CSE 344

HW2

Burak Kocausta

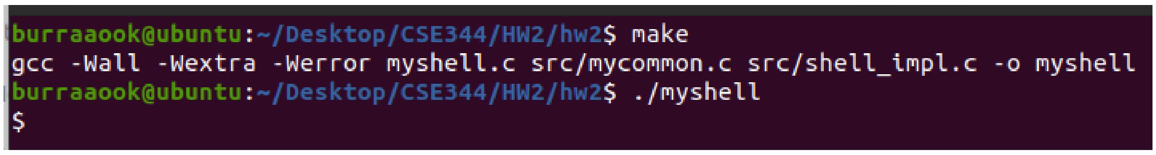
1901042605

Content

1- Compilation and Run

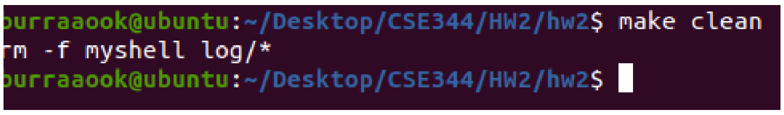
All parts are compiled with ‘make’ command.

Compilation and Run



```
burraaook@ubuntu:~/Desktop/CSE344/HW2/hw2$ make
gcc -Wall -Wextra -Werror myshell.c src/mycommon.c src/shell_impl.c -o myshell
burraaook@ubuntu:~/Desktop/CSE344/HW2/hw2$ ./myshell
$
```

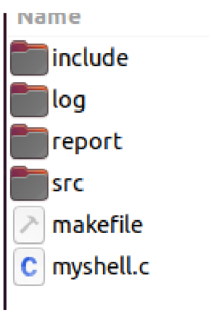
Clean



```
burraaook@ubuntu:~/Desktop/CSE344/HW2/hw2$ make clean
rm -f myshell log/*
burraaook@ubuntu:~/Desktop/CSE344/HW2/hw2$ 
```

- clean also removes logged files

2- Files



```
Name
include
log
report
src
makefile
myshell.c
```

include: header files

log: .txt log files

report: pdf report

src: implementation of header files

myshell.c: driver program

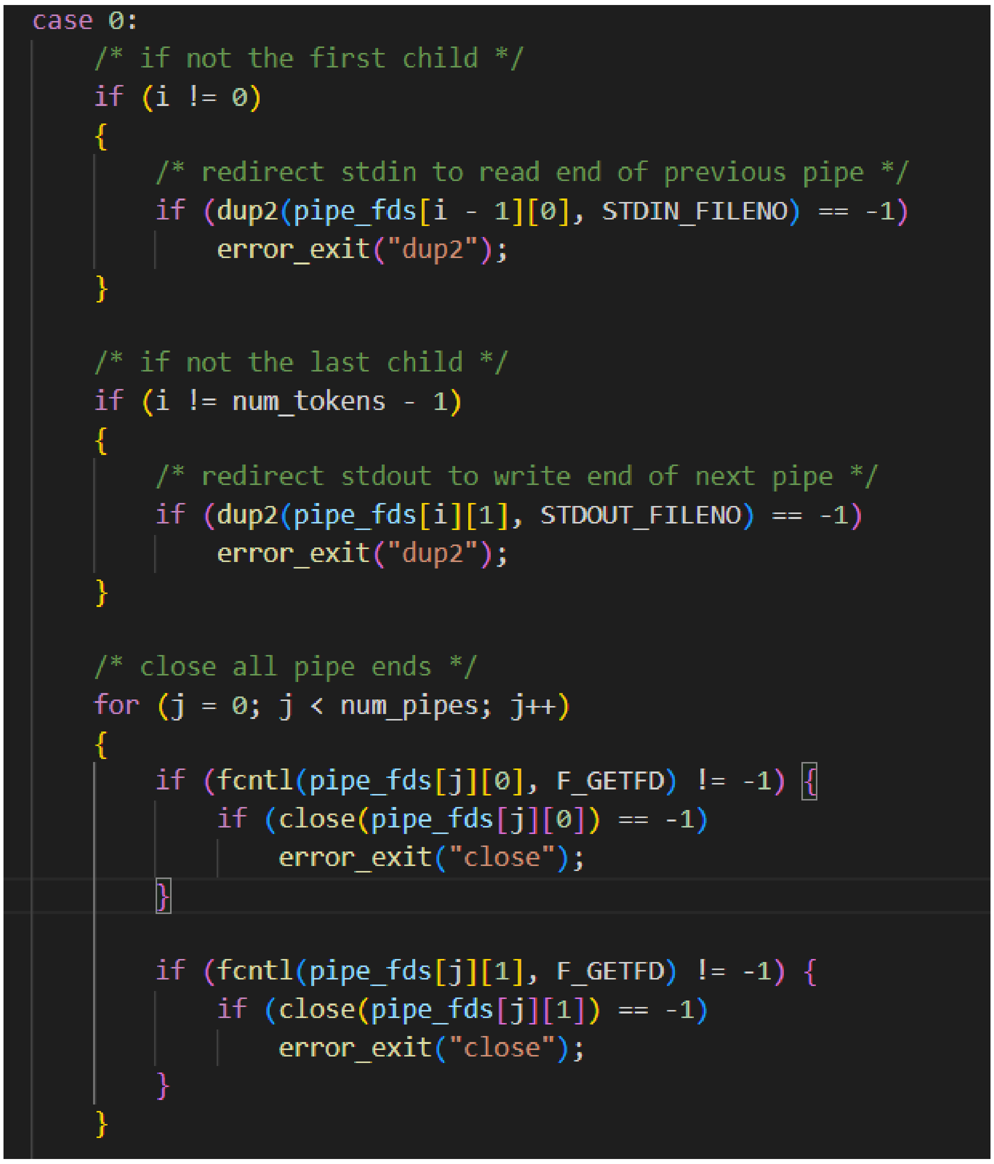
3- Detailed Explanation of The Solution Approach

1) General

- I created myshell, and shell_impl files, and made my implementations on those files. myshell is the driver program. I continued using mycommon file from previous homework. This program gets user command, then parses it according to “|”, “<”, “>” characters, and executes. Between every command there must be “|” character. Every command is another process which is also child of the parent process. After I parse, the commands according to the pipe, fork is used for creating another process, then another parsing is done for the “<”, “>” operators. Then I changed the process image for that child process to execute shell command. The general flow of the program is like that, other than that signal handling, error handling, and logging is done. Logs are in the log directory. I used one of the exec-family system calls, to change the core image. It is said execl in the instructions, but we are informed during lecture about other exec-family functions are not prohibited. During program execution with multiple processes, only the child processes I created can be seen. I don’t invoke the commands through the sh process, they are created with simple fork-exec.

2) Piping

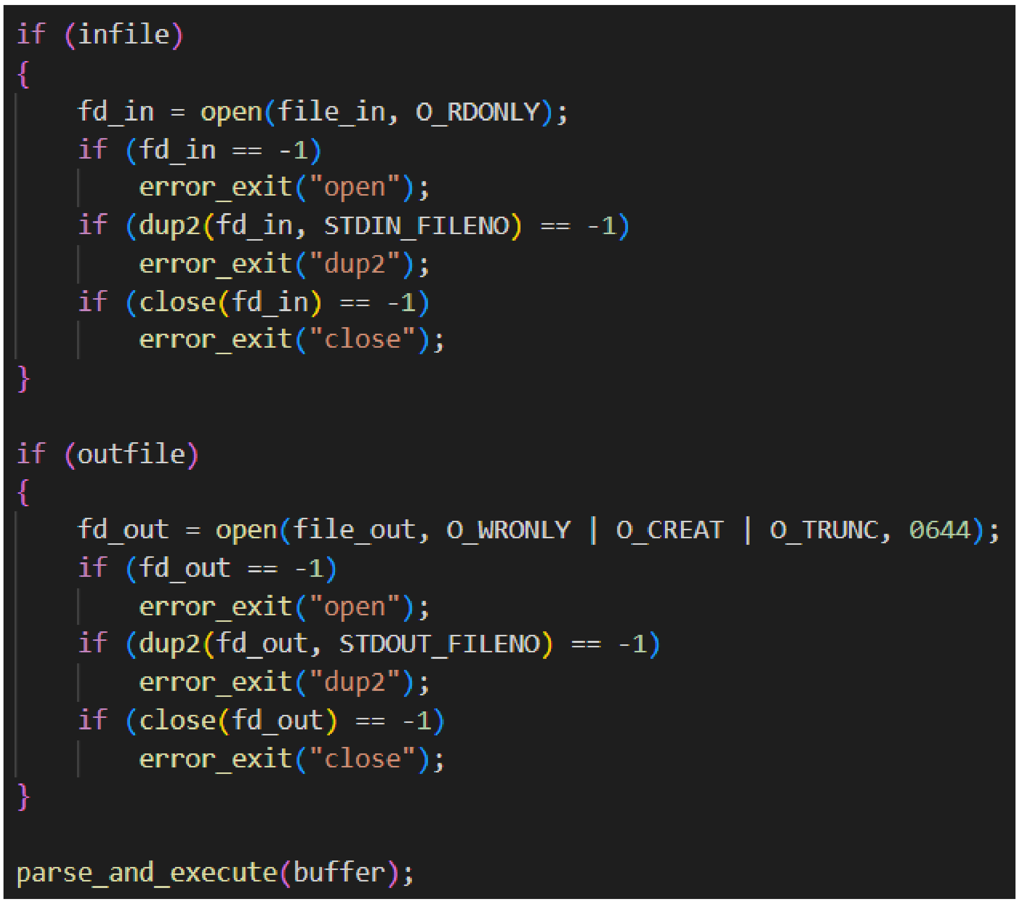
- Every command between pipes is another process, so every command between pipes must be created using fork. After child processes are created using fork, they are connected using pipe system call. When I am connecting the processes, I used 2 conditions which are checking for if it is first child, or last child. With those conditions there are 3 different operations for first, last, and middle children. It is required while connecting their read, write ends. I redirected pipe fds using dup2 to stdin or stdout. Then, I closed all the unused file descriptors. Parent process holds the pipe end till the child process comes. Because of not leaving pipes unconnected, parent holds it till child takes. Parent forks children in order. It must be like this because, bytes are flowing on the between pipes, if one process tries to read or write before turn come to it, error will occur. Therefore, in the parent process this order is ensured. Also, if there are n commands, there must be n-1 pipes. For one command, as expected there are no pipe.



```
case 0:
    /* if not the first child */
    if (i != 0)
    {
        /* redirect stdin to read end of previous pipe */
        if (dup2(pipe_fds[i - 1][0], STDIN_FILENO) == -1)
            error_exit("dup2");
    }

    /* if not the last child */
    if (i != num_tokens - 1)
    {
        /* redirect stdout to write end of next pipe */
        if (dup2(pipe_fds[i][1], STDOUT_FILENO) == -1)
            error_exit("dup2");
    }

    /* close all pipe ends */
    for (j = 0; j < num_pipes; j++)
    {
        if (fcntl(pipe_fds[j][0], F_GETFD) != -1) {
            if (close(pipe_fds[j][0]) == -1)
                error_exit("close");
        }

        if (fcntl(pipe_fds[j][1], F_GETFD) != -1) {
            if (close(pipe_fds[j][1]) == -1)
                error_exit("close");
        }
    }
```

3) Redirection Operator

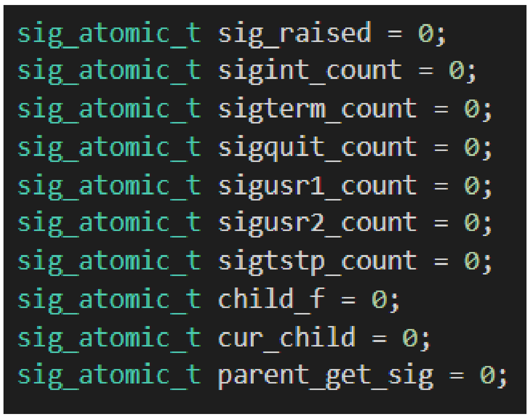
- After parsing according to the pipe operator is done. This means that there could be only “<”, “>” operators. So next step is parsing according to redirection operators. I do this in child process, it would be unnecessary to do this in parent, because too many token could be accumulated in parent memory. I created a function for this operation which is called handle_redirect(). I checked both “<”, ”>” operators in a loop, because there could be more than one operator for those. Then I redirected their stdin, and stdout according to which of the redirection operator it is. I used these loops for determining where the filename is positioned. It changes with the type of redirection, so I searched it separately. After finding them, I parsed the command which is easy because it is on the start position. So, if there are redirection operators exists, I redirect them to stdin or stdout (could be both) with using dup2. Then they are discarded, only command is left from there.



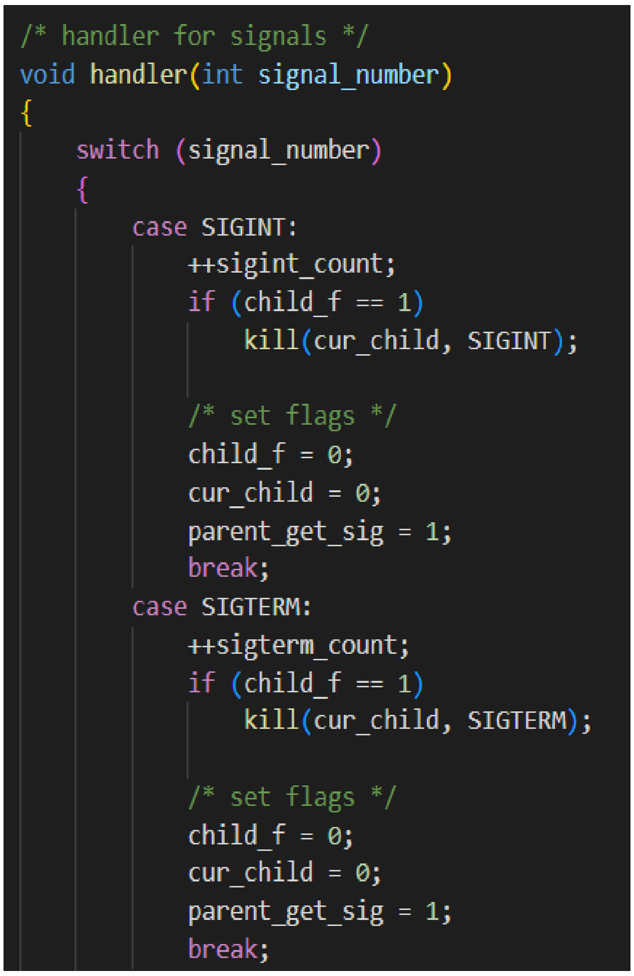
```
if (infile)
{
    fd_in = open(file_in, O_RDONLY);
    if (fd_in == -1)
        error_exit("open");
    if (dup2(fd_in, STDIN_FILENO) == -1)
        error_exit("dup2");
    if (close(fd_in) == -1)
        error_exit("close");
}

if (outfile)
{
    fd_out = open(file_out, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd_out == -1)
        error_exit("open");
    if (dup2(fd_out, STDOUT_FILENO) == -1)
        error_exit("dup2");
    if (close(fd_out) == -1)
        error_exit("close");
}

parse_and_execute(buffer);
```

4) Signal Handling

- I write my signal handler in myshell file. It works for parent processes, and because handlers should not be heavy operations, I changed the global signal variables on that handler according to the coming signal. Then I do my things in the flow of my program according to those flags. For the other flags, those who terminate child processes, I am checking the status on the wait function. If it is sigkill signal, I clean the memory then terminate the process. If it is sigint, or sigterm, I again clean the memory, but this time process returns to prompt again. If parent gets sigint, sigterm, and sigquit, it terminates all child cleans up and return to prompt.



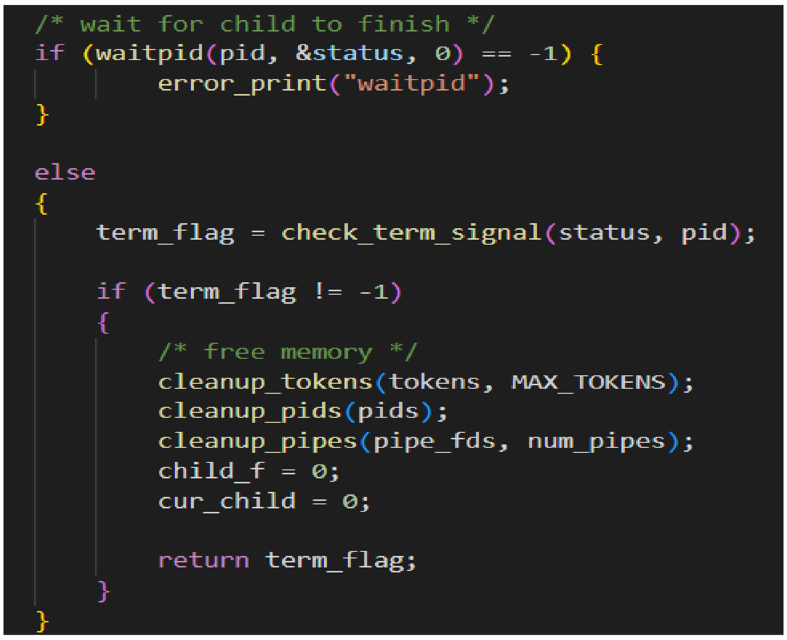
```c
sig_atomic_t sig_raised = 0;
sig_atomic_t sigint_count = 0;
sig_atomic_t sigterm_count = 0;
sig_atomic_t sigquit_count = 0;
sig_atomic_t sigusr1_count = 0;
sig_atomic_t sigusr2_count = 0;
sig_atomic_t sigtstp_count = 0;
sig_atomic_t child_f = 0;
sig_atomic_t cur_child = 0;
sig_atomic_t parent_get_sig = 0;
```



```c
/* handler for signals */
void handler(int signal_number)
{
    switch (signal_number)
    {
        case SIGINT:
            ++sigint_count;
            if (child_f == 1)
                kill(cur_child, SIGINT);

            /* set flags */
            child_f = 0;
            cur_child = 0;
            parent_get_sig = 1;
            break;
        case SIGTERM:
            ++sigterm_count;
            if (child_f == 1)
                kill(cur_child, SIGTERM);

            /* set flags */
            child_f = 0;
            cur_child = 0;
            parent_get_sig = 1;
            break;
```

