README file

Members in the group

- Jasmina Destanović
- Iman Ahmagić

List of all files/directories in our submission

- df.c
- finaltask.c
- foo.txt
- forkalone.c
- grep.c
- matrix.c
- piping.c
- shellproject.c
- syscalls.c
- test.txt
- wc.c

Theoretical questions

- **If we have a single-core, uniprocessor system that supports multiprogramming, how many processes can be in a running state in such a system, at any given time?**

In a single-core, uniprocessor system that supports multiprogramming there is only one process that can execute instructions on the CPU. The reason for that is because of the CPU's ability to handle instructions for only one process at a time..

Even though there can be only one process actively running, it still allows for multiple processes to be loaded into memory at the same time, they just can't be actively running simultaneously.

So the CPU rapidly switches between different processes, giving each process a turn to execute for a short period before being swapped out for another process. (aka context switching)

For instance, when a user is simultaneously using multiple activities on their pc, web browsing (like Google Chrome), music playback (like Spotify) and email communication (via Microsoft Outlook). The CPU can only execute instructions for one process at the time, the operating system rapidly switches between these processes, so each application gets its time to run on the CPU.

- **Explain why system calls are needed for a shared memory method of inter-process communication (IPC). If there are multiple threads in one process, are the system calls needed for sharing memory between those threads?**

System calls are needed for a shared memory of inter-process communication because they provide the necessary mechanisms for processes to request access to shared memory regions, manage synchronization and enforce memory protection between processes.

For instance, in a program utilizing **fork()** system call to create new processes. Upon calling **fork()** doesn't automatically share the memory between the parent and child processes. Instead, the memory is copied/duplicated so each process has its own separate memory space.

Any changes made to that memory is only seen by the process which made the change. So to achieve shared memory, system calls such as **shmget(), shmat(), shmdt()** and **shmctl()** are needed to make shared memory.

shmget() - creates shared memory segments
shmat() - attaches a shared memory segment
shmdt() - detaches a shared memory segment
shmctl() - controls shared memory segments

In contrast, within a single process with multiple threads, system calls may not be necessary. Threads within the same process share the same memory space by inheriting it, and they can access shared memory directly without the need for system calls.

Explanation and outline of assignments

1.Shell interface

Basic prompt

```
jasmina@Ubuntu2:~$ nano shellproject.c
jasmina@Ubuntu2:~$ gcc -o shellproject shellproject.c
jasmina@Ubuntu2:~$ ./shellproject
prompt$
```

```
  GNU nano 6.2
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

#define MAX_LINE 100

int main(){
        char command[MAX_LINE];
        int should_run=1;


        while(should_run){
                printf("prompt$ ");
                fflush(stdout);

                fgets(command, MAX_LINE, stdin);

                if(command[strlen(command) -1] == '\n')
                command[strlen(command) -1] = '\n';


                if(strcmp(command, "exit") == 0)
                        break;



                pid_t pid = fork();

                if(pid < 0){
                        perror("fork");
                        exit(EXIT_FAILURE);
                }else if(pid == 0){
                        if(execlp(command, command, NULL) == -1){
                                perror("exec");
                                exit(EXIT_FAILURE);
                        }
                } else{
                        wait(NULL);
                }
        }
        return 0;
}
```

Prompt with the username

```
jasmina@Ubuntu2:~$ nano shellproject.c
jasmina@Ubuntu2:~$ gcc -o shellproject shellproject.c
jasmina@Ubuntu2:~$ ./shellproject
Program started
jasmina@machine:~$
```

Shell interface program

This program has a basic shell interface that repeatedly prompts user for input, reads the input and executes the command

Prompt with username code

```
  GNU nano 6.2
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

#define MAX_LINE 100
#define PROMPT_SIZE 100

int main(){
        printf("Program started\n");
        char command[MAX_LINE];
        char prompt[PROMPT_SIZE];
        char *username;
        int should_run=1;


        username = getenv("USER");

        while(should_run){
                snprintf(prompt, PROMPT_SIZE, "%s@machine:~$ ", username);
                printf("%s", prompt);
                fflush(stdout);

                fgets(command, MAX_LINE, stdin);

                if(command[strlen(command) -1] == '\n')
                command[strlen(command) -1] = '\n';


                if(strcmp(command, "exit") == 0)
                        break;



                pid_t pid = fork();

                if(pid < 0){
                        perror("fork");
                        exit(EXIT_FAILURE);
                }else if(pid == 0){
                        if(execlp(command, command, NULL) == -1){
                                perror("exec");
                                exit(EXIT_FAILURE);
                        }
                } else{
```

