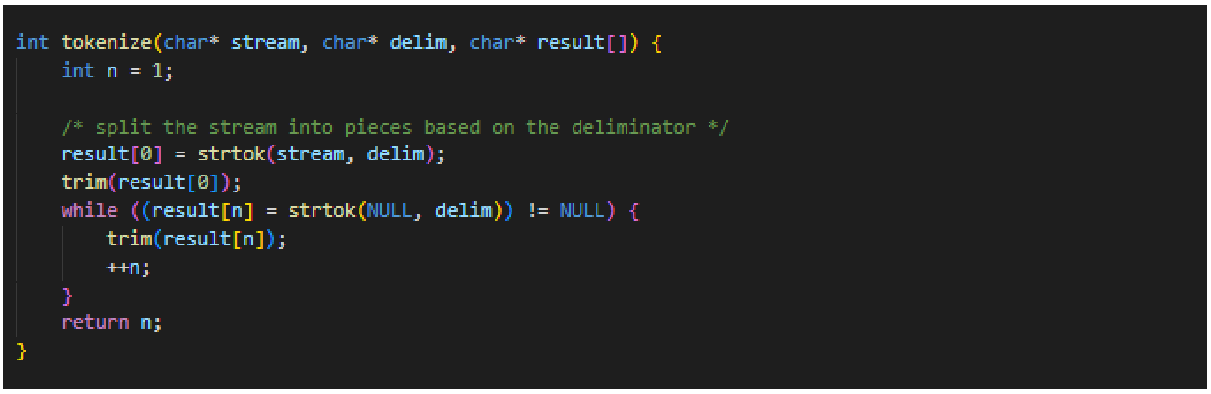
GTU Department of Computer Engineering
CSE 344 - Spring 2023
Homework 2 Report

Emirkan Burak Yılmaz
1901042659

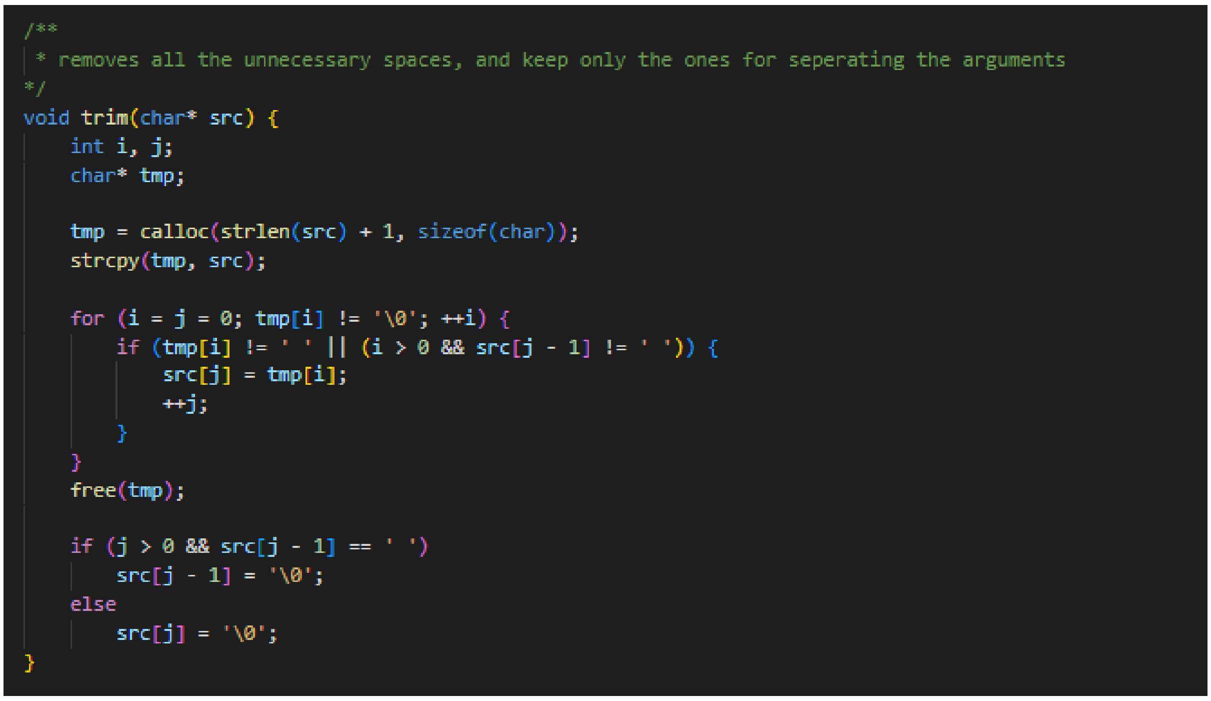
1 Tokenization Strategy

Tokenization process consists of two stages. In the first stage scanned command line is parsed to commands based on the pipe symbol '|', so that commands are separated and can be investigate separately. In the second stage, the commands that are parsed in first step are parsed again based on the space character so that file redirection operations are done.



```c
int tokenize(char* stream, char* delim, char* result[]) {
    int n = 1;

    /* split the stream into pieces based on the deliminator */
    result[0] = strtok(stream, delim);
    trim(result[0]);
    while ((result[n] = strtok(NULL, delim)) != NULL) {
        trim(result[n]);
        ++n;
    }
    return n;
}
```