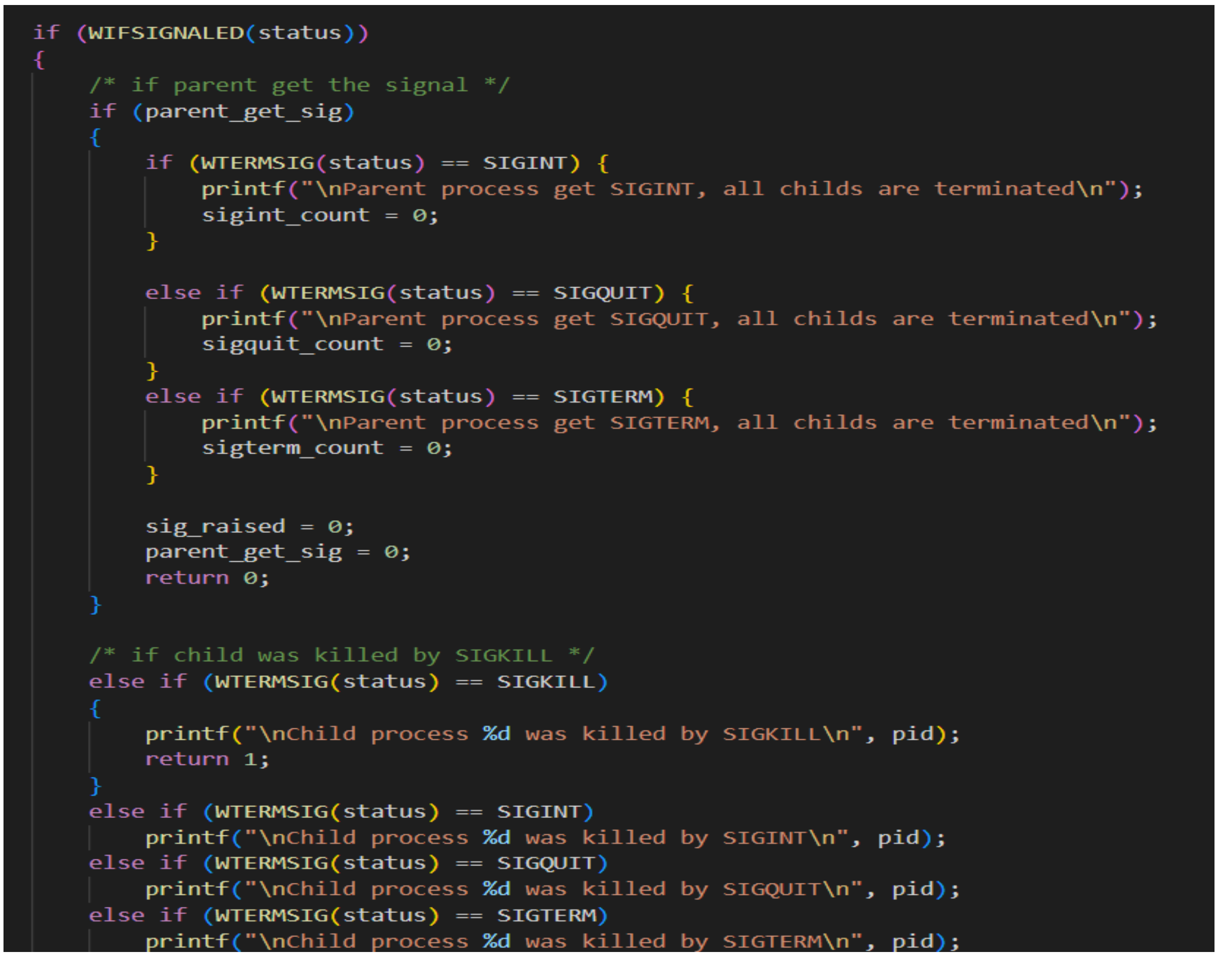
- Signal flags

- setting flags, and killing child in handler.



```c
/* wait for child to finish */
if (waitpid(pid, &status, 0) == -1) {
        error_print("waitpid");
}

else
{
    term_flag = check_term_signal(status, pid);

    if (term_flag != -1)
    {
        /* free memory */
        cleanup_tokens(tokens, MAX_TOKENS);
        cleanup_pids(pids);
        cleanup_pipes(pipe_fds, num_pipes);
        child_f = 0;
        cur_child = 0;

        return term_flag;
    }
}
```

- Checking termination of child.



```c
if (WIFSIGNALED(status))
{
    /* if parent get the signal */
    if (parent_get_sig)
    {
        if (WTERMSIG(status) == SIGINT) {
            printf("\nParent process get SIGINT, all childs are terminated\n");
            sigint_count = 0;
        }

        else if (WTERMSIG(status) == SIGQUIT) {
            printf("\nParent process get SIGQUIT, all childs are terminated\n");
            sigquit_count = 0;
        }
        else if (WTERMSIG(status) == SIGTERM) {
            printf("\nParent process get SIGTERM, all childs are terminated\n");
            sigterm_count = 0;
        }

        sig_raised = 0;
        parent_get_sig = 0;
        return 0;
    }

    /* if child was killed by SIGKILL */
    else if (WTERMSIG(status) == SIGKILL)
    {
        printf("\nChild process %d was killed by SIGKILL\n", pid);
        return 1;
    }
    else if (WTERMSIG(status) == SIGINT)
        printf("\nChild process %d was killed by SIGINT\n", pid);
    else if (WTERMSIG(status) == SIGQUIT)
        printf("\nChild process %d was killed by SIGQUIT\n", pid);
    else if (WTERMSIG(status) == SIGTERM)
        printf("\nChild process %d was killed by SIGTERM\n", pid);
```

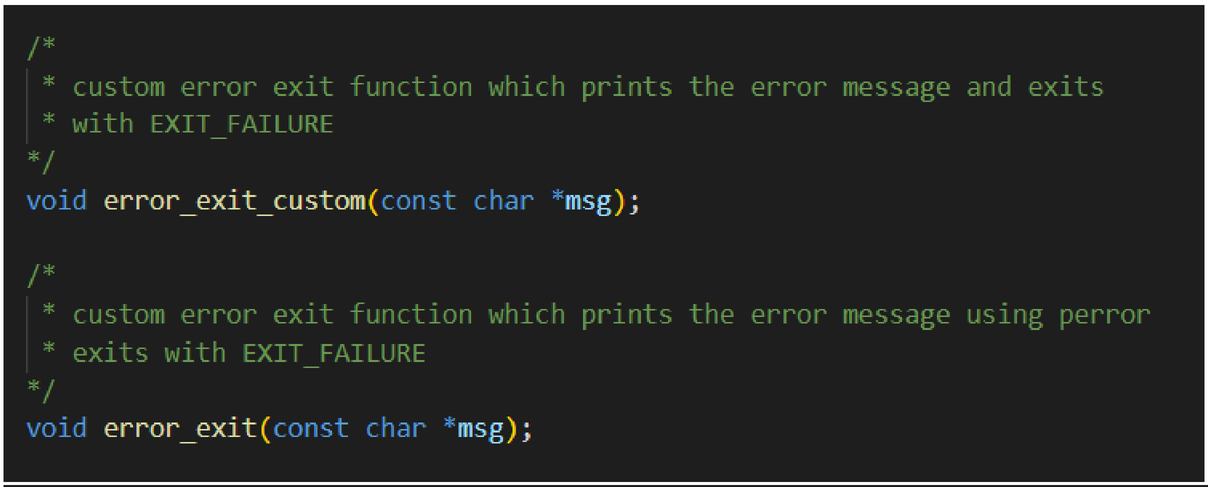
- Determining who got the signal with return status, and flags.

5) Logging

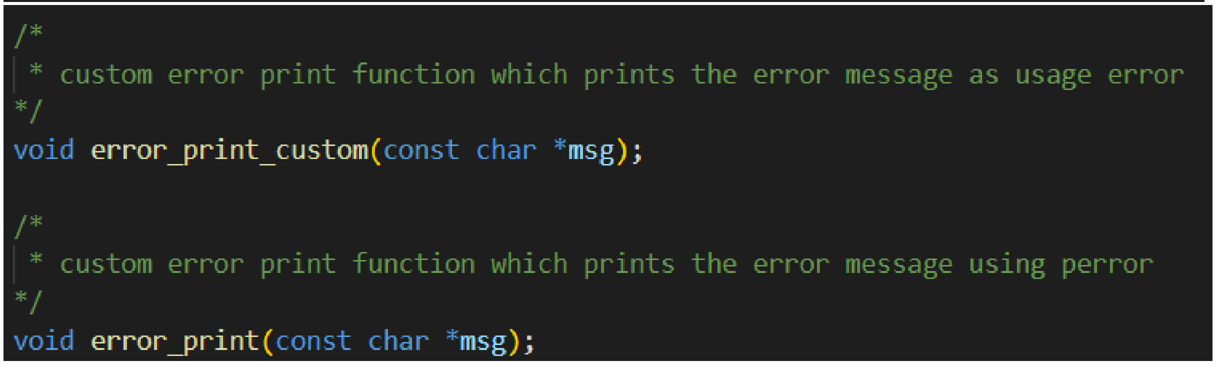
- I log the executions according to the timestamp. filename is timestamp, inside file there are pid, and the command. Because of executions are executed so fast, child processes tries to write same file. Because file name is timestamp. So, I opened this in append mode. If they want to write on the same time, information are appended to end of the file. I used fcntl lock during writing in case myshell runs more than one time, this is an extra precaution.

6) Error Handling

- If an error occurs on the parent process due to system calls, I printed the error using perror, and cleanup everything then returned. If error is occurred on the parent process due to usage error, I printed the usage error then returned. Other possibilities are errors might occur on the child process, I terminated them with necessary cleanups.



```
/*
 * custom error exit function which prints the error message and exits
 * with EXIT_FAILURE
*/
void error_exit_custom(const char *msg);

/*
 * custom error exit function which prints the error message using perror
 * exits with EXIT_FAILURE
*/
void error_exit(const char *msg);
```

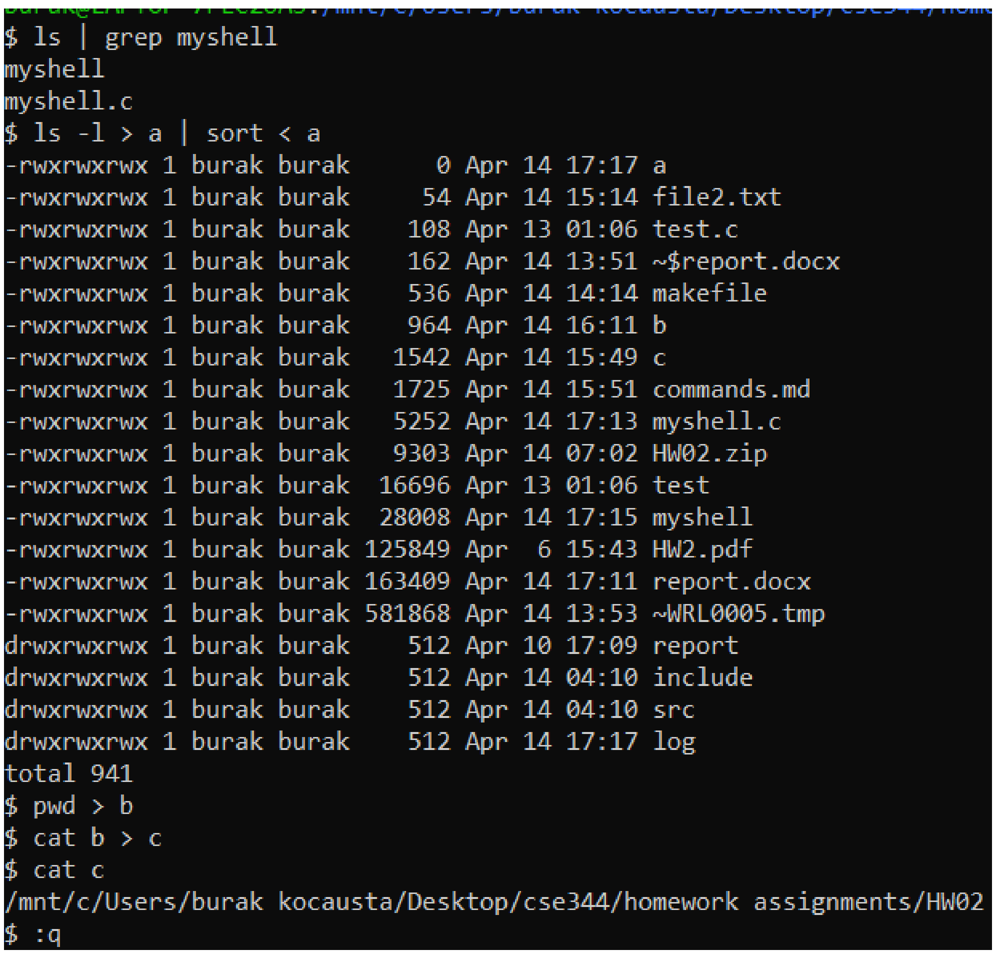


```
/*
 * custom error print function which prints the error message as usage error
*/
void error_print_custom(const char *msg);

/*
 * custom error print function which prints the error message using perror
*/
void error_print(const char *msg);
```

4- Test Results

- I tested some commands, signals, also log file is shown. I shared memory result of cases.

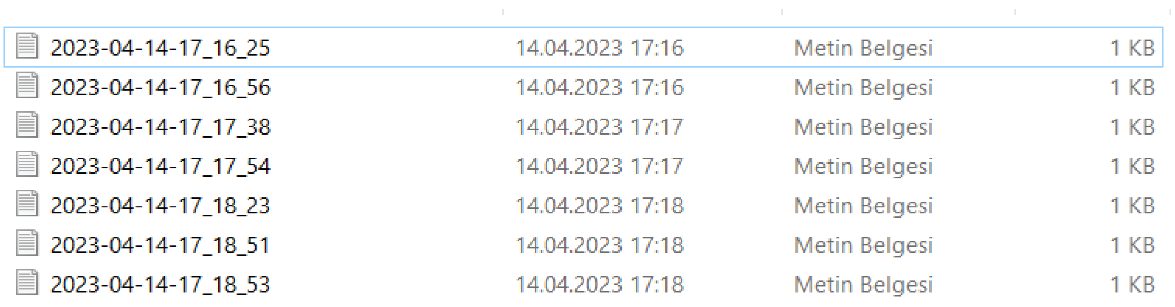
Test – commands 1:



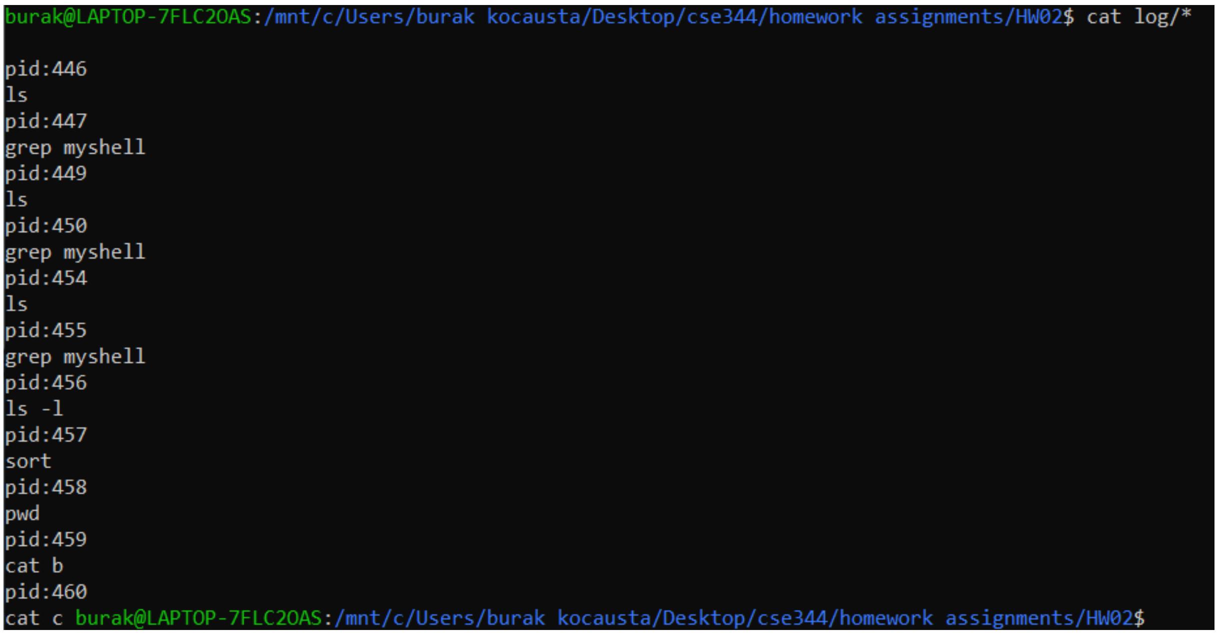
```
$ ls | grep myshell
myshell
myshell.c
$ ls -l > a | sort < a
-rwxrwxrwx 1 burak burak      0 Apr 14 17:17 a
-rwxrwxrwx 1 burak burak     54 Apr 14 15:14 file2.txt
-rwxrwxrwx 1 burak burak    108 Apr 13 01:06 test.c
-rwxrwxrwx 1 burak burak    162 Apr 14 13:51 ~$report.docx
-rwxrwxrwx 1 burak burak    536 Apr 14 14:14 makefile
-rwxrwxrwx 1 burak burak    964 Apr 14 16:11 b
-rwxrwxrwx 1 burak burak   1542 Apr 14 15:49 c
-rwxrwxrwx 1 burak burak   1725 Apr 14 15:51 commands.md
-rwxrwxrwx 1 burak burak   5252 Apr 14 17:13 myshell.c
-rwxrwxrwx 1 burak burak   9303 Apr 14 07:02 HW02.zip
-rwxrwxrwx 1 burak burak  16696 Apr 13 01:06 test
-rwxrwxrwx 1 burak burak  28008 Apr 14 17:15 myshell
-rwxrwxrwx 1 burak burak 125849 Apr  6 15:43 HW2.pdf
-rwxrwxrwx 1 burak burak 163409 Apr 14 17:11 report.docx
-rwxrwxrwx 1 burak burak 581868 Apr 14 13:53 ~WRL0005.tmp
drwxrwxrwx 1 burak burak    512 Apr 10 17:09 report
drwxrwxrwx 1 burak burak    512 Apr 14 04:10 include
drwxrwxrwx 1 burak burak    512 Apr 14 04:10 src
drwxrwxrwx 1 burak burak    512 Apr 14 17:17 log
total 941
$ pwd > b
$ cat b > c
$ cat c
/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02
$ :q
```

- Simple commands that connected with pipe, and redirection. First one greps myshell from ls, other one writes ls -l to a file, than sorts that file. Lastly, pwd is written to b file, then it is redirected to c, and c content is pwd.

Log:



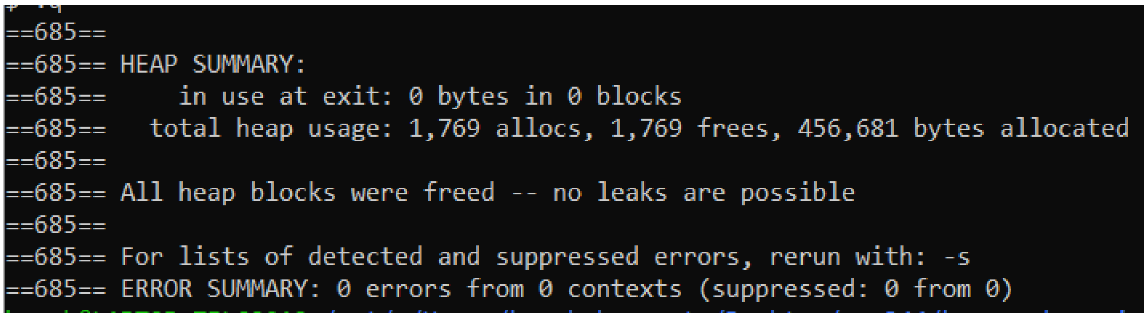
2023-04-14-17_16_25	14.04.2023 17:16	Metin Belgesi	1 KB
2023-04-14-17_16_56	14.04.2023 17:16	Metin Belgesi	1 KB
2023-04-14-17_17_38	14.04.2023 17:17	Metin Belgesi	1 KB
2023-04-14-17_17_54	14.04.2023 17:17	Metin Belgesi	1 KB
2023-04-14-17_18_23	14.04.2023 17:18	Metin Belgesi	1 KB
2023-04-14-17_18_51	14.04.2023 17:18	Metin Belgesi	1 KB
2023-04-14-17_18_53	14.04.2023 17:18	Metin Belgesi	1 KB



```
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$ cat log/*

pid:446
ls
pid:447
grep myshell
pid:449
ls
pid:450
grep myshell
pid:454
ls
pid:455
grep myshell
pid:456
ls -l
pid:457
sort
pid:458
pwd
pid:459
cat b
pid:460
cat c burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$
```

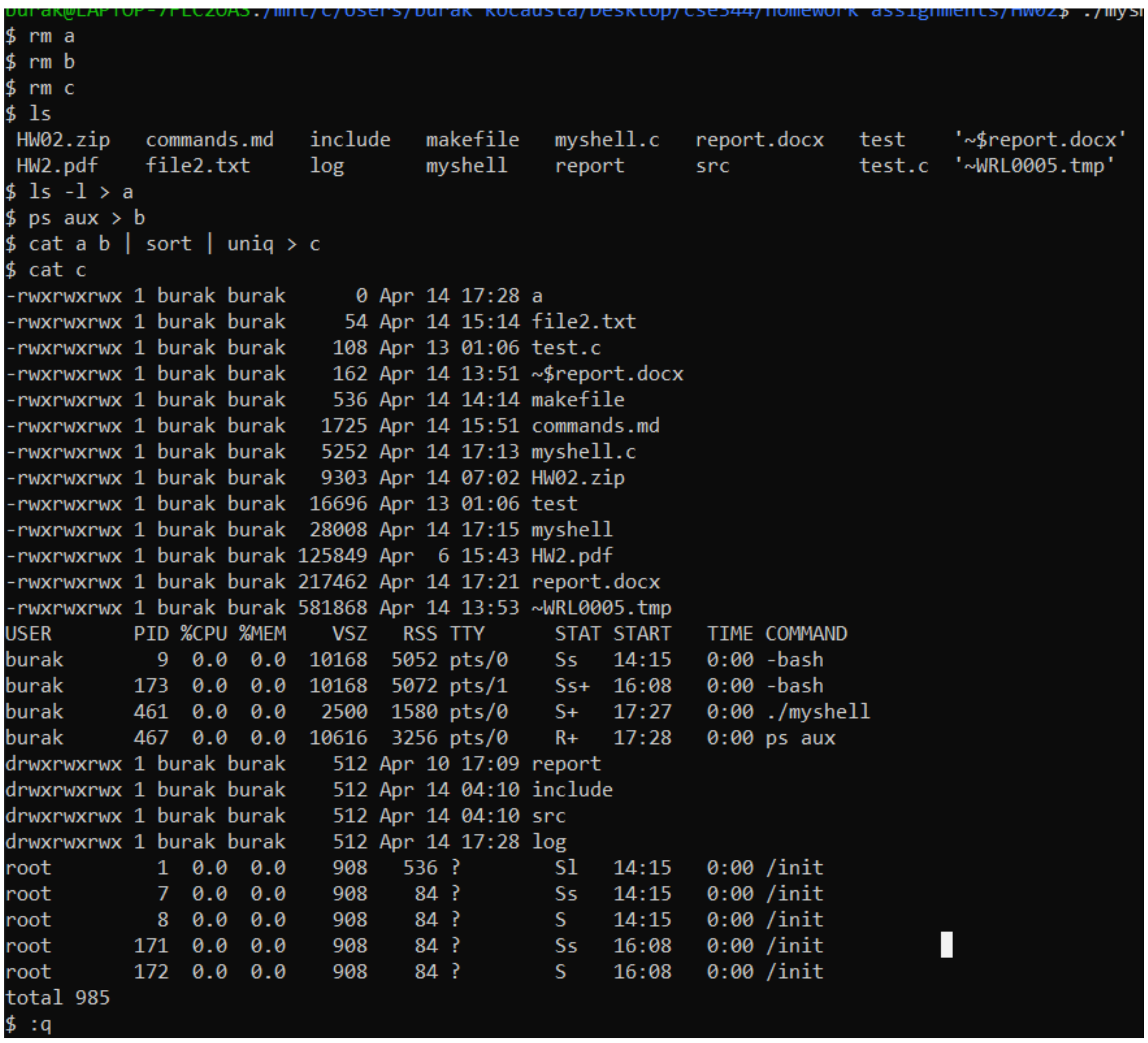
- Commands are added to log file as expected. Those are in log directory.

Valgrind:



```
==685==
==685== HEAP SUMMARY:
==685==     in use at exit: 0 bytes in 0 blocks
==685==   total heap usage: 1,769 allocs, 1,769 frees, 456,681 bytes allocated
==685==
==685== All heap blocks were freed -- no leaks are possible
==685==
==685== For lists of detected and suppressed errors, rerun with: -s
==685== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

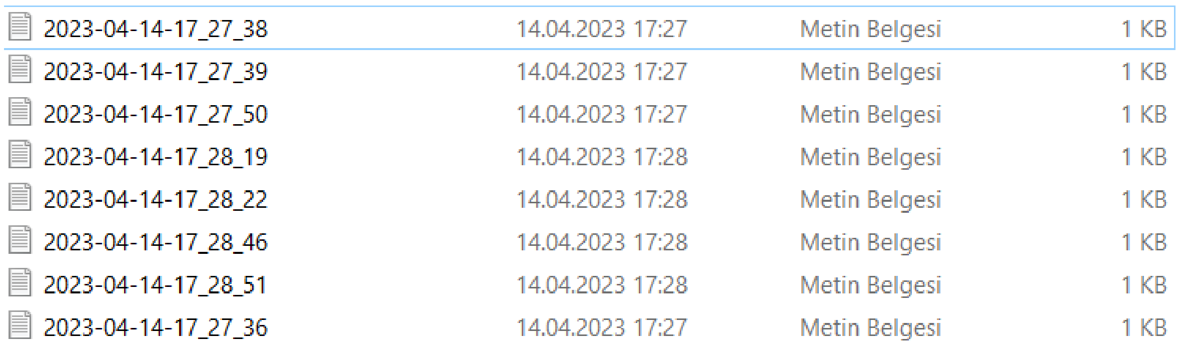
Test – commands 2:



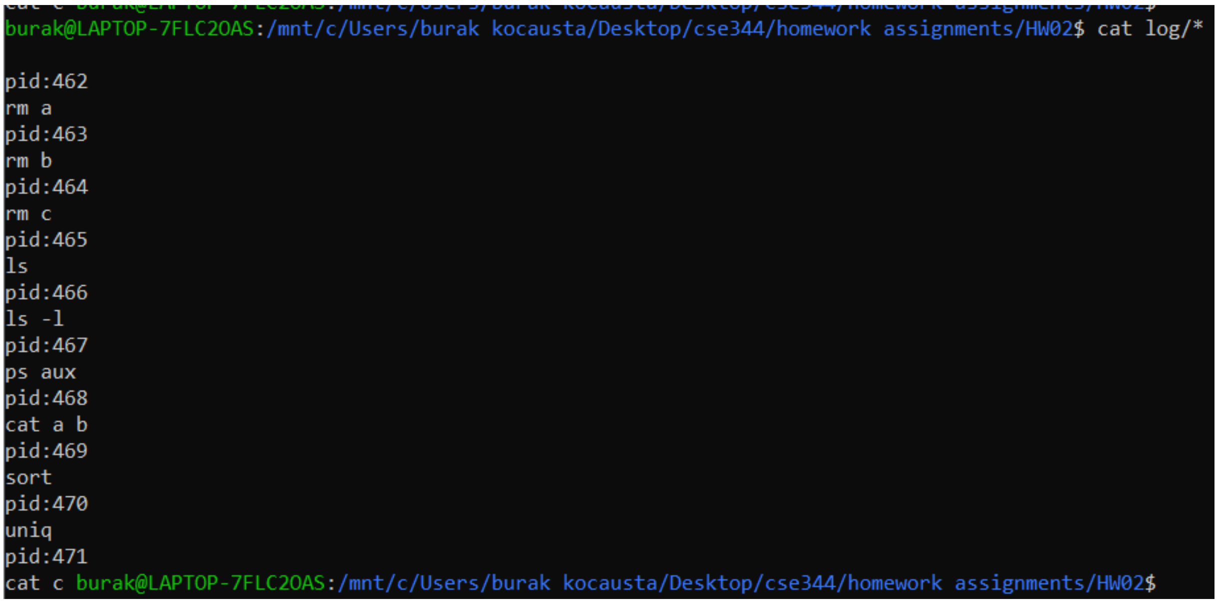
```
burak@LAPTOP-7FLC20AS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$ ./mysh
$ rm a
$ rm b
$ rm c
$ ls
 HW02.zip   commands.md   include   makefile   myshell.c   report.docx   test     '~$report.docx'
 HW2.pdf    file2.txt     log       myshell    report      src           test.c   '~WRL0005.tmp'
$ ls -l > a
$ ps aux > b
$ cat a b | sort | uniq > c
$ cat c
-rwxrwxrwx 1 burak burak      0 Apr 14 17:28 a
-rwxrwxrwx 1 burak burak     54 Apr 14 15:14 file2.txt
-rwxrwxrwx 1 burak burak    108 Apr 13 01:06 test.c
-rwxrwxrwx 1 burak burak    162 Apr 14 13:51 ~$report.docx
-rwxrwxrwx 1 burak burak    536 Apr 14 14:14 makefile
-rwxrwxrwx 1 burak burak   1725 Apr 14 15:51 commands.md
-rwxrwxrwx 1 burak burak   5252 Apr 14 17:13 myshell.c
-rwxrwxrwx 1 burak burak   9303 Apr 14 07:02 HW02.zip
-rwxrwxrwx 1 burak burak  16696 Apr 13 01:06 test
-rwxrwxrwx 1 burak burak  28008 Apr 14 17:15 myshell
-rwxrwxrwx 1 burak burak 125849 Apr  6 15:43 HW2.pdf
-rwxrwxrwx 1 burak burak 217462 Apr 14 17:21 report.docx
-rwxrwxrwx 1 burak burak 581868 Apr 14 13:53 ~WRL0005.tmp
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
burak        9  0.0  0.0  10168  5052 pts/0    Ss   14:15   0:00 -bash
burak      173  0.0  0.0  10168  5072 pts/1    Ss+  16:08   0:00 -bash
burak      461  0.0  0.0   2500  1580 pts/0    S+   17:27   0:00 ./myshell
burak      467  0.0  0.0  10616  3256 pts/0    R+   17:28   0:00 ps aux
drwxrwxrwx 1 burak burak    512 Apr 10 17:09 report
drwxrwxrwx 1 burak burak    512 Apr 14 04:10 include
drwxrwxrwx 1 burak burak    512 Apr 14 04:10 src
drwxrwxrwx 1 burak burak    512 Apr 14 17:28 log
root         1  0.0  0.0    908   536 ?        Sl   14:15   0:00 /init
root         7  0.0  0.0    908    84 ?        Ss   14:15   0:00 /init
root         8  0.0  0.0    908    84 ?        S    14:15   0:00 /init
root       171  0.0  0.0    908    84 ?        Ss   16:08   0:00 /init
root       172  0.0  0.0    908    84 ?        S    16:08   0:00 /init
total 985
$ :q
```

- Redirect ls -l to a then ps aux to b. After that sort unified a and b, and save to c.

Log:

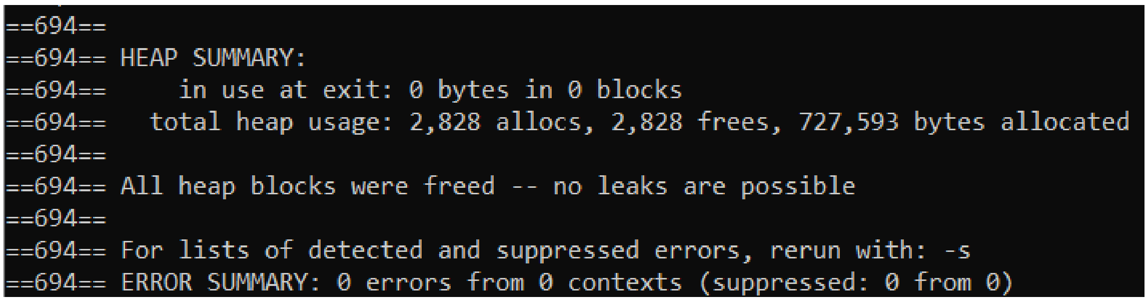


2023-04-14-17_27_38	14.04.2023 17:27	Metin Belgesi	1 KB
2023-04-14-17_27_39	14.04.2023 17:27	Metin Belgesi	1 KB
2023-04-14-17_27_50	14.04.2023 17:27	Metin Belgesi	1 KB
2023-04-14-17_28_19	14.04.2023 17:28	Metin Belgesi	1 KB
2023-04-14-17_28_22	14.04.2023 17:28	Metin Belgesi	1 KB
2023-04-14-17_28_46	14.04.2023 17:28	Metin Belgesi	1 KB
2023-04-14-17_28_51	14.04.2023 17:28	Metin Belgesi	1 KB
2023-04-14-17_27_36	14.04.2023 17:27	Metin Belgesi	1 KB



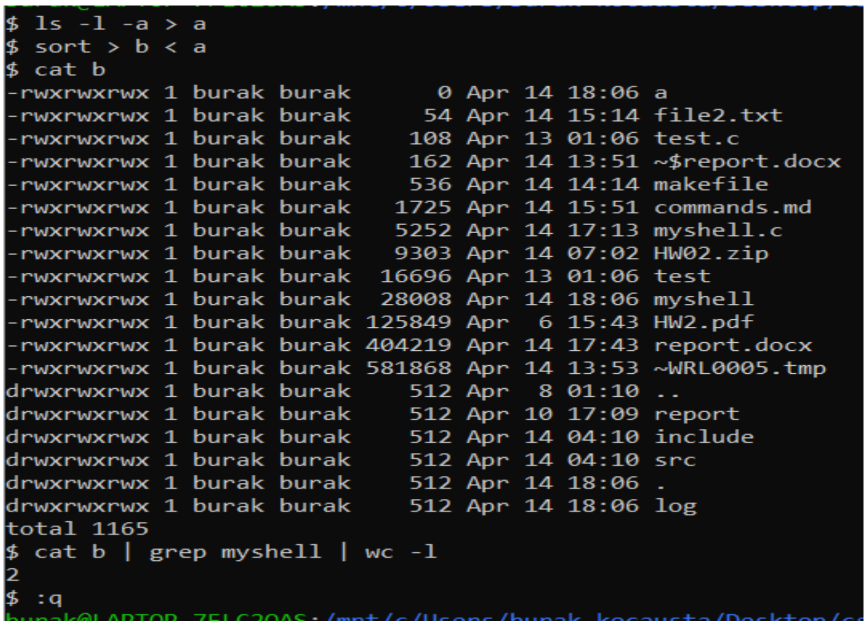
```
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$ cat log/*

