CSE 344 System Programming

HW2 Report

1901042673 – Ersel Celal Eren

In this homework, it is asked to implement a terminal emulator in C. To do this task, I followed this path : I get input from user like

“cat file1.txt file2.txt | sort | uniq > merged_files.txt” and parse it. Then I created child process with fork() syscall for each command (in this example cat,sort and uniq). To handle multiple commands in one line, I used pipes. I created (n-1) pipe, n is the number of commands which is 3 here”. Then I set a communication between commands.

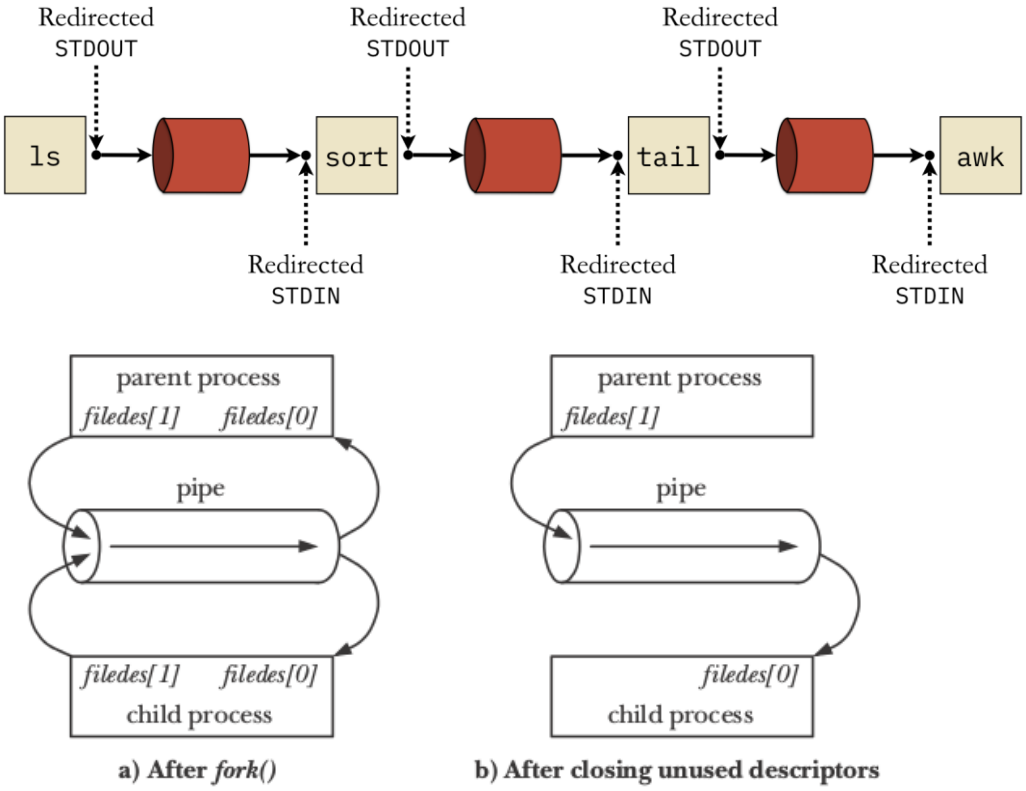

Figure 44-3: Setting up a pipe to transfer data from a parent to a child

So, these are images that I found when I start to develop program. They give me an idea of how to start.

First, I define signal handlers. With sigaction() syscall, SIGINT and SIGTERM signals are ignored during the program. Also I defined another signal handler to handle termination of child process if program gets SIGKILL.

```c
int main() {

    //ignore signals
    struct sigaction act;
    act.sa_handler = SIG_IGN;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;

    //handle sigchld
    struct sigaction sa;
    sa.sa_sigaction = handle_sigchld;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = SA_SIGINFO;

    //register sigchld
    if(sigaction(SIGCHLD, &sa, NULL) == -1){
        perror("sigaction");
        return 1;
    }

    //register sigint
    if (sigaction(SIGINT, &act, NULL) == -1) {
        perror("sigaction");
        return 1;
    }

    //
    if (sigaction(SIGTERM, &act, NULL) == -1) {
        perror("sigaction");
        return 1;
    }
```