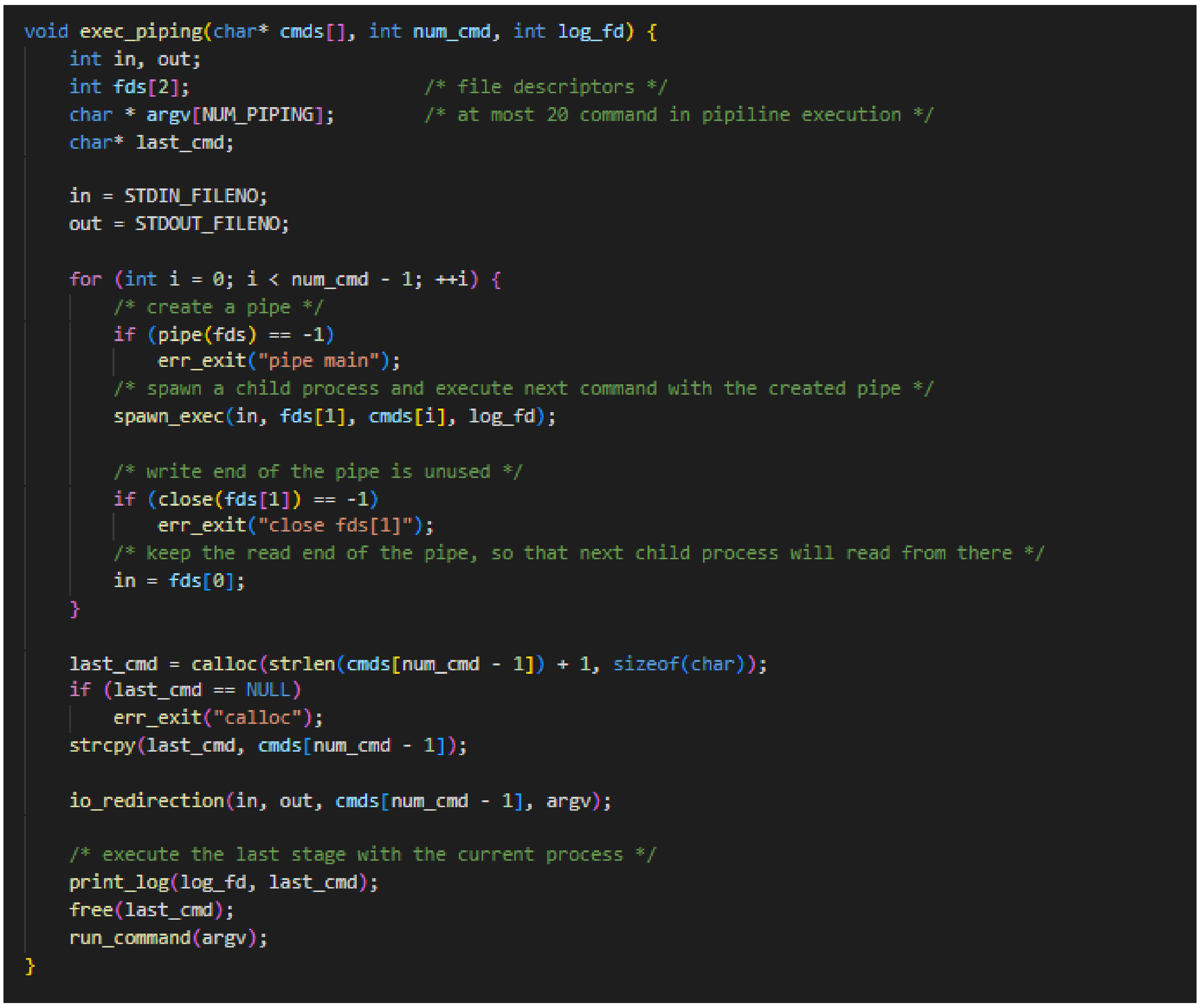
The user input may contain unnecessary spaces that needs to be removed. For that reason tokenize() function uses trim() to remove those unnecessary spaces.



```c
/**
 * removes all the unnecessary spaces, and keep only the ones for seperating the arguments
*/
void trim(char* src) {
    int i, j;
    char* tmp;

    tmp = calloc(strlen(src) + 1, sizeof(char));
    strcpy(tmp, src);

    for (i = j = 0; tmp[i] != '\0'; ++i) {
        if (tmp[i] != ' ' || (i > 0 && src[j - 1] != ' ')) {
            src[j] = tmp[i];
            ++j;
        }
    }
    free(tmp);

    if (j > 0 && src[j - 1] == ' ')
        src[j - 1] = '\0';
    else
        src[j] = '\0';
}
```

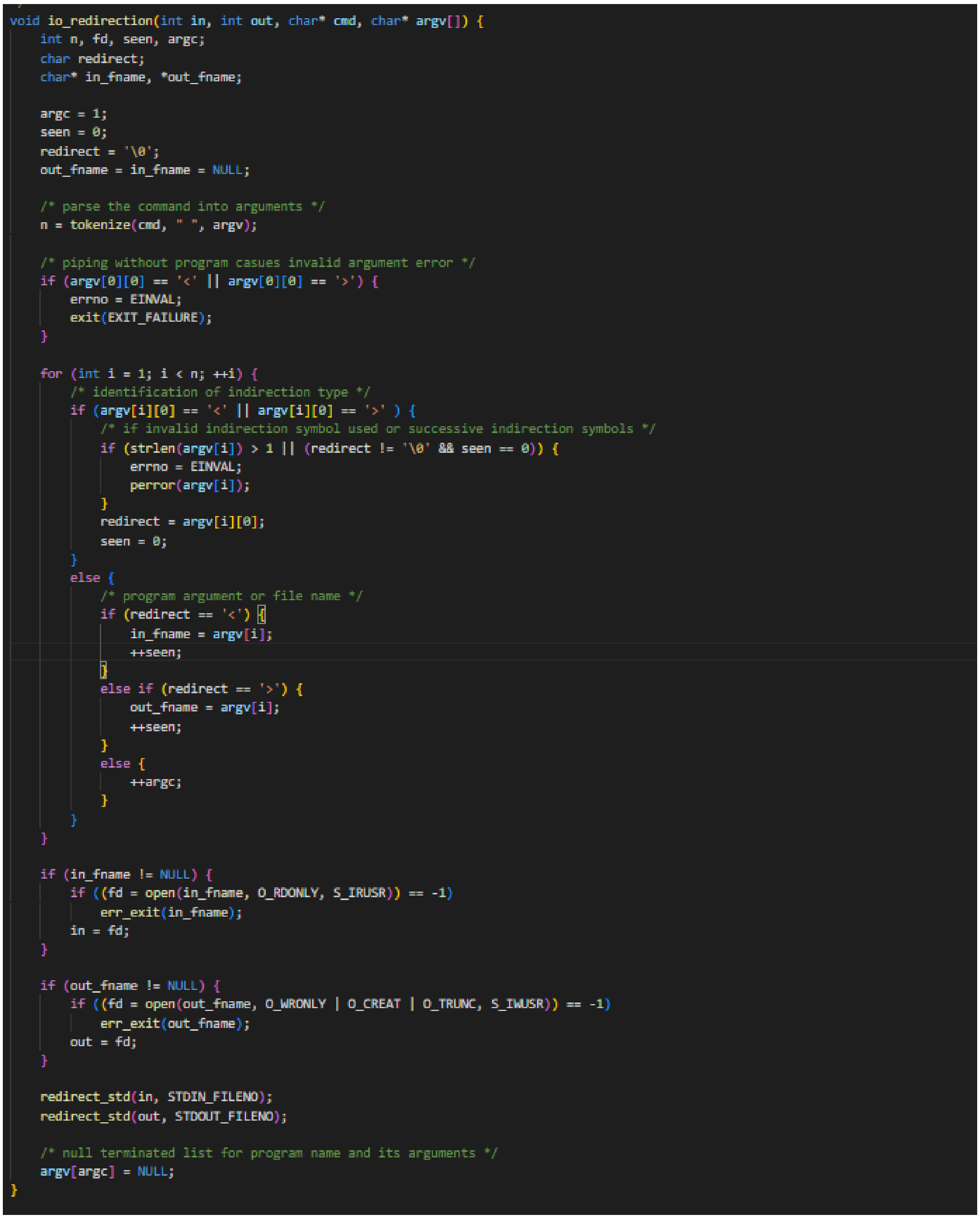
2 Piping

In the piping, multiple commands exist on command line. Since we have the commands separately after the first stage tokenization, we can execute them individually. Before creating pipe structure, understanding the fact that the last command of piping should output to the original process' file descriptor 1 and the first should read from original process file descriptor 0. To achieve this, we need to create child processes in order, carrying along the input side of the previous pipe call.


```c
void exec_piping(char* cmds[], int num_cmd, int log_fd) {
    int in, out;
    int fds[2];                     /* file descriptors */
    char * argv[NUM_PIPING];        /* at most 20 command in pipiline execution */
    char* last_cmd;

    in = STDIN_FILENO;
    out = STDOUT_FILENO;

    for (int i = 0; i < num_cmd - 1; ++i) {
        /* create a pipe */
        if (pipe(fds) == -1)
            err_exit("pipe main");
        /* spawn a child process and execute next command with the created pipe */
        spawn_exec(in, fds[1], cmds[i], log_fd);

        /* write end of the pipe is unused */
        if (close(fds[1]) == -1)
            err_exit("close fds[1]");
        /* keep the read end of the pipe, so that next child process will read from there */
        in = fds[0];
    }

    last_cmd = calloc(strlen(cmds[num_cmd - 1]) + 1, sizeof(char));
    if (last_cmd == NULL)
        err_exit("calloc");
    strcpy(last_cmd, cmds[num_cmd - 1]);

    io_redirection(in, out, cmds[num_cmd - 1], argv);

    /* execute the last stage with the current process */
    print_log(log_fd, last_cmd);
    free(last_cmd);
    run_command(argv);
}
```

exec_piping() function uses the mentinoned fact. It creates a pipe, and a process which uses the write port of previous execution as read port. So that the inputs and outputs of the running processes are conntected to each other.

3 I/O Redirection

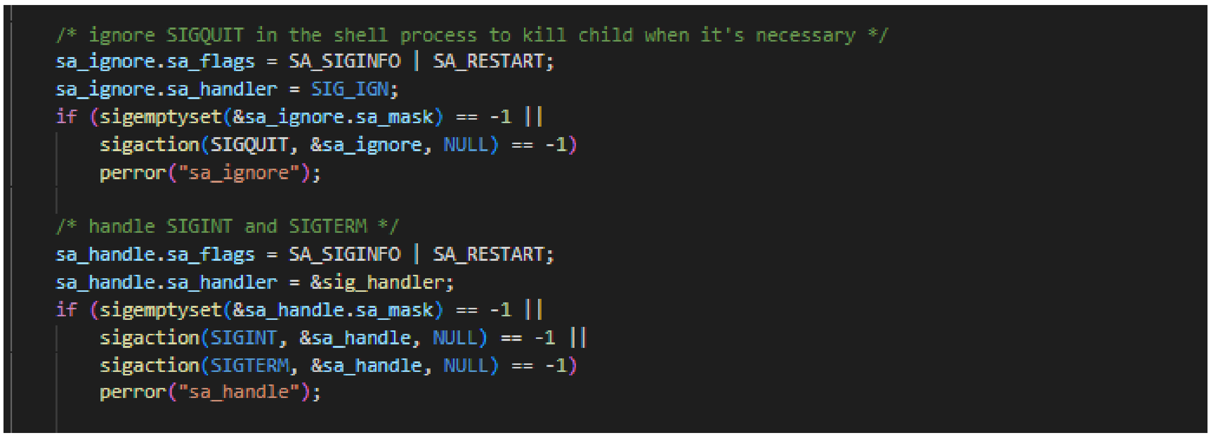


```c
void io_redirection(int in, int out, char* cmd, char* argv[]) {
    int n, fd, seen, argc;
    char redirect;
    char* in_fname, *out_fname;

    argc = 1;
    seen = 0;
    redirect = '\0';
    out_fname = in_fname = NULL;

    /* parse the command into arguments */
    n = tokenize(cmd, " ", argv);

    /* piping without program casues invalid argument error */
    if (argv[0][0] == '<' || argv[0][0] == '>') {
        errno = EINVAL;
        exit(EXIT_FAILURE);
    }

    for (int i = 1; i < n; ++i) {
        /* identification of indirection type */
        if (argv[i][0] == '<' || argv[i][0] == '>' ) {
            /* if invalid indirection symbol used or successive indirection symbols */
            if (strlen(argv[i]) > 1 || (redirect != '\0' && seen == 0)) {
                errno = EINVAL;
                perror(argv[i]);
            }
            redirect = argv[i][0];
            seen = 0;
        }
        else {
            /* program argument or file name */
            if (redirect == '<') {
                in_fname = argv[i];
                ++seen;
            }
            else if (redirect == '>') {
                out_fname = argv[i];
                ++seen;
            }
            else {
                ++argc;
            }
        }
    }

    if (in_fname != NULL) {
        if ((fd = open(in_fname, O_RDONLY, S_IRUSR)) == -1)
            err_exit(in_fname);
        in = fd;
    }

    if (out_fname != NULL) {
        if ((fd = open(out_fname, O_WRONLY | O_CREAT | O_TRUNC, S_IWUSR)) == -1)
            err_exit(out_fname);
        out = fd;
    }

    redirect_std(in, STDIN_FILENO);
    redirect_std(out, STDOUT_FILENO);

    /* null terminated list for program name and its arguments */
    argv[argc] = NULL;
}
```

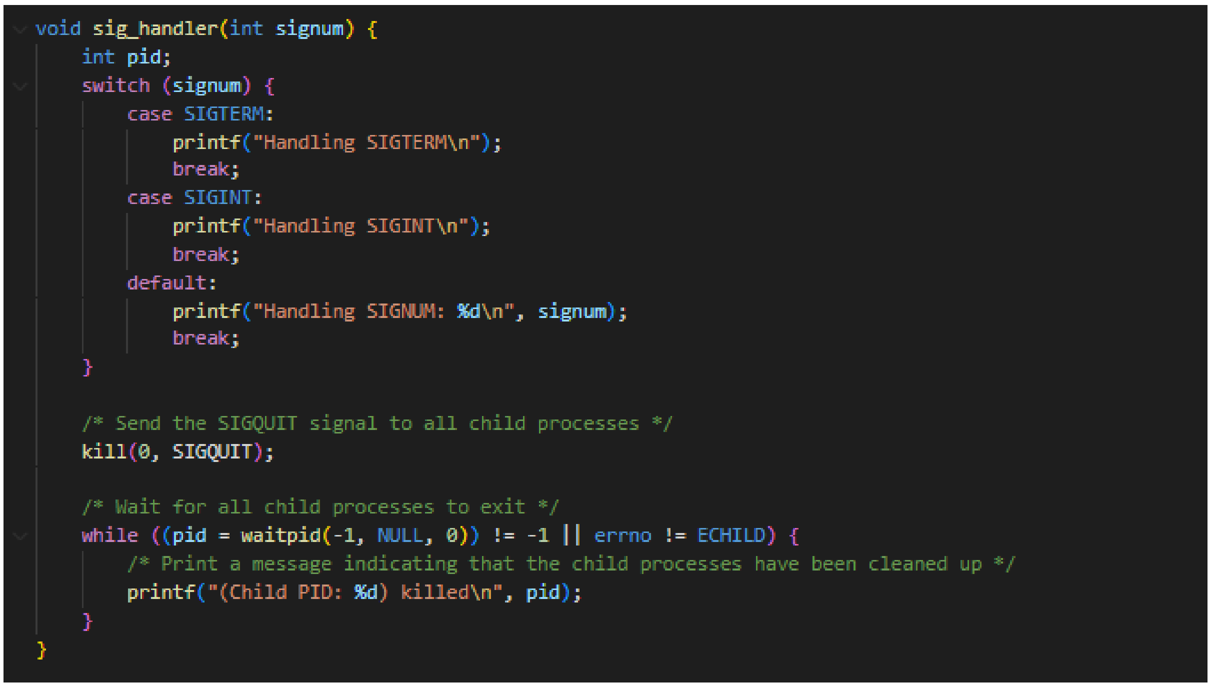
I/O redirection is done by opening the target file in proper format according to indirection type and duplicating it as standard input or output file descriptor.

4 Signal Handling

Signal handler function sig_handler() is written to handle the signals SIGINT and SIGTERM. Before the handler execution we set the shell process to ignore SIGQUIT signal. In the handler function, SIGQUIT signal is sent to all the processes in the current group. Then all the child processes are killed. Since the parent process was set before to ignore SIGQUIT signal it can continue to its execution by prompting user for next command. For the signal SIGKILL, we don't have anything to do except terminating all the processes, since it cannot be catch or ignored.



```c
/* ignore SIGQUIT in the shell process to kill child when it's necessary */
sa_ignore.sa_flags = SA_SIGINFO | SA_RESTART;
sa_ignore.sa_handler = SIG_IGN;
if (sigemptyset(&sa_ignore.sa_mask) == -1 ||
    sigaction(SIGQUIT, &sa_ignore, NULL) == -1)
    perror("sa_ignore");

/* handle SIGINT and SIGTERM */
sa_handle.sa_flags = SA_SIGINFO | SA_RESTART;
sa_handle.sa_handler = &sig_handler;
if (sigemptyset(&sa_handle.sa_mask) == -1 ||
    sigaction(SIGINT, &sa_handle, NULL) == -1 ||
    sigaction(SIGTERM, &sa_handle, NULL) == -1)
    perror("sa_handle");
```

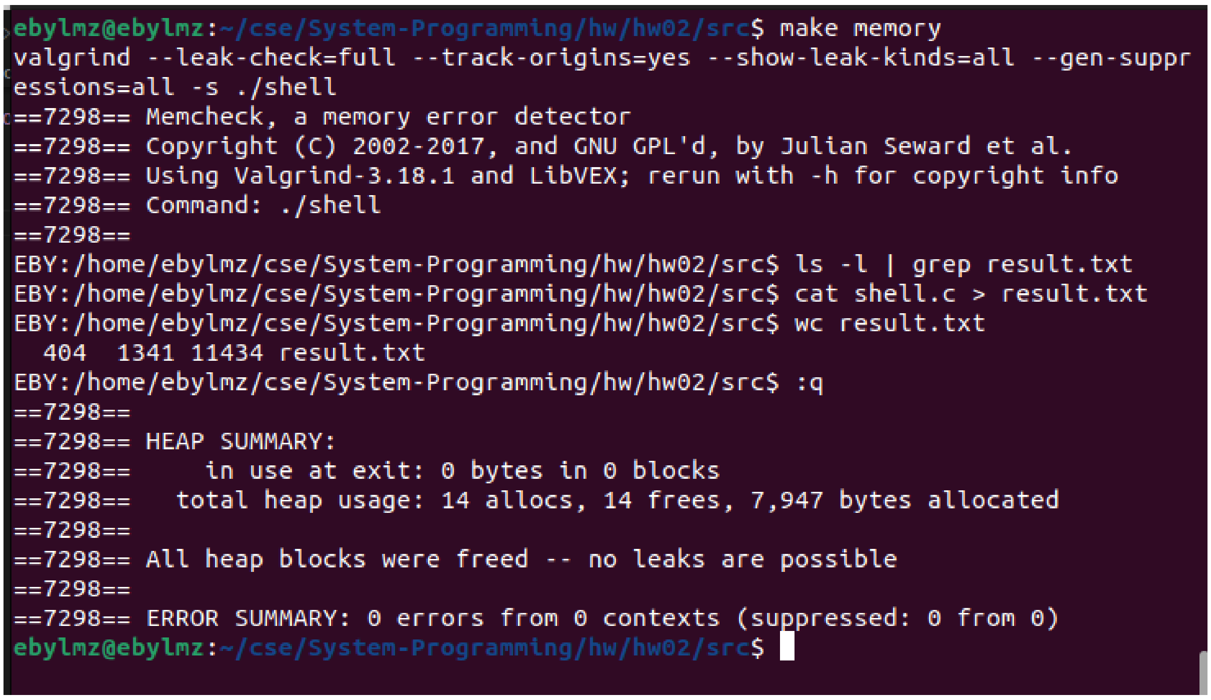


```c
void sig_handler(int signum) {
    int pid;
    switch (signum) {
        case SIGTERM:
            printf("Handling SIGTERM\n");
            break;
        case SIGINT:
            printf("Handling SIGINT\n");
            break;
        default:
            printf("Handling SIGNUM: %d\n", signum);
            break;
    }

    /* Send the SIGQUIT signal to all child processes */
    kill(0, SIGQUIT);

    /* Wait for all child processes to exit */
    while ((pid = waitpid(-1, NULL, 0)) != -1 || errno != ECHILD) {
        /* Print a message indicating that the child processes have been cleaned up */
        printf("(Child PID: %d) killed\n", pid);
    }
}
```

5 Testing

5.1 Memory Leak

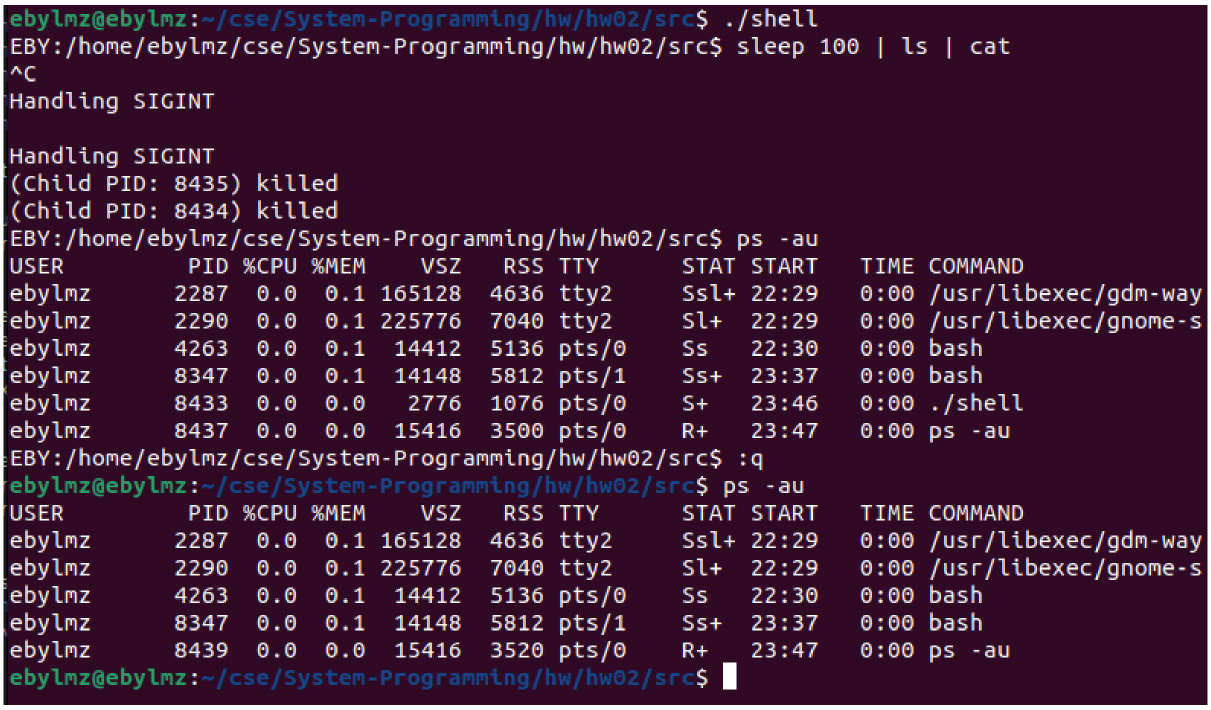
Memory leaks are checked with Valgrind and as it can be seen below, there is no memory leak.



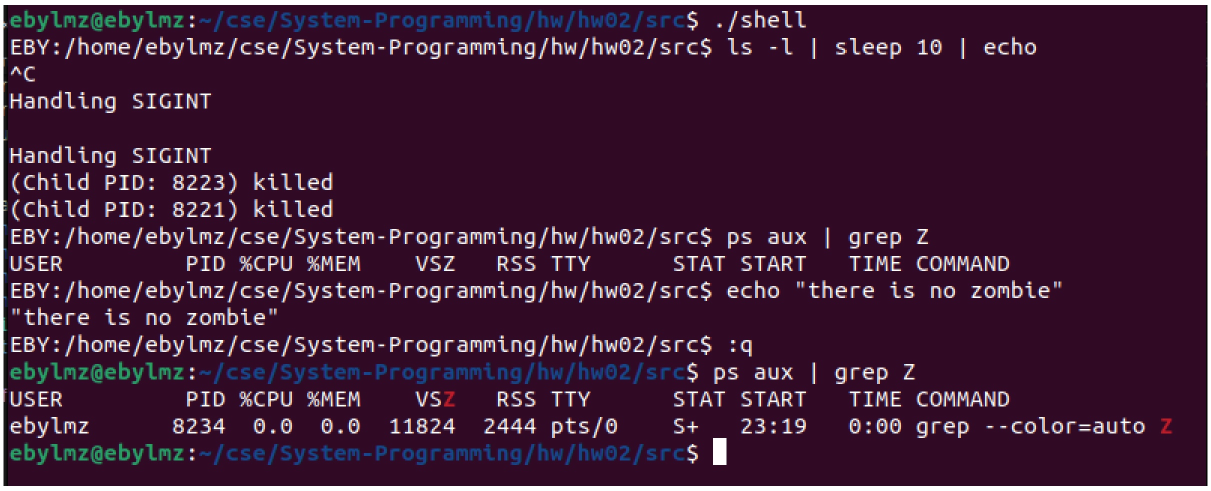
```
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ make memory
valgrind --leak-check=full --track-origins=yes --show-leak-kinds=all --gen-suppr
essions=all -s ./shell
==7298== Memcheck, a memory error detector
==7298== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7298== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==7298== Command: ./shell
==7298==
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ ls -l | grep result.txt
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ cat shell.c > result.txt
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ wc result.txt
  404  1341 11434 result.txt
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ :q
==7298==
==7298== HEAP SUMMARY:
==7298==     in use at exit: 0 bytes in 0 blocks
==7298==   total heap usage: 14 allocs, 14 frees, 7,947 bytes allocated
==7298==
==7298== All heap blocks were freed -- no leaks are possible
==7298==
==7298== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$
```

5.2 Cleaning Up Child Processes & Zombie Processes

All the parent processes wait their children to terminate, so there should be no zombie process. We can use ps aux | grep Z command to check the zombie process.



```
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ./shell
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ sleep 100 | ls | cat
^C
Handling SIGINT

Handling SIGINT
(Child PID: 8435) killed
(Child PID: 8434) killed
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ ps -au
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ebylmz      2287  0.0  0.1 165128  4636 tty2     Ssl+ 22:29   0:00 /usr/libexec/gdm-way
ebylmz      2290  0.0  0.1 225776  7040 tty2     Sl+  22:29   0:00 /usr/libexec/gnome-s
ebylmz      4263  0.0  0.1  14412  5136 pts/0    Ss   22:30   0:00 bash
ebylmz      8347  0.0  0.1  14148  5812 pts/1    Ss+  23:37   0:00 bash
ebylmz      8433  0.0  0.0   2776  1076 pts/0    S+   23:46   0:00 ./shell
ebylmz      8437  0.0  0.0  15416  3500 pts/0    R+   23:47   0:00 ps -au
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ :q
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ps -au
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ebylmz      2287  0.0  0.1 165128  4636 tty2     Ssl+ 22:29   0:00 /usr/libexec/gdm-way
ebylmz      2290  0.0  0.1 225776  7040 tty2     Sl+  22:29   0:00 /usr/libexec/gnome-s
ebylmz      4263  0.0  0.1  14412  5136 pts/0    Ss   22:30   0:00 bash
ebylmz      8347  0.0  0.1  14148  5812 pts/1    Ss+  23:37   0:00 bash
ebylmz      8439  0.0  0.0  15416  3520 pts/0    R+   23:47   0:00 ps -au
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$
```

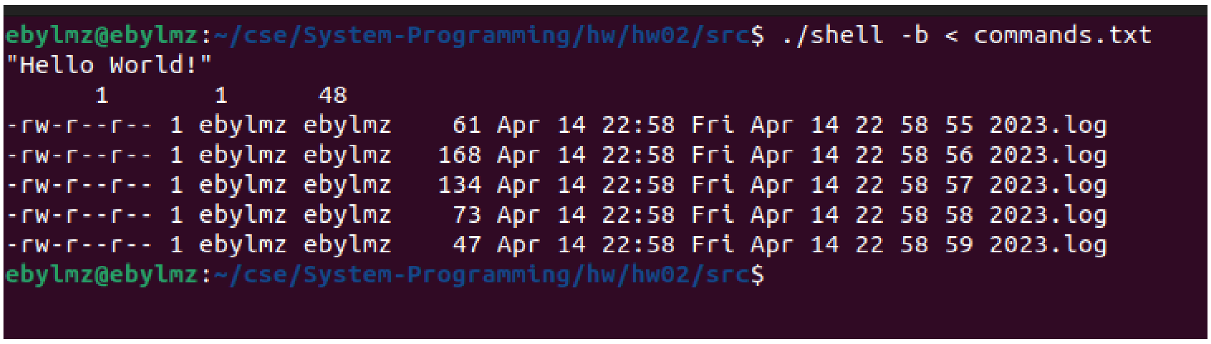


```
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ./shell
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ ls -l | sleep 10 | echo
^C
Handling SIGINT

Handling SIGINT
(Child PID: 8223) killed
(Child PID: 8221) killed
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ ps aux | grep Z
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ echo "there is no zombie"
"there is no zombie"
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ :q
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ps aux | grep Z
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ebylmz       8234  0.0  0.0  11824  2444 pts/0    S+   23:19   0:00 grep --color=auto Z
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$
```