pid:462
rm a
pid:463
rm b
pid:464
rm c
pid:465
ls
pid:466
ls -l
pid:467
ps aux
pid:468
cat a b
pid:469
sort
pid:470
uniq
pid:471
cat c burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$
```

Valgrind:



```
==694==
==694== HEAP SUMMARY:
==694==     in use at exit: 0 bytes in 0 blocks
==694==   total heap usage: 2,828 allocs, 2,828 frees, 727,593 bytes allocated
==694==
==694== All heap blocks were freed -- no leaks are possible
==694==
==694== For lists of detected and suppressed errors, rerun with: -s
==694== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

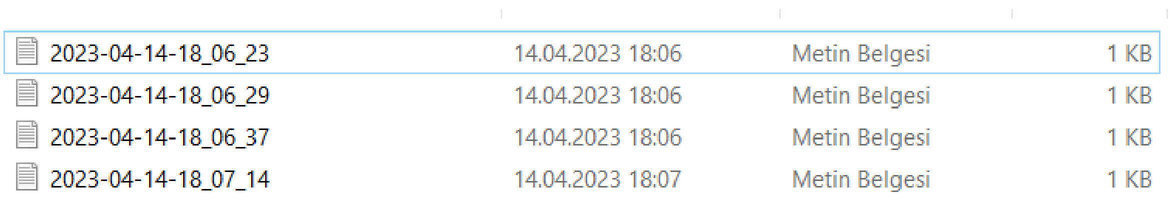
Test – commands 3:



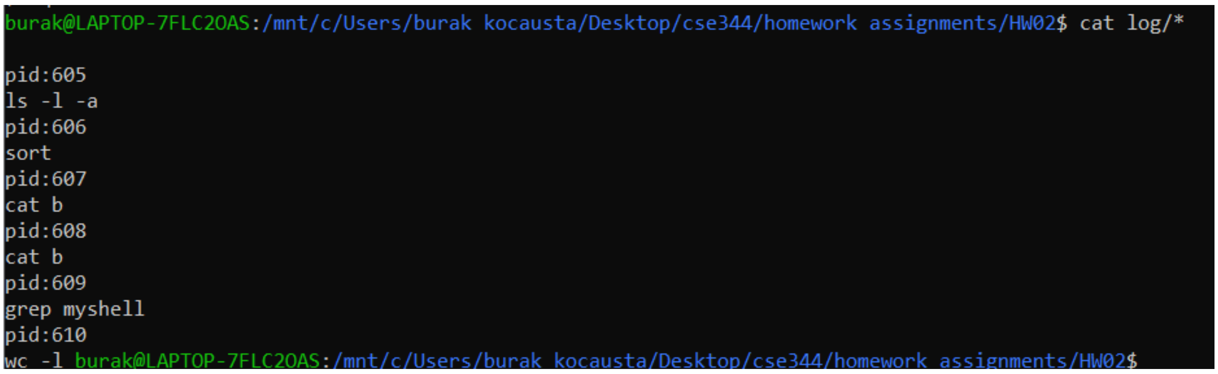
```
$ ls -l -a > a
$ sort > b < a
$ cat b
-rwxrwxrwx 1 burak burak      0 Apr 14 18:06 a
-rwxrwxrwx 1 burak burak     54 Apr 14 15:14 file2.txt
-rwxrwxrwx 1 burak burak    108 Apr 13 01:06 test.c
-rwxrwxrwx 1 burak burak    162 Apr 14 13:51 ~$report.docx
-rwxrwxrwx 1 burak burak    536 Apr 14 14:14 makefile
-rwxrwxrwx 1 burak burak   1725 Apr 14 15:51 commands.md
-rwxrwxrwx 1 burak burak   5252 Apr 14 17:13 myshell.c
-rwxrwxrwx 1 burak burak   9303 Apr 14 07:02 HW02.zip
-rwxrwxrwx 1 burak burak  16696 Apr 13 01:06 test
-rwxrwxrwx 1 burak burak  28008 Apr 14 18:06 myshell
-rwxrwxrwx 1 burak burak 125849 Apr  6 15:43 HW2.pdf
-rwxrwxrwx 1 burak burak 404219 Apr 14 17:43 report.docx
-rwxrwxrwx 1 burak burak 581868 Apr 14 13:53 ~WRL0005.tmp
drwxrwxrwx 1 burak burak    512 Apr  8 01:10 ..
drwxrwxrwx 1 burak burak    512 Apr 10 17:09 report
drwxrwxrwx 1 burak burak    512 Apr 14 04:10 include
drwxrwxrwx 1 burak burak    512 Apr 14 04:10 src
drwxrwxrwx 1 burak burak    512 Apr 14 18:06 .
drwxrwxrwx 1 burak burak    512 Apr 14 18:06 log
total 1165
$ cat b | grep myshell | wc -l
2
$ :q
```

- I tested this for checking what happens if < > used together. It works correctly.

Log:

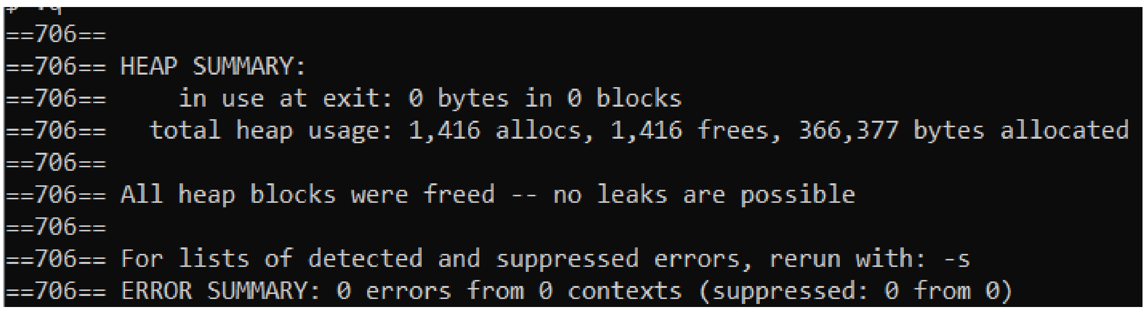


2023-04-14-18_06_23	14.04.2023 18:06	Metin Belgesi	1 KB
2023-04-14-18_06_29	14.04.2023 18:06	Metin Belgesi	1 KB
2023-04-14-18_06_37	14.04.2023 18:06	Metin Belgesi	1 KB
2023-04-14-18_07_14	14.04.2023 18:07	Metin Belgesi	1 KB



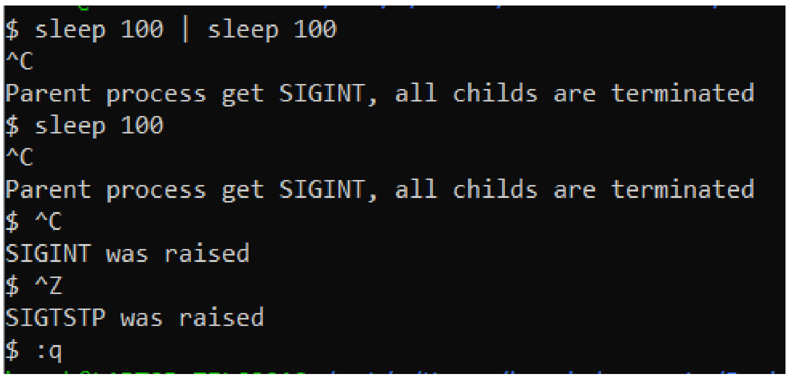
```
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$ cat log/*

pid:605
ls -l -a
pid:606
sort
pid:607
cat b
pid:608
cat b
pid:609
grep myshell
pid:610
wc -l burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$
```

Valgrind:



```
==706==
==706== HEAP SUMMARY:
==706==     in use at exit: 0 bytes in 0 blocks
==706==   total heap usage: 1,416 allocs, 1,416 frees, 366,377 bytes allocated
==706==
==706== All heap blocks were freed -- no leaks are possible
==706==
==706== For lists of detected and suppressed errors, rerun with: -s
==706== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

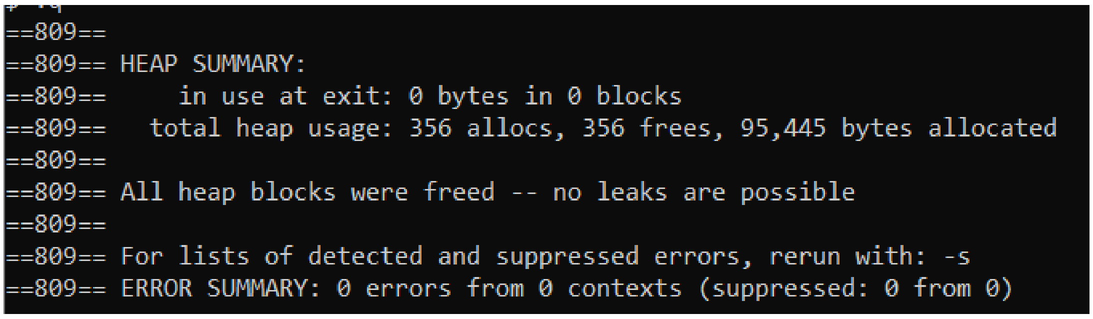
Test – SIGINT, and SIGTSTP from command prompt:



```
$ sleep 100 | sleep 100
^C
Parent process get SIGINT, all childs are terminated
$ sleep 100
^C
Parent process get SIGINT, all childs are terminated
$ ^C
SIGINT was raised
$ ^Z
SIGTSTP was raised
$ :q
```

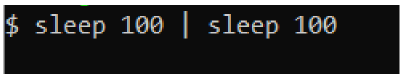
- When the interrupt signal comes with ctrl + c, all childs are terminated. Returns to command prompt.

Valgrind:



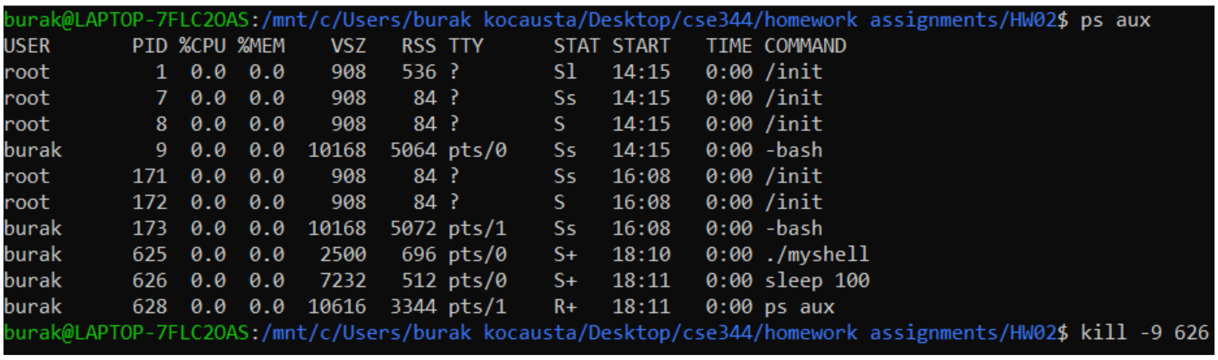
```
==809==
==809== HEAP SUMMARY:
==809==     in use at exit: 0 bytes in 0 blocks
==809==   total heap usage: 356 allocs, 356 frees, 95,445 bytes allocated
==809==
==809== All heap blocks were freed -- no leaks are possible
==809==
==809== For lists of detected and suppressed errors, rerun with: -s
==809== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Test – sending SIGKILL to child:



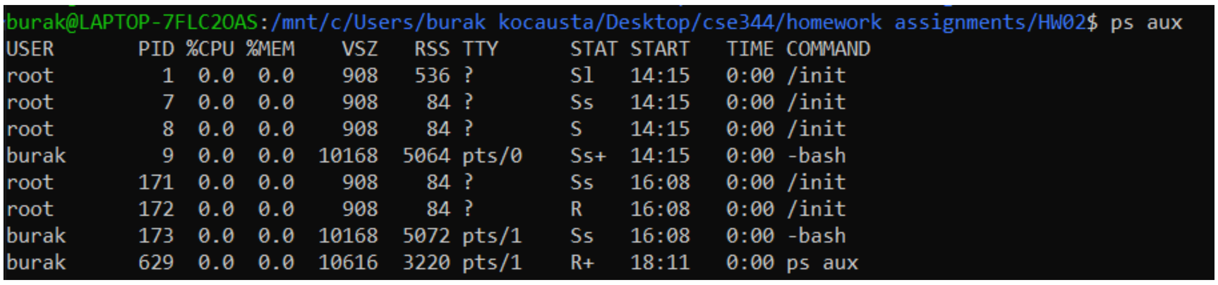
```
$ sleep 100 | sleep 100
```

Before kill

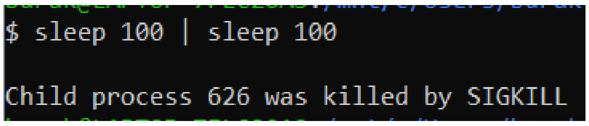


```
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0    908   536 ?        Sl   14:15   0:00 /init
root         7  0.0  0.0    908    84 ?        Ss   14:15   0:00 /init
root         8  0.0  0.0    908    84 ?        S    14:15   0:00 /init
burak        9  0.0  0.0  10168  5064 pts/0    Ss   14:15   0:00 -bash
root       171  0.0  0.0    908    84 ?        Ss   16:08   0:00 /init
root       172  0.0  0.0    908    84 ?        S    16:08   0:00 /init
burak      173  0.0  0.0  10168  5072 pts/1    Ss   16:08   0:00 -bash
burak      625  0.0  0.0   2500   696 pts/0    S+   18:10   0:00 ./myshell
burak      626  0.0  0.0   7232   512 pts/0    S+   18:11   0:00 sleep 100
burak      628  0.0  0.0  10616  3344 pts/1    R+   18:11   0:00 ps aux
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$ kill -9 626
```

After kill



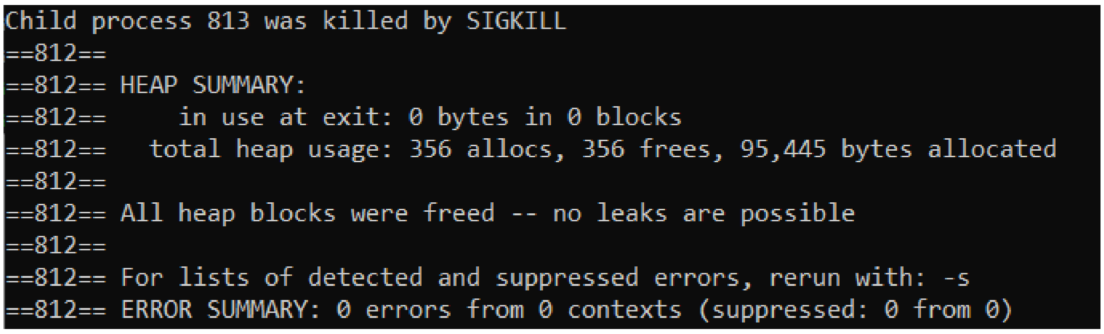
```
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0    908   536 ?        Sl   14:15   0:00 /init
root         7  0.0  0.0    908    84 ?        Ss   14:15   0:00 /init
root         8  0.0  0.0    908    84 ?        S    14:15   0:00 /init
burak        9  0.0  0.0  10168  5064 pts/0    Ss+  14:15   0:00 -bash
root       171  0.0  0.0    908    84 ?        Ss   16:08   0:00 /init
root       172  0.0  0.0    908    84 ?        R    16:08   0:00 /init
burak      173  0.0  0.0  10168  5072 pts/1    Ss   16:08   0:00 -bash
burak      629  0.0  0.0  10616  3220 pts/1    R+   18:11   0:00 ps aux
```



```
$ sleep 100 | sleep 100

Child process 626 was killed by SIGKILL
```

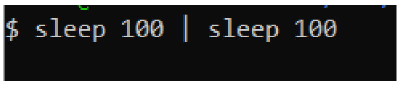
- Sleep 100 commands are executing, I sent kill signal to child process. After child is killed, parent is terminated because of child is being killed.

Valgrind:



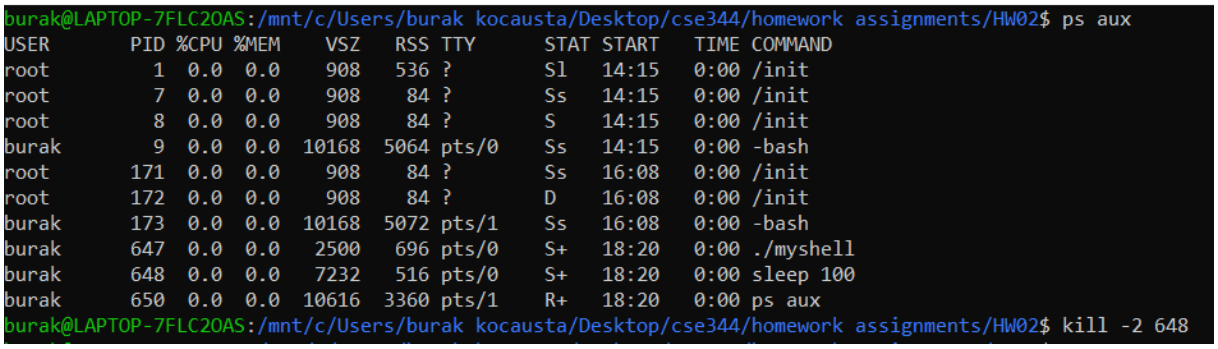
```
Child process 813 was killed by SIGKILL
==812==
==812== HEAP SUMMARY:
==812==     in use at exit: 0 bytes in 0 blocks
==812==   total heap usage: 356 allocs, 356 frees, 95,445 bytes allocated
==812==
==812== All heap blocks were freed -- no leaks are possible
==812==
==812== For lists of detected and suppressed errors, rerun with: -s
==812== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Test – sending SIGINT to child:

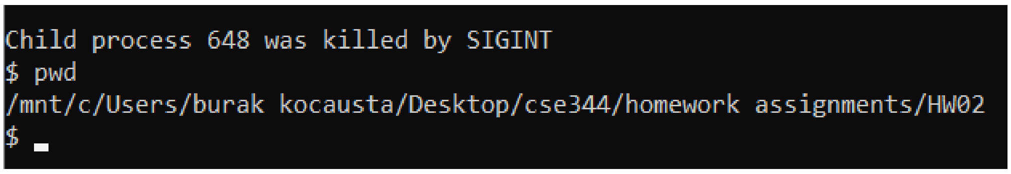