Here the signal handler for child processes if program gets SIGKILL signal. It safely terminates process to prevent zombie processes.

```c
void handle_sigchld(int signum, siginfo_t *siginfo, void *context) { // h
    pid_t pid;
    int status;
    pid = waitpid(siginfo->si_pid, &status, WNOHANG);
    if (pid > 0) {
        if (WIFSIGNALED(status) && WTERMSIG(status) == SIGKILL) {
            printf("Child process %d was killed with SIGKILL.\n", pid);
            kill(getpid(), SIGTERM);
        }
    }
}
```

After signal handlers, it goes through infinite loop and gets command line inputs from until user inputs ‘:q’.

```
while(1){

    //--------------- GET INPUT FROM USER AND PARSE IT -----------------
    printf("Enter a command: ");
    fgets(command, MAX_COMMAND_LEN, stdin); // get input from user

    //if input is :q then exit
    if (strcmp(command, ":q\n") == 0) { // if user enters :q, exit
        exit(EXIT_SUCCESS);
        return 0;
    }

    // parse the command
    commands[number_of_commands] = strtok(command, "|");
    while (commands[number_of_commands] != NULL && number_of_commands < MAX_ARGS) {
        number_of_commands++;
        commands[number_of_commands] = strtok(NULL, "|");
    }
    for (int j = 0; j < number_of_commands; j++) {
        trim(commands[j]); // trim whitespace from the beginning and end of each command
    }
```

After that, I created (n-1) pipe for communication between processes.

```
//---------------- Create Pipes ----------------
int pipefds[number_of_commands - 1][2]; // stores the file descriptors for the pipes

for (int i = 0; i < number_of_commands - 1; i++) {
    pipe(pipefds[i]);
}
//----------------------------------------------

pid_t children[number_of_commands];
int status;
```

Then I start for loop for each command that user gave. I parsed it to ‘args’ string array seperated by whitespace character. Also called fork() syscall and created child process. In each iteration, this code block parse single command with its arguments. For example “sort file1.txt > file2.txt” will be something like this {“sort”, “file1.txt”, “>”, “file2.txt”}.

```
// Loop through each command and execute it in a child process
for (int i = 0; i < number_of_commands; i++) {

    children[i] = fork(); //

    int number_of_args = 0;
    char* args[MAX_ARGS]; // stores the arguments of the command
    args[number_of_args] = strtok(commands[i], " "); // parse the command

    while (args[number_of_args] != NULL && number_of_args < MAX_ARGS) { // store the arguments in the args array
        number_of_args++;
        args[number_of_args] = strtok(NULL, " ");
    }
    args[number_of_args] = NULL;
```

After checking errors in fork() syscall, I do some works for child process. I close unnecessary pipe file descriptors and redirect STDOUT and STDIN to pipe. In second ‘If’ block, I make operations special to first command, last command and commands in between.

```c
if (children[i] == -1) { // fork failed
    perror("fork");
    exit(EXIT_FAILURE);
}

else if (children[i] == 0) { // child process
    for (int j = 0; j < number_of_commands - 1; j++) { // close unnecessary pipes
            if(j==i-1){
                close(pipefds[j][1]);
            }
            else if(j==i){
                close(pipefds[j][0]);
            }
            else{
                close(pipefds[j][0]);
                close(pipefds[j][1]);
            }
        }
    if(number_of_commands>1){
        if(i==0){ // first command
            dup2(pipefds[0][1], STDOUT_FILENO);
            close(pipefds[0][1]);
        }
        else if(i==number_of_commands-1){ // last command
            dup2(pipefds[i-1][0], STDIN_FILENO);
            close(pipefds[i-1][0]);
        }
        else{  // middle command
            dup2(pipefds[i-1][0], STDIN_FILENO);
            dup2(pipefds[i][1], STDOUT_FILENO);
            close(pipefds[i-1][0]);
            close(pipefds[i][1]);
        }
    }
```

Then I check If there is a redirection with ‘>’ and ‘<’ . If there is a redirection, I do dup2 syscall and set input and output of command.

```c
// check if there is a '<' or '>' symbol in the command
char* input_file = NULL;
char* output_file = NULL;
for (int i = 1; args[i] != NULL; i++) {
    if (strcmp(args[i], "<") == 0)
        input_file = args[i+1];
    else if (strcmp(args[i], ">") == 0)
        output_file = args[i+1];
}

for (int i = 1; args[i] != NULL; i++) {
    // find first '<' or '>' symbol
    if (strcmp(args[i], "<") == 0 || strcmp(args[i], ">") == 0) {
        // remove '<' or '>' and everything after it from args array
        args[i] = NULL;
        break;
    }
}
```

And this is code block that I redirect STDIN and STDