Command: ModemManager, Start Time: 0
donau@donau:/home/donau/starter-code mishell$ 
```

Process tree visualization:


PID: 1 Cmd: systemd 0
PID: 1001 Cmd: cron 0
PID: 1011 Cmd: login 0
PID: 7734 Cmd: bash 0
PID: 7177 Cmd: systemd 0
PID: 8086 Cmd: VGAuthService 0
PID: 9252 Cmd: ModemManager 0