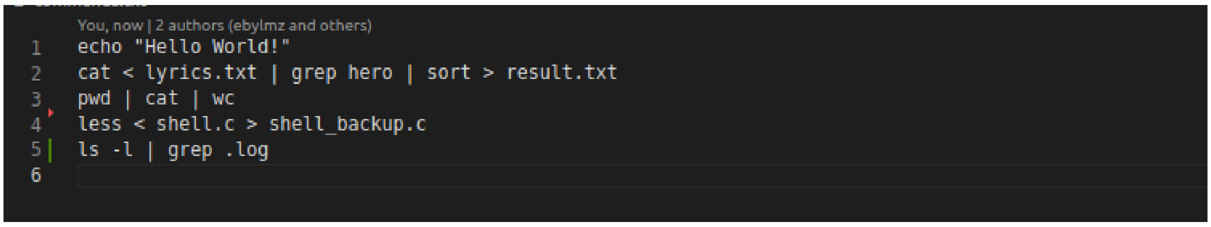
5.3 Running

5.3.1 Bash mode

In the bash mode shell does not prompt with working directory, so that results of the commands can be seen better. This mode is useful when indirection a file which contains the commands.



```
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ./shell -b < commands.txt
"Hello World!"
      1       1      48
-rw-r--r-- 1 ebylmz ebylmz    61 Apr 14 22:58 Fri Apr 14 22 58 55 2023.log
-rw-r--r-- 1 ebylmz ebylmz   168 Apr 14 22:58 Fri Apr 14 22 58 56 2023.log
-rw-r--r-- 1 ebylmz ebylmz   134 Apr 14 22:58 Fri Apr 14 22 58 57 2023.log
-rw-r--r-- 1 ebylmz ebylmz    73 Apr 14 22:58 Fri Apr 14 22 58 58 2023.log
-rw-r--r-- 1 ebylmz ebylmz    47 Apr 14 22:58 Fri Apr 14 22 58 59 2023.log
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$
```

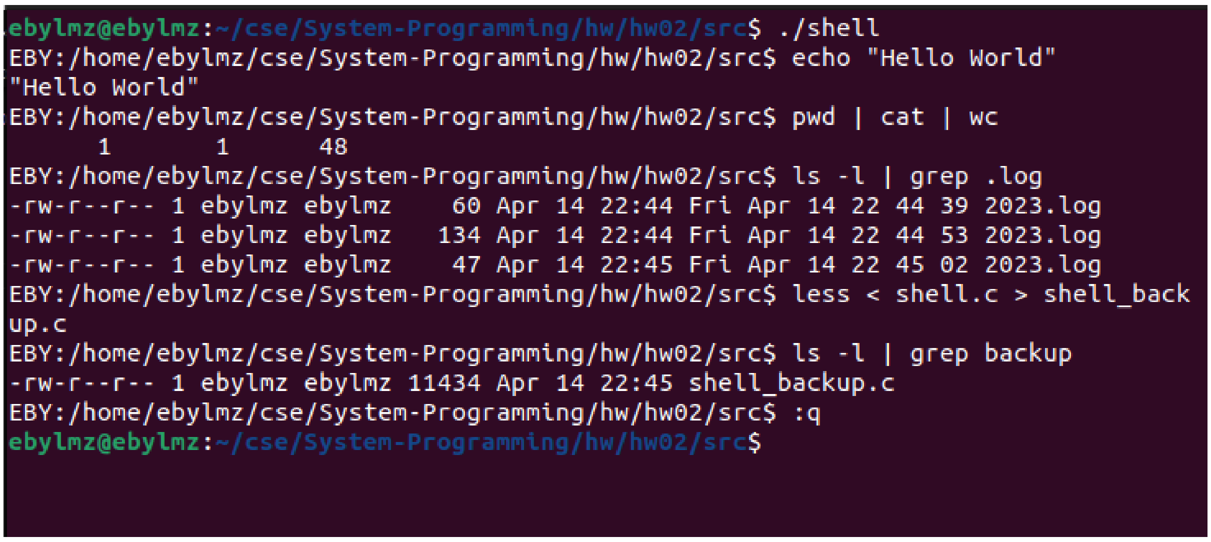


You, now | 2 authors (ebylmz and others)

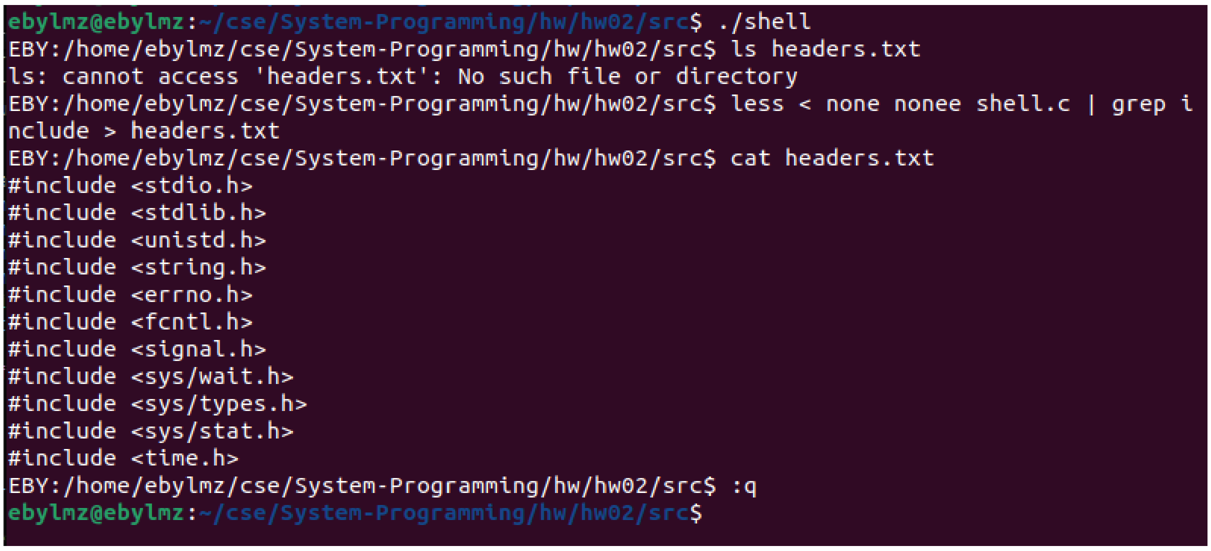
```
echo "Hello World!"
cat < lyrics.txt | grep hero | sort > result.txt
pwd | cat | wc
less < shell.c > shell_backup.c
ls -l | grep .log
```

5.3.2 Piping & I/O redirection

Up to 20 pipe execution can be done.

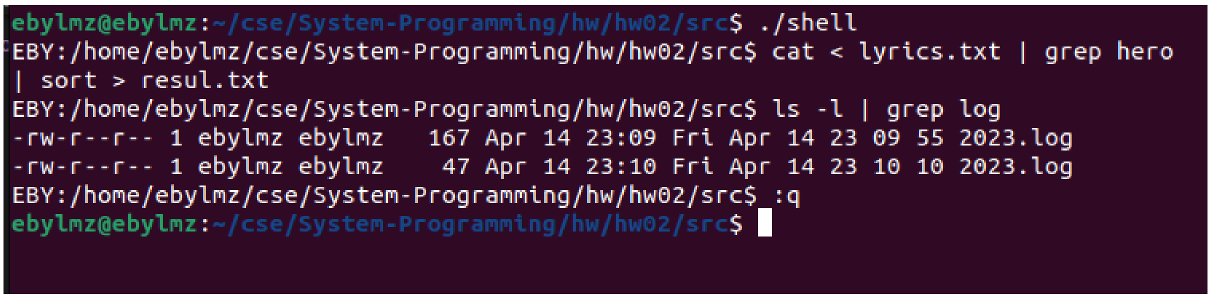

```
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ./shell
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ echo "Hello World"
"Hello World"
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ pwd | cat | wc
      1       1      48
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ ls -l | grep .log
-rw-r--r-- 1 ebylmz ebylmz    60 Apr 14 22:44 Fri Apr 14 22 44 39 2023.log
-rw-r--r-- 1 ebylmz ebylmz   134 Apr 14 22:44 Fri Apr 14 22 44 53 2023.log
-rw-r--r-- 1 ebylmz ebylmz    47 Apr 14 22:45 Fri Apr 14 22 45 02 2023.log
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ less < shell.c > shell_back
up.c
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ ls -l | grep backup
-rw-r--r-- 1 ebylmz ebylmz 11434 Apr 14 22:45 shell_backup.c
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ :q
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$
```

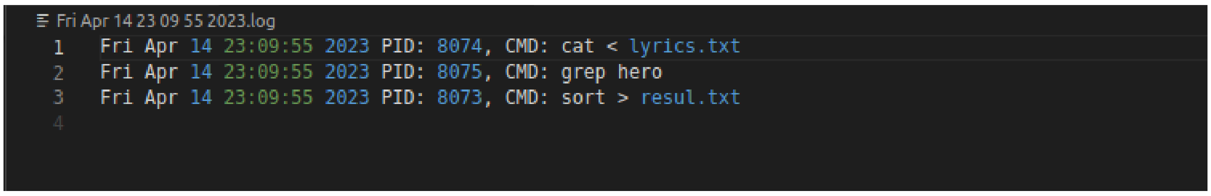
In I/O redirection the last argument after the redirection symbol is considered as true argument. So the other arguments are just ignored.

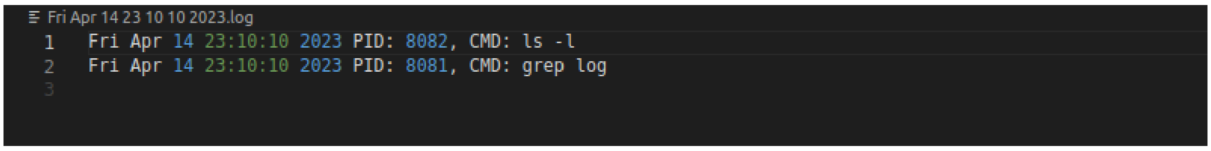

```
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ./shell
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ ls headers.txt
ls: cannot access 'headers.txt': No such file or directory
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ less < none nonee shell.c | grep i
nclude > headers.txt
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ cat headers.txt
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ :q
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$
```

5.3.2.1 Log file

After each command line executed, a log file named with execution time is created.


```
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ./shell
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ cat < lyrics.txt | grep hero
| sort > resul.txt
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ ls -l | grep log
-rw-r--r-- 1 ebylmz ebylmz   167 Apr 14 23:09 Fri Apr 14 23 09 55 2023.log
-rw-r--r-- 1 ebylmz ebylmz    47 Apr 14 23:10 Fri Apr 14 23 10 10 2023.log
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ :q
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$
```

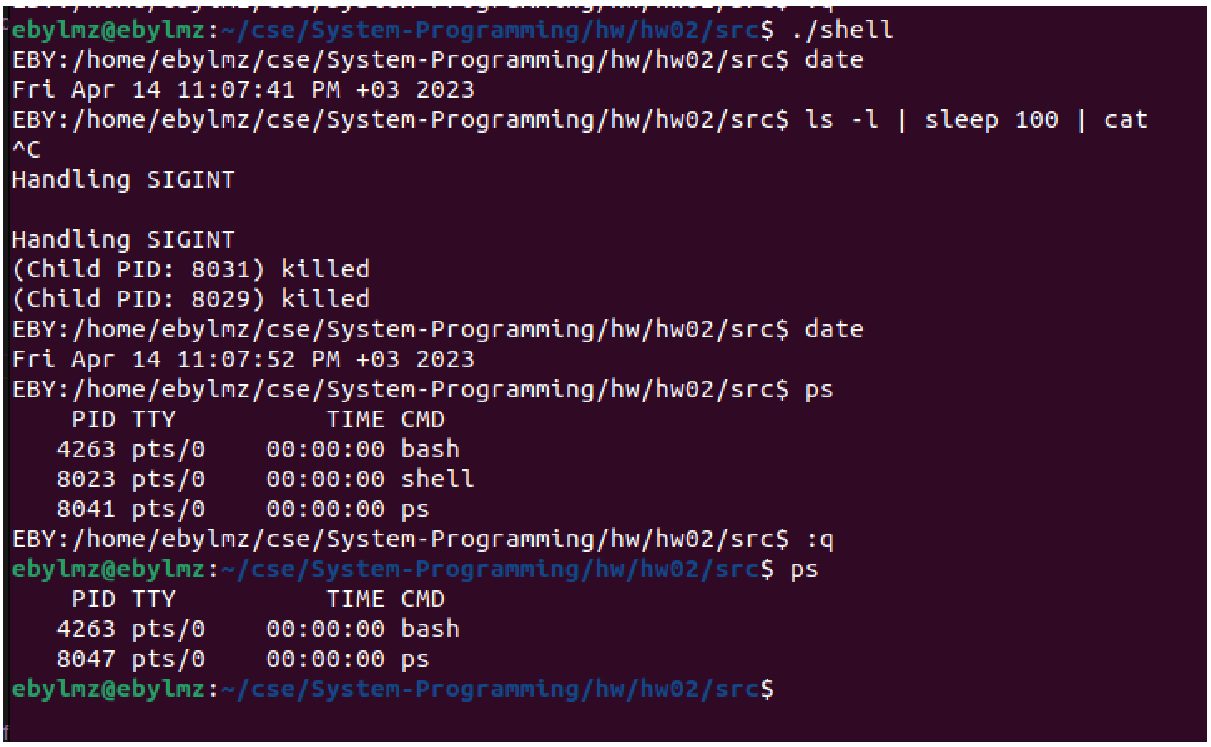

```
Fri Apr 14 23 09 55 2023.log
Fri Apr 14 23:09:55 2023 PID: 8074, CMD: cat < lyrics.txt
Fri Apr 14 23:09:55 2023 PID: 8075, CMD: grep hero
Fri Apr 14 23:09:55 2023 PID: 8073, CMD: sort > resul.txt
```



```
Fri Apr 14 23 10 10 2023.log
Fri Apr 14 23:10:10 2023 PID: 8082, CMD: ls -l
Fri Apr 14 23:10:10 2023 PID: 8081, CMD: grep log
```

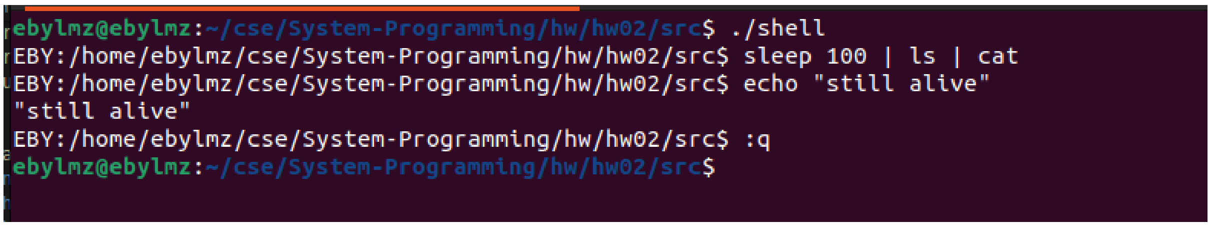
5.3.3 Signal Handling

SIGINT can be checked by pressing CTR-C. When the SIGINT occurs, handler kills the child and parent process continues its execution by prompting for new command.

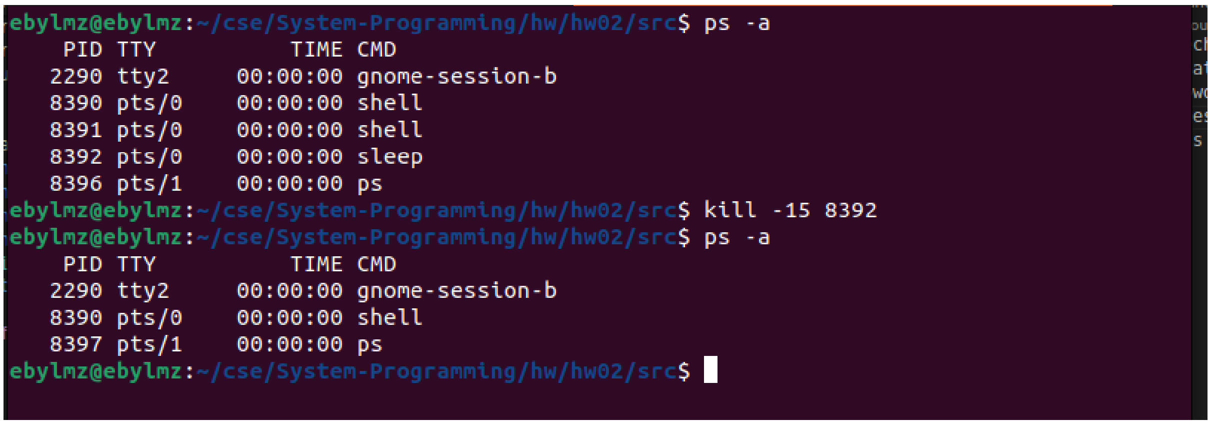


```
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ./shell
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ date
Fri Apr 14 11:07:41 PM +03 2023
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ ls -l | sleep 100 | cat
^C
Handling SIGINT

Handling SIGINT
(Child PID: 8031) killed
(Child PID: 8029) killed
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ date
Fri Apr 14 11:07:52 PM +03 2023
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ ps
    PID TTY          TIME CMD
   4263 pts/0    00:00:00 bash
   8023 pts/0    00:00:00 shell
   8041 pts/0    00:00:00 ps
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ :q
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ps
    PID TTY          TIME CMD
   4263 pts/0    00:00:00 bash
   8047 pts/0    00:00:00 ps
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$
```

SIGTERM signal which can be sent through another terminal during the execution of the shell.



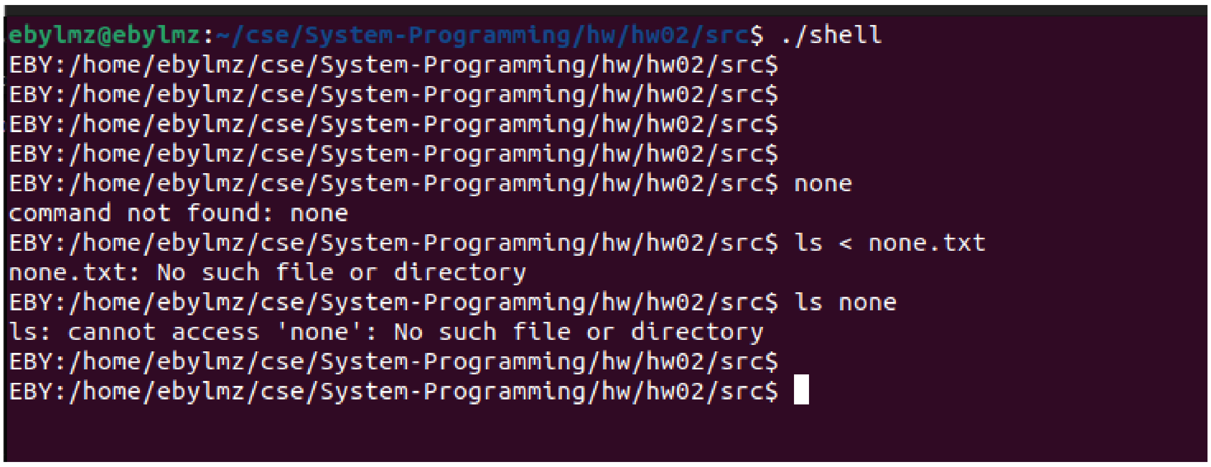
```
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ./shell
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ sleep 100 | ls | cat
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ echo "still alive"
"still alive"
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ :q
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$
```



```
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ps -a
    PID TTY          TIME CMD
   2290 tty2     00:00:00 gnome-session-b
   8390 pts/0    00:00:00 shell
   8391 pts/0    00:00:00 shell
   8392 pts/0    00:00:00 sleep
   8396 pts/1    00:00:00 ps
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ kill -15 8392
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ps -a
    PID TTY          TIME CMD
   2290 tty2     00:00:00 gnome-session-b
   8390 pts/0    00:00:00 shell
   8397 pts/1    00:00:00 ps
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$
```

5.3.4 Error Handling

Standard error outputs are displayed on the shell.



```
ebylmz@ebylmz:~/cse/System-Programming/hw/hw02/src$ ./shell
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ none
command not found: none
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ ls < none.txt
none.txt: No such file or directory
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$ ls none
ls: cannot access 'none': No such file or directory
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$
EBY:/home/ebylmz/cse/System-Programming/hw/hw02/src$
```