2. Shell programs/commands

wc command

- wc - word count
- this program will count the number of lines, words and bytes in a specific file
- the main function will check if there is at least 1 file name provided, and if not then it will return a message and exit
- the program iterates through each filename that was provided as command-line argument, then call the wc function for each file

```
jasmina@Ubuntu2:~$ nano wc.c
jasmina@Ubuntu2:~$ gcc -o wc wc.c
jasmina@Ubuntu2:~$ nano test.txt
jasmina@Ubuntu2:~$ ./wc test.txt
The file has 5 words, 1 lines, 25 bytes:  test.txt
jasmina@Ubuntu2:~$
```

```
jasmina@Ubuntu2: ~
  GNU nano 6.2                          test.txt *
this file has five words
```

jasmina@Ub

GNU nano 6.2 wc

```
void wc(const char *filename){
        FILE *file = fopen(filename, "r");
        if(file==NULL){
                fprintf(stderr, "File can't be opened '%s'\n", filename);
                return;
        }

        int lines = 0, words = 0, bytes = 0;
        char c;
        int word=0;


        while((c = fgetc(file)) != EOF){
                bytes++;

                if(c=='\n'){
                        lines++;
                }

                if(!isspace(c)){
                        if(!word){
                                word=1;
                                words++;
                        }
                }else{
                        word=0;
                }
        }

        fclose(file);
        printf("The file has %d words, %d lines, %d bytes:  %s\n", words, lines,  bytes, filename);
}


int main(int argc, char *argv[]){
        if(argc<2){
                fprintf(stderr,"The file %s <filename>\n", argv[0]);
                return 1;
        }

        for(int i=1; i<argc; i++){
                wc(argv[i]);
        }

        return 0;
```

grep command

- grep - global regular expression print, searches for text that matches the regular expression pattern
- this program will read each line from the specified file and check if the pattern exist
- the main function checks if the pattern and filename is provided, if not exists, if yes it calls the grep function

```
GNU nano 6.2
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void grep(const char *pattern, const char *filename){
        FILE *file = fopen(filename, "r");
        if(file == NULL){
                fprintf(stderr, "File not found '%s'\n", filename);
                return;
        }

        char line[256];
        while(fgets(line, sizeof(line), file)){
                if(strstr(line, pattern)){
                        printf("%s", line);
                }
        }

        fclose(file);
}

int main(int argc, char *argv[]){
        if(argc < 3){
                fprintf(stderr, "Usage is %s <pattern> <filename>\n", argv[0]);
                return 1;
        }

        grep(argv[1], argv[2]);

        return 0;
}
```

```
jasmina@Ubuntu2:~$ nano grep.c
jasmina@Ubuntu2:~$ gcc -o grep grep.c
jasmina@Ubuntu2:~$ nano test.txt
jasmina@Ubuntu2:~$ ./grep file test.txt
this file has five words
this sentence has a file in it
file
some file
jasmina@Ubuntu2:~$
```

df command

- df - disk free
- this program will display disk space usage information
- the code retrieves file system statistics using statvfs function and prints the total and free space in bytes for the file system

```
  GNU nano 6.2
#include <stdio.h>
#include <stdlib.h>
#include <sys/statvfs.h>

void df(const char *path){
        struct statvfs stat;
        if(statvfs(path, &stat) != 0){
                perror("No information");
                return;
        }

        unsigned long long free_blocks = stat.f_frsize;
        unsigned long long block_size = stat.f_frsize;
        unsigned long long total_blocks = stat.f_blocks;

        unsigned long long total = total_blocks * block_size;
        unsigned long long free = free_blocks * block_size;

        printf("Total space: %llu bytes\n", total);
        printf("Free space: %llu bytes\n", free);

}


int main(int argc, char *argv[]){
        if(argc<2){
                fprintf(stderr, "Usage: %s <path>\n", argv[0]);
                return 1;
        }

        df(argv[1]);
        return 0;
}
```

```
jasmina@Ubuntu2:~$ nano df.c
jasmina@Ubuntu2:~$ gcc -o df df.c
jasmina@Ubuntu2:~$ ./df /
Total space: 25709252608 bytes
Free space: 16777216 bytes
jasmina@Ubuntu2:~$
```

cmatrix

- this program will generate a matrix-like animation by continuously updating a 2D array of characters
- each character is chosen from the ASCII range

```
GNU nano 6.2
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

#define WIDTH 100
#define HEIGHT 50
#define DELAY 100000

void scr(){
        printf("\033[2J\033[1;1H");
}


void matrix_drawing(){

        if(!isatty(fileno(stdout))){
                FILE *output = stdout;
                if(output==NULL){
                        fprintf(stderr, "Not working\n");
                        exit(1);
                }
                stdout = output;
        }


        char matrix[HEIGHT][WIDTH];
        int i, j;


        while(1){
                scr();

                for(i=0; i<HEIGHT; i++){
                        for(j=0; j<WIDTH; j++){
                                matrix[i][j] = rand() % 94 + 33;
                        }
                }

                for(i=0; i<HEIGHT; i++){
                        for(j=0; j<WIDTH; j++){
                                printf("%c", matrix[i][j]);
                        }
                        printf("\n");
                }
```

```c
        char matrix[HEIGHT][WIDTH];
        int i, j;


        while(1){
                scr();

                for(i=0; i<HEIGHT; i++){
                        for(j=0; j<WIDTH; j++){
                                matrix[i][j] = rand() % 94 + 33;
                        }
                }

                for(i=0; i<HEIGHT; i++){
                        for(j=0; j<WIDTH; j++){
                                printf("%c", matrix[i][j]);
                        }
                        printf("\n");
                }


                fflush(stdout);
                usleep(DELAY);
        }
}


int main(){
        srand(time(NULL));
        scr();
        matrix_drawing();

        return 0;
}
```

Ubuntu Software

```
jasmina@Ubuntu2:~$ nano matrix.c
jasmina@Ubuntu2:~$ gcc -o matrix matrix.c
jasmina@Ubuntu2:~$ ./matrix
```

Piping

- piping - allows output of one command to be used as an input for another command (chain of commands)
- print function checks if stdout is redirected or piped using isatty()
- if yes the function opens the specified file for writing, then it writes it to the output stdout or the file

```
GNU nano 6.2
#include <stdio.h>
#include <unistd.h>


void print(const char *output){
        if(!isatty(fileno(stdout))){
                FILE *outputfile = stdout;

                if(outputfile == NULL){
                        fprintf(stderr, "Error\n");
                        return;
                }
                stdout = outputfile;
        }

        fprintf(stdout, "%s", output);
}

int main(){
        const char *output = "shell project\n";
        print(output);
        return 0;
}
```

```
jasmina@Ubuntu2:~$ nano piping.c
jasmina@Ubuntu2:~$ gcc -o piping piping.c
jasmina@Ubuntu2:~$ ./piping > test.txt
jasmina@Ubuntu2:~$ ./piping | less
jasmina@Ubuntu2:~$ cat test.txt
shell project
jasmina@Ubuntu2:~$
```

3. System Calls

fork() - the way of operating systems create another process where process clones itself (new processes are child processes, original process is the parent process)

```
jasmina@Ubuntu2: ~
GNU nano 6.2                    forkalone.c
#include <stdio.h>
#include <unistd.h>

int main(){
        while(1){
                int pid=fork();
                if(pid==0){
                        printf("Child process pid: %d\n", getpid());
                }else if(pid<0){
                        perror("fork");
                        break;
                }

        }
        return 0;
}
```

```
jasmina@Ubuntu2:~$ nano forkalone.c
jasmina@Ubuntu2:~$ gcc -o forkalone forkalone.c
jasmina@Ubuntu2:~$ ./forkalone
Child process pid: 194817
Child process pid: 194816
Child process pid: 194818
Child process pid: 194815
Child process pid: 194820
Child process pid: 194819
Child process pid: 194821
Child process pid: 194822
Child process pid: 194823
Child process pid: 194824
Child process pid: 194826
Child process pid: 194829
Child process pid: 194825
Child process pid: 194830
Child process pid: 194834
Child process pid: 194849
Child process pid: 194852
Child process pid: 194853
Child process pid: 194854
Child process pid: 194831
Child process pid: 194851
Child process pid: 194836
Child process pid: 194850
Child process pid: 194835
Child process pid: 194827
Child process pid: 194855
Child process pid: 194837
Child process pid: 194904
Child process pid: 194828
Child process pid: 194832
Child process pid: 194868
Child process pid: 194869
Child process pid: 194870
Child process pid: 194878
Child process pid: 194856
Child process pid: 194885
```

- an example of a program where parent process forks a child process
- fork system call creates a new process, where in the parent process fork() returns the child's process ID (pid)
- child process executes a new program using exec()
- exec() replaces the child process with a new program
- parent process waits for the child to finish using wait()

```
GNU nano 6.2
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main(){
        pid_t pid = fork();

        if(pid < 0){
                fprintf(stderr, "Failed\n");
                exit(EXIT_FAILURE);
        }

        else if(pid==0){
                printf("Child process\n");
                execl("/bin/ls", "ls", "-l", NULL);

                perror("execl");
                exit(EXIT_FAILURE);
        }

        else{
                printf("Parent process\n");
                int status;
                wait(&status);

                if(WIFEXITED(status)){
                        printf("Child process %d\n", WEXITSTATUS(status));
                }else{
                        printf("Stopped");
                }
        }
        return 0;
}
```

```
jasmina@Ubuntu2:~$ nano syscalls.c
jasmina@Ubuntu2:~$ gcc -o syscalls syscalls.c
jasmina@Ubuntu2:~$ ./syscalls
Parent process
Child process
total 272
-rw-rw-r-- 1 jasmina jasmina 10650 mar 13 12:37 assignment.txt
-rwxrwxr-x 1 jasmina jasmina 16008 mar 13 12:18 calculator
-rw-rw-r-- 1 jasmina jasmina   309 mar 13 12:18 calculator.c
-rw-rw-r-- 1 jasmina jasmina  1157 mar 13 12:40 calculator.zip
-rw-rw-r-- 1 jasmina jasmina    14 mar 21 17:13 cat
-rw-rw-r-- 1 jasmina jasmina  1157 mar 13 12:40 cfiles.zip
drwxr-xr-x 3 jasmina jasmina  4096 mar 16 07:55 Desktop
-rwxrwxr-x 1 jasmina jasmina 16192 mar 21 16:38 df
-rw-rw-r-- 1 jasmina jasmina   678 mar 21 16:37 df.c
drwxr-xr-x 2 jasmina jasmina  4096 mar 13 12:09 Documents
drwxr-xr-x 2 jasmina jasmina  4096 mar 16 07:55 Downloads
-rw-rw-r-- 1 jasmina jasmina    24 mar 21 13:57 file.txt
-rw-rw-r-- 1 jasmina jasmina  1657 mar 13 12:37 final.txt
-rwxrwxr-x 1 jasmina jasmina 16048 mar 13 12:30 findSmallest
-rw-rw-r-- 1 jasmina jasmina   544 mar 13 12:28 findSmallest.c
-rw-rw-r-- 1 jasmina jasmina   545 mar 13 12:29 findvalue.c
-rwxrwxr-x 1 jasmina jasmina 16280 mar 21 16:25 grep
-rw-rw-r-- 1 jasmina jasmina   543 mar 21 16:20 grep.c
-rw-rw-r-- 1 jasmina jasmina    55 mar 14 13:00 jasmina.c
-rwxrwxr-x 1 jasmina jasmina 16520 mar 21 17:02 matrix
-rw-rw-r-- 1 jasmina jasmina   750 mar 21 17:01 matrix.c
drwxr-xr-x 2 jasmina jasmina  4096 mar 13 12:09 Music
-rw-r--r-- 1 jasmina jasmina    14 mar 21 17:11 naksa
drwxr-xr-x 3 jasmina jasmina  4096 mar 21 17:18 Pictures
-rwxrwxr-x 1 jasmina jasmina 16208 mar 21 17:14 piping
-rw-rw-r-- 1 jasmina jasmina   349 mar 21 17:10 piping.c
drwxr-xr-x 2 jasmina jasmina  4096 mar 13 12:09 Public
-rwxrwxr-x 1 jasmina jasmina 16592 mar 21 13:44 shellproject
-rw-rw-r-- 1 jasmina jasmina   840 mar 21 13:43 shellproject.c
drwx------ 3 jasmina jasmina  4096 mar 13 12:09 snap
-rwxrwxr-x 1 jasmina jasmina 16352 mar 21 17:26 syscalls
-rw-rw-r-- 1 jasmina jasmina   540 mar 21 17:25 syscalls.c
drwxr-xr-x 2 jasmina jasmina  4096 mar 13 12:09 Templates
-rw-rw-r-- 1 jasmina jasmina    14 mar 21 17:14 test.txt
drwxr-xr-x 2 jasmina jasmina  4096 mar 13 12:09 Videos
-rwxrwxr-x 1 jasmina jasmina 16232 mar 21 14:12 wc
-rw-rw-r-- 1 jasmina jasmina   738 mar 21 14:11 wc.c
Child process 0
jasmina@Ubuntu2:~$
```

clone and execle

- clone - similar to fork, but allows for specifying which parts of the process should be shared between parent and child
- execle - allows for specifying the environment for the new process
- child() function will be executed by the cloned process and print the message
- then the child() function uses execle() to execute ls -l command
- the main() function allocates the memory for the child process's stack using malloc()
- clone() is a system call that will create a new process with shared memory (CLONE_VM), then sends SIGCHILD signal to the parent process
- in clone() we specify the address of the child() function, the top of the stack (stack + 8192) and flags CLONE_VM and SIGCHILD
- after the clone() call we wait for the child to finish using wait()
- then the allocated stack memory is free

```
  GNU nano 6.2
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <sched.h>


int child(void *arg){
        printf("Child process with clone\n");


        char *args[] = {"/bin/ls", "-l", NULL};
        execle("/bin/ls", "ls", "-l", NULL, NULL);


        perror("execle");
        exit(EXIT_FAILURE);

}


int main(){
        char *stack = malloc(8192);


        int pid = clone(child,  stack + 8192, CLONE_VM | SIGCHLD, NULL);


        if(pid == -1){
                perror("clone");
                exit(EXIT_FAILURE);
        }


        wait(NULL);

        free(stack);

        return 0;
}
```

```
jasmina@Ubuntu2:~$ nano syscalls.c
jasmina@Ubuntu2:~$ gcc -o syscalls syscalls.c
jasmina@Ubuntu2:~$ ./syscalls
Child process with clone
total 272
drwxr-xr-x 3 jasmina jasmina  4096 Mar 16 07:55 Desktop
drwxr-xr-x 2 jasmina jasmina  4096 Mar 13 12:09 Documents
drwxr-xr-x 2 jasmina jasmina  4096 Mar 16 07:55 Downloads
drwxr-xr-x 2 jasmina jasmina  4096 Mar 13 12:09 Music
drwxr-xr-x 3 jasmina jasmina  4096 Mar 21 17:18 Pictures
drwxr-xr-x 2 jasmina jasmina  4096 Mar 13 12:09 Public
drwxr-xr-x 2 jasmina jasmina  4096 Mar 13 12:09 Templates
drwxr-xr-x 2 jasmina jasmina  4096 Mar 13 12:09 Videos
-rw-rw-r-- 1 jasmina jasmina 10650 Mar 13 12:37 assignment.txt
-rwxrwxr-x 1 jasmina jasmina 16008 Mar 13 12:18 calculator
-rw-rw-r-- 1 jasmina jasmina   309 Mar 13 12:18 calculator.c
-rw-rw-r-- 1 jasmina jasmina  1157 Mar 13 12:40 calculator.zip
-rw-rw-r-- 1 jasmina jasmina    14 Mar 21 17:13 cat
-rw-rw-r-- 1 jasmina jasmina  1157 Mar 13 12:40 cfiles.zip
-rwxrwxr-x 1 jasmina jasmina 16192 Mar 21 16:38 df
-rw-rw-r-- 1 jasmina jasmina   678 Mar 21 16:37 df.c
-rw-rw-r-- 1 jasmina jasmina    24 Mar 21 13:57 file.txt
-rw-rw-r-- 1 jasmina jasmina  1657 Mar 13 12:37 final.txt
-rwxrwxr-x 1 jasmina jasmina 16048 Mar 13 12:30 findSmallest
-rw-rw-r-- 1 jasmina jasmina   544 Mar 13 12:28 findSmallest.c
-rw-rw-r-- 1 jasmina jasmina   545 Mar 13 12:29 findvalue.c
-rwxrwxr-x 1 jasmina jasmina 16280 Mar 21 16:25 grep
-rw-rw-r-- 1 jasmina jasmina   543 Mar 21 16:20 grep.c
-rw-rw-r-- 1 jasmina jasmina    55 Mar 14 13:00 jasmina.c
-rwxrwxr-x 1 jasmina jasmina 16520 Mar 21 17:02 matrix
-rw-rw-r-- 1 jasmina jasmina   750 Mar 21 17:01 matrix.c
-rw-r--r-- 1 jasmina jasmina    14 Mar 21 17:11 naksa
-rwxrwxr-x 1 jasmina jasmina 16208 Mar 21 17:14 piping
-rw-rw-r-- 1 jasmina jasmina   349 Mar 21 17:10 piping.c
-rwxrwxr-x 1 jasmina jasmina 16592 Mar 21 13:44 shellproject
-rw-rw-r-- 1 jasmina jasmina   840 Mar 21 13:43 shellproject.c
drwx------ 3 jasmina jasmina  4096 Mar 13 12:09 snap
-rwxrwxr-x 1 jasmina jasmina 16288 Mar 21 17:41 syscalls
-rw-rw-r-- 1 jasmina jasmina   555 Mar 21 17:40 syscalls.c
-rw-rw-r-- 1 jasmina jasmina    14 Mar 21 17:14 test.txt
-rwxrwxr-x 1 jasmina jasmina 16232 Mar 21 14:12 wc
-rw-rw-r-- 1 jasmina jasmina   738 Mar 21 14:11 wc.c
jasmina@Ubuntu2:~$
```

forkbomb

- forkbomb is a malicious program which creates a large number of child processes very quick and that way can consume system resources
- so it creates a lot of child processes, which create more child processes, and so on
- it's malicious because it can quickly consume a large portion of resources and memory, which leads to program crashing
- forkbombs are considered a form of DDoS attacks and are mostly used by attackers to disable a system
- common representation of forkbomb in Bash shell `:(){:|:&};:`

here is an example of code, which has an infinite loop **while(1)** and inside the **fork()** system call is invoked, so it will create a new process by duplicating the current process

```
Terminal
GNU nano 6.2                    forkbomb.c
#include <unistd.h>

int main(){
        while(1){
                fork();
        }
        return 0;
}
```

4. Add some color to your shell and name it

```
jasmina@Ubuntu2:~ $sudo nano ~/.bashrc
jasmina@Ubuntu2:~ $source ~/.bashrc
jasmina@Ubuntu2:~ $sudo nano ~/.bashrc
jasmina@Ubuntu2:~ $source ~/.bashrc
jasmina@Ubuntu2:~ $sudo nano ~/.bashrc
jasmina@Ubuntu2:~ $source ~/.bashrc
jasmina@Ubuntu2:~ $
```

```
iman@Ubuntu2:~ $sudo nano ~/.bashrc
iman@Ubuntu2:~ $source ~/.bashrc
iman@Ubuntu2:~ $sudo nano ~/.bashrc
iman@Ubuntu2:~ $source ~/.bashrc
iman@Ubuntu2:~ $sudo nano ~/bashrc
iman@Ubuntu2:~ $sudo nano ~/.bashrc
iman@Ubuntu2:~ $source ~/.bashrc
jasmina@Ubuntu2:~ $sudo nano ~/.bashrc
jasmina@Ubuntu2:~ $source ~/.bashrc
jasmina@Ubuntu2:~ $sudo nano ~/.bashrc
jasmina@Ubuntu2:~ $source ~/.bashrc
jasmina@Ubuntu2:~ $
```

this was achieved by

- opening **~/.bashrc** (configuration file for the Bash shell)
- editing the PS1 (prompt string 1) variable to change username (**\u**), hostname (**\h**), and add color with ANSI escape codes
- applying the changes with **source ~/.bashrc**

5. Consider the following sample code from a simple shell program. Now, suppose the shell wishes to redirect the output of the command not to STDOUT but to a file “foo.txt”. Show how you would modify the above code to achieve this output redirection.

this program takes in user input *command* , executes it in a child process and displays the output in a file called *foo.txt*

Terminal

GNU nano 6.2 finaltask.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <string.h>

int main(){
        char command[100];

        printf("Choose a command  ");
        fgets(command, sizeof(command), stdin);

        command[strcspn(command, "\n")] = '\0';


        int rc = fork();
        if(rc == -1){
                perror("fork");
                exit(EXIT_FAILURE);
        }
        else if(rc==0){
                int fd = open("foo.txt", O_CREAT | O_WRONLY | O_TRUNC, 0644);
                if(fd == -1){
                        perror("open");
                        exit(EXIT_FAILURE);
                }

                if(dup2(fd, STDOUT_FILENO) == -1){
                        perror("dup2");
                        exit(EXIT_FAILURE);
                }


                close(fd);

                system(command);
                exit(EXIT_SUCCESS);
        }else{
                wait(NULL);
        }

        return 0;
}
```

testing out the program

commands: **df -h** (displays disk storage usage) and **ls -l** (lists files and directories in the current directory with information)

```
Terminal
jasmina@Ubuntu2:~ $nano finaltask.c
jasmina@Ubuntu2:~ $gcc -o finaltask finaltask.c
jasmina@Ubuntu2:~ $nano foo.txt
jasmina@Ubuntu2:~ $cat foo.txt
this is the foo.txt file
jasmina@Ubuntu2:~ $./finaltask
Choose a command  df -h
jasmina@Ubuntu2:~ $cat foo.txt
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           196M  1,5M  195M   1% /run
/dev/sda3        24G   12G   11G  52% /
tmpfs           980M     0  980M   0% /dev/shm
tmpfs           5,0M  4,0K  5,0M   1% /run/lock
/dev/sda2       512M  6,1M  506M   2% /boot/efi
tmpfs           196M  128K  196M   1% /run/user/1001
jasmina@Ubuntu2:~ $./finaltask
Choose a command  ls -l
jasmina@Ubuntu2:~ $cat foo.txt
total 316
-rw-rw-r-- 1 jasmina jasmina 10650 mar 13 12:37 assignment.txt
-rwxrwxr-x 1 jasmina jasmina 16008 mar 13 12:18 calculator
-rw-rw-r-- 1 jasmina jasmina   309 mar 13 12:18 calculator.c
-rw-rw-r-- 1 jasmina jasmina  1157 mar 13 12:40 calculator.zip
-rw-rw-r-- 1 jasmina jasmina    14 mar 21 17:13 cat
-rw-rw-r-- 1 jasmina jasmina  1157 mar 13 12:40 cfiles.zip
drwxr-xr-x 3 jasmina jasmina  4096 mar 16 07:55 Desktop
-rwxrwxr-x 1 jasmina jasmina 16192 mar 21 16:38 df
-rw-rw-r-- 1 jasmina jasmina   678 mar 21 16:37 df.c
drwxr-xr-x 2 jasmina jasmina  4096 mar 13 12:09 Documents
drwxr-xr-x 2 jasmina jasmina  4096 mar 16 07:55 Downloads
-rw-rw-r-- 1 jasmina jasmina    24 mar 21 13:57 file.txt
-rwxrwxr-x 1 jasmina jasmina 16480 mar 22 09:20 finaltask
-rw-rw-r-- 1 jasmina jasmina   669 mar 22 09:19 finaltask.c
-rw-rw-r-- 1 jasmina jasmina  1657 mar 13 12:37 final.txt
-rwxrwxr-x 1 jasmina jasmina 16048 mar 13 12:30 findSmallest
-rw-rw-r-- 1 jasmina jasmina   544 mar 13 12:28 findSmallest.c
-rw-rw-r-- 1 jasmina jasmina   545 mar 13 12:29 findvalue.c
-rw-rw-r-- 1 jasmina jasmina     0 mar 22 09:21 foo.txt
-rwxrwxr-x 1 jasmina jasmina 16096 mar 22 08:03 forkalone
-rw-rw-r-- 1 jasmina jasmina   213 mar 22 08:02 forkalone.c
-rwxrwxr-x 1 jasmina jasmina 16280 mar 21 16:25 grep
-rw-rw-r-- 1 jasmina jasmina   543 mar 21 16:20 grep.c
-rw-rw-r-- 1 jasmina jasmina    55 mar 14 13:00 jasmina.c
-rwxrwxr-x 1 jasmina jasmina 16520 mar 21 17:02 matrix
-rw-rw-r-- 1 jasmina jasmina   750 mar 21 17:01 matrix.c
drwxr-xr-x 2 jasmina jasmina  4096 mar 13 12:09 Music
-rw-r--r-- 1 jasmina jasmina    14 mar 21 17:11 naksa
drwxr-xr-x 3 jasmina jasmina  4096 mar 21 17:18 Pictures
-rwxrwxr-x 1 jasmina jasmina 16208 mar 21 17:14 piping
-rw-rw-r-- 1 jasmina jasmina   349 mar 21 17:10 piping.c
```

command: **du** (estimates file space usage)

```
jasmina@Ubuntu2:~ $./finaltask
Choose a command  du
jasmina@Ubuntu2:~ $cat foo.txt
4       ./Downloads
28      ./snap/snapd-desktop-integration/common/.cache/gio-modules
76      ./snap/snapd-desktop-integration/common/.cache/immodules
112     ./snap/snapd-desktop-integration/common/.cache
116     ./snap/snapd-desktop-integration/common
4       ./snap/snapd-desktop-integration/83/.config/gtk-3.0
4       ./snap/snapd-desktop-integration/83/.config/gtk-2.0
4       ./snap/snapd-desktop-integration/83/.config/ibus
4       ./snap/snapd-desktop-integration/83/.config/dconf
8       ./snap/snapd-desktop-integration/83/.config/fontconfig
44      ./snap/snapd-desktop-integration/83/.config
4       ./snap/snapd-desktop-integration/83/.local/share/glib-2.0/schemas
8       ./snap/snapd-desktop-integration/83/.local/share/glib-2.0
4       ./snap/snapd-desktop-integration/83/.local/share/icons
16      ./snap/snapd-desktop-integration/83/.local/share
20      ./snap/snapd-desktop-integration/83/.local
72      ./snap/snapd-desktop-integration/83
192     ./snap/snapd-desktop-integration
196     ./snap
96      ./Pictures/Screenshots
100     ./Pictures
84      ./.config/pulse
8       ./.config/nautilus
4       ./.config/goa-1.0
8       ./.config/gtk-3.0
8       ./.config/evolution/sources
12      ./.config/evolution
8       ./.config/gedit
4       ./.config/enchant
12      ./.config/ibus/bus
16      ./.config/ibus
4       ./.config/gnome-session/saved-session
8       ./.config/gnome-session
12      ./.config/dconf
4       ./.config/update-notifier
8       ./.config/yelp
192     ./.config
```

command: **ss** (socket statistics, network socket information)

```
Terminal
jasmina@Ubuntu2:~ $./finaltask
Choose a command  ss
jasmina@Ubuntu2:~ $cat foo.txt
Netid State  Recv-Q Send-Q                          Local Address:Port      Peer Address:Port  Process
u_str ESTAB  272    0                                            * 27047                 * 27051
u_str ESTAB  0      0                                            * 25893                 * 25894
u_str ESTAB  0      0                                            * 19787                 * 19788
u_str ESTAB  0      0                   /run/systemd/journal/stdout 19461                 * 19453
u_str ESTAB  0      0                   /run/systemd/journal/stdout 24875                 * 24869
u_str ESTAB  0      0                                            * 26988                 * 26990
u_str ESTAB  0      0                                            * 26574                 * 26577
u_str ESTAB  0      0                   /run/systemd/journal/stdout 25713                 * 25711
u_dgr ESTAB  0      0                                            * 23057                 * 23058
u_str ESTAB  0      0                      /run/user/1001/wayland-0 26969                 * 26638
u_str ESTAB  0      0                                            * 26526                 * 26530
u_str ESTAB  0      0                                            * 18841                 * 19111
u_str ESTAB  0      0                      /run/user/1001/wayland-0 26973                 * 26713
u_str ESTAB  0      0                                            * 25525                 * 25529
u_str ESTAB  0      0                                            * 25238                 * 25242
u_str ESTAB  0      0                   /run/systemd/journal/stdout 24601                 * 24597
u_str ESTAB  0      0                                            * 19325                 * 19326
u_str ESTAB  0      0      @/home/jasmina/.cache/ibus/dbus-Ainf4neT 27208                 * 27207
u_str ESTAB  0      0                                            * 25627                 * 25641
u_dgr ESTAB  0      0                         /run/systemd/notify 16667                 * 0
u_str ESTAB  0      0                                            * 24906                 * 24907
u_str ESTAB  0      0                                            * 27082                 * 27095
u_str ESTAB  0      0                      /run/user/1001/wayland-0 26976                 * 26960
u_str ESTAB  0      0                   /run/systemd/journal/stdout 19269                 * 19268
u_str ESTAB  0      0                   /run/systemd/journal/stdout 17959                 * 17954
u_str ESTAB  0      0                            /run/user/1001/bus 26712                 * 26704
u_str ESTAB  0      0                            /run/user/1001/bus 26684                 * 26682
u_str ESTAB  0      0                   /run/systemd/journal/stdout 25061                 * 25060
u_str ESTAB  0      0                                            * 23578                 * 23579
```

commands: **uname -a** (displays info about the user) , **ps** (info about active processes) and **ps aux** (info about all processes running on the system)