```
$ sleep 100 | sleep 100
```

Before SIGINT

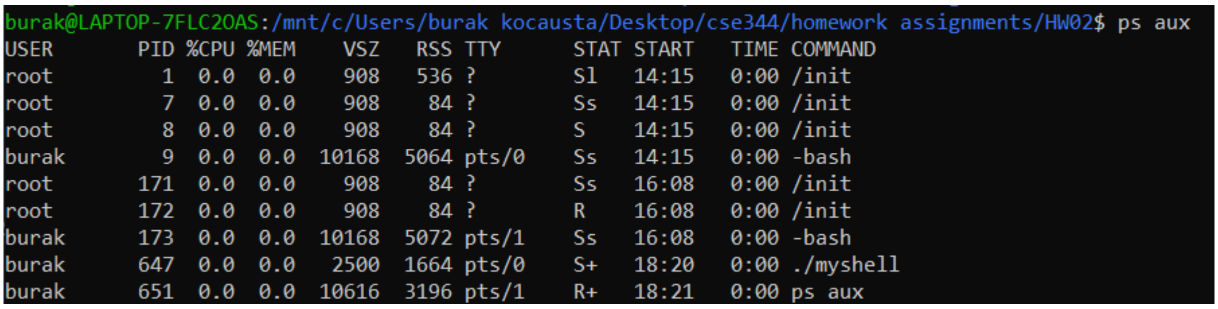


```
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0    908   536 ?        Sl   14:15   0:00 /init
root         7  0.0  0.0    908    84 ?        Ss   14:15   0:00 /init
root         8  0.0  0.0    908    84 ?        S    14:15   0:00 /init
burak        9  0.0  0.0  10168  5064 pts/0    Ss   14:15   0:00 -bash
root       171  0.0  0.0    908    84 ?        Ss   16:08   0:00 /init
root       172  0.0  0.0    908    84 ?        D    16:08   0:00 /init
burak      173  0.0  0.0  10168  5072 pts/1    Ss   16:08   0:00 -bash
burak      647  0.0  0.0   2500   696 pts/0    S+   18:20   0:00 ./myshell
burak      648  0.0  0.0   7232   516 pts/0    S+   18:20   0:00 sleep 100
burak      650  0.0  0.0  10616  3360 pts/1    R+   18:20   0:00 ps aux
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$ kill -2 648
```



```
Child process 648 was killed by SIGINT
$ pwd
/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02
$ 
```

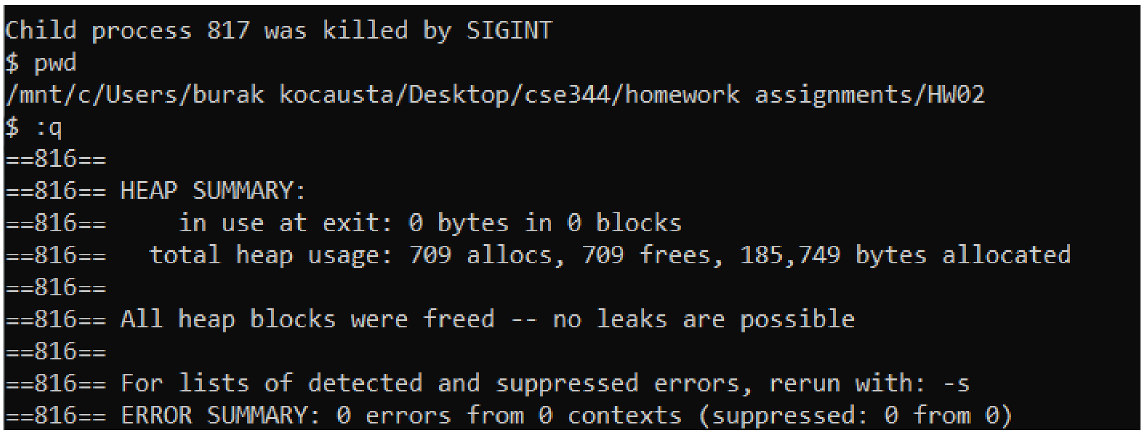
After SIGINT



```
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0    908   536 ?        Sl   14:15   0:00 /init
root         7  0.0  0.0    908    84 ?        Ss   14:15   0:00 /init
root         8  0.0  0.0    908    84 ?        S    14:15   0:00 /init
burak        9  0.0  0.0  10168  5064 pts/0    Ss   14:15   0:00 -bash
root       171  0.0  0.0    908    84 ?        Ss   16:08   0:00 /init
root       172  0.0  0.0    908    84 ?        R    16:08   0:00 /init
burak      173  0.0  0.0  10168  5072 pts/1    Ss   16:08   0:00 -bash
burak      647  0.0  0.0   2500  1664 pts/0    S+   18:20   0:00 ./myshell
burak      651  0.0  0.0  10616  3196 pts/1    R+   18:21   0:00 ps aux
```

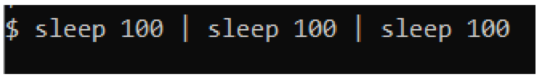
- Sending sigint to child, parent gets it then terminates all child and returns to command prompt.

Valgrind:

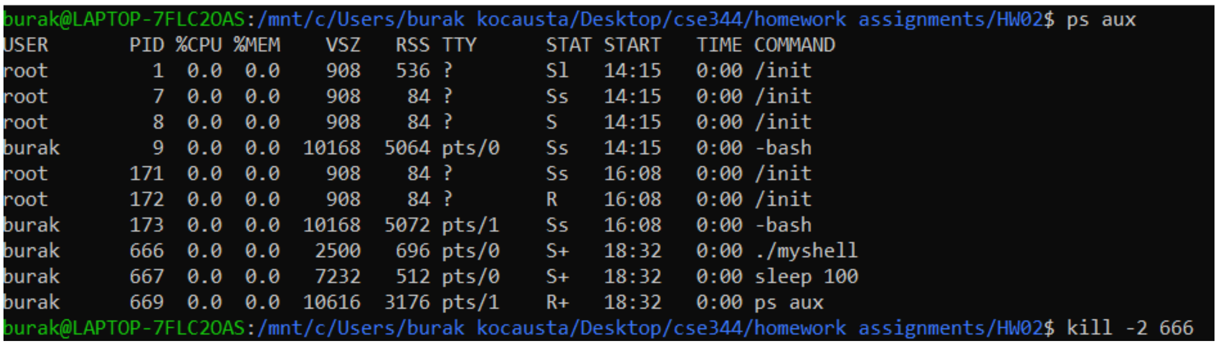


```
Child process 817 was killed by SIGINT
$ pwd
/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02
$ :q
==816==
==816== HEAP SUMMARY:
==816==     in use at exit: 0 bytes in 0 blocks
==816==   total heap usage: 709 allocs, 709 frees, 185,749 bytes allocated
==816==
==816== All heap blocks were freed -- no leaks are possible
==816==
==816== For lists of detected and suppressed errors, rerun with: -s
==816== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

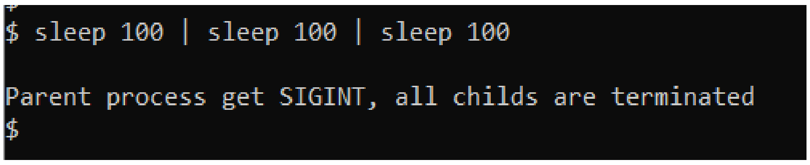
Test – send SIGINT to parent using kill:



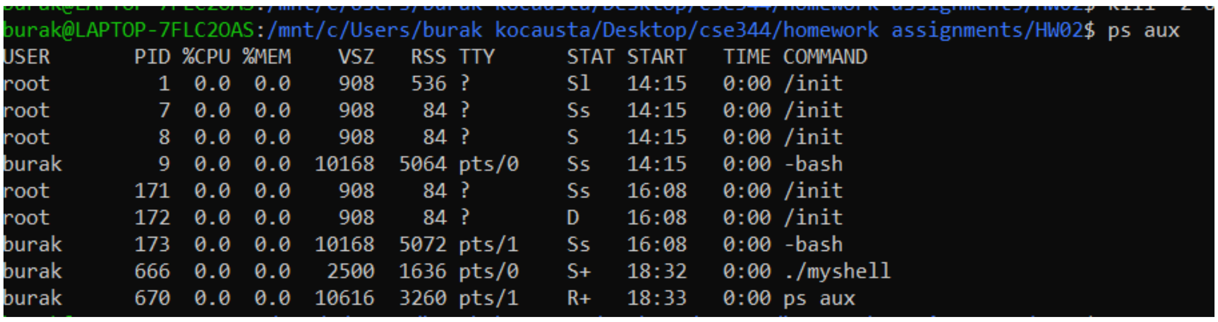
```
$ sleep 100 | sleep 100 | sleep 100
```



```
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0    908   536 ?        Sl   14:15   0:00 /init
root         7  0.0  0.0    908    84 ?        Ss   14:15   0:00 /init
root         8  0.0  0.0    908    84 ?        S    14:15   0:00 /init
burak        9  0.0  0.0  10168  5064 pts/0    Ss   14:15   0:00 -bash
root       171  0.0  0.0    908    84 ?        Ss   16:08   0:00 /init
root       172  0.0  0.0    908    84 ?        R    16:08   0:00 /init
burak      173  0.0  0.0  10168  5072 pts/1    Ss   16:08   0:00 -bash
burak      666  0.0  0.0   2500   696 pts/0    S+   18:32   0:00 ./myshell
burak      667  0.0  0.0   7232   512 pts/0    S+   18:32   0:00 sleep 100
burak      669  0.0  0.0  10616  3176 pts/1    R+   18:32   0:00 ps aux
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$ kill -2 666
```



```
$ sleep 100 | sleep 100 | sleep 100

Parent process get SIGINT, all childs are terminated
$
```



```
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse344/homework assignments/HW02$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0    908   536 ?        Sl   14:15   0:00 /init
root         7  0.0  0.0    908    84 ?        Ss   14:15   0:00 /init
root         8  0.0  0.0    908    84 ?        S    14:15   0:00 /init
burak        9  0.0  0.0  10168  5064 pts/0    Ss   14:15   0:00 -bash
root       171  0.0  0.0    908    84 ?        Ss   16:08   0:00 /init
root       172  0.0  0.0    908    84 ?        D    16:08   0:00 /init
burak      173  0.0  0.0  10168  5072 pts/1    Ss   16:08   0:00 -bash
burak      666  0.0  0.0   2500  1636 pts/0    S+   18:32   0:00 ./myshell
burak      670  0.0  0.0  10616  3260 pts/1    R+   18:33   0:00 ps aux
```

- I sent sigint to myshell process, it received it at terminated the childs. Returned to command prompt.