```
jasmina@Ubuntu2:~ $./finaltask
Choose a command  uname -a
jasmina@Ubuntu2:~ $cat foo.txt
Linux Ubuntu2 6.5.0-21-generic #21~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Fri Feb  9 13:32:52 UTC 2 x86_64 x86_64 x86_64 GNU/Linux
jasmina@Ubuntu2:~ $./finaltask
Choose a command  ps
jasmina@Ubuntu2:~ $cat foo.txt
    PID TTY          TIME CMD
 195862 pts/0    00:00:00 bash
 196995 pts/0    00:00:00 finaltask
 196996 pts/0    00:00:00 finaltask
 196997 pts/0    00:00:00 sh
 196998 pts/0    00:00:00 ps
jasmina@Ubuntu2:~ $./finaltask
Choose a command  ps aux
jasmina@Ubuntu2:~ $cat foo.txt
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.5 166680 11096 ?        Ss   07:54   0:03 /sbin/init splash
root           2  0.0  0.0      0     0 ?        S    07:54   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   07:54   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   07:54   0:00 [rcu_par_gp]
root           5  0.0  0.0      0     0 ?        I<   07:54   0:00 [slub_flushwq]
root           6  0.0  0.0      0     0 ?        I<   07:54   0:00 [netns]
root           8  0.0  0.0      0     0 ?        I<   07:54   0:00 [kworker/0:0H-kblockd]
root          11  0.0  0.0      0     0 ?        I<   07:54   0:00 [mm_percpu_wq]
root          12  0.0  0.0      0     0 ?        I    07:54   0:00 [rcu_tasks_kthread]
root          13  0.0  0.0      0     0 ?        I    07:54   0:00 [rcu_tasks_rude_kthread]
root          14  0.0  0.0      0     0 ?        I    07:54   0:00 [rcu_tasks_trace_kthread]
root          15  0.0  0.0      0     0 ?        S    07:54   0:01 [ksoftirqd/0]
root          16  0.0  0.0      0     0 ?        I    07:54   0:04 [rcu_preempt]
root          17  0.0  0.0      0     0 ?        S    07:54   0:00 [migration/0]
root          18  0.0  0.0      0     0 ?        S    07:54   0:00 [idle_inject/0]
root          19  0.0  0.0      0     0 ?        S    07:54   0:00 [cpuhp/0]
root          20  0.0  0.0      0     0 ?        S    07:54   0:00 [kdevtmpfs]
root          21  0.0  0.0      0     0 ?        I<   07:54   0:00 [inet_frag_wq]
root          22  0.0  0.0      0     0 ?        S    07:54   0:00 [kauditd]
root          24  0.0  0.0      0     0 ?        S    07:54   0:00 [khungtaskd]
root          25  0.0  0.0      0     0 ?        S    07:54   0:00 [oom_reaper]
root          27  0.0  0.0      0     0 ?        I<   07:54   0:00 [writeback]
root          28  0.0  0.0      0     0 ?        S    07:54   0:01 [kcompactd0]
root          29  0.0  0.0      0     0 ?        SN   07:54   0:00 [ksmd]
root          30  0.0  0.0      0     0 ?        SN   07:54   0:00 [khugepaged]
root          31  0.0  0.0      0     0 ?        I<   07:54   0:00 [kintegrityd]
root          32  0.0  0.0      0     0 ?        I<   07:54   0:00 [kblockd]
root          33  0.0  0.0      0     0 ?        I<   07:54   0:00 [blkcg_punt_bio]
root          34  0.0  0.0      0     0 ?        I<   07:54   0:00 [tpm_dev_wq]
root          35  0.0  0.0      0     0 ?        I<   07:54   0:00 [ata_sff]
```

command: ls -a (lists files and directories including the hidden ones)

```
jasmina@Ubuntu2:~ $./finaltask
Choose a command  ls -a
jasmina@Ubuntu2:~ $cat foo.txt
.
..
assignment.txt
.bash_history
.bash_logout
.bashrc
.cache
calculator
calculator.c
calculator.zip
cat
cfiles.zip
.config
Desktop
df
df.c
Documents
Downloads
file.txt
finaltask
finaltask.c
final.txt
findSmallest
findSmallest.c
findvalue.c
foo.txt
forkalone
forkalone.c
grep
grep.c
jasmina.c
.lesshst
.local
matrix
matrix.c
Music
naksa
.pam_environment
Pictures
piping
piping.c
.profile
Public
shellproject
shellproject.c
snap
.sudo_as_admin_successful
syscalls
```

resources

- What is Multi Core CPU? E learning animation video
- Multi-Programming Operating System | Easy Explaination using Animation
- Understanding Fork Bombs in 5 Minutes or Less
- Linux Fork Bomb "Virus"
- 60 Linux Commands you NEED to know (in 10 minutes)
- System Calls
- https://skillsforall.com/course/operating-systems-basics?courseLang=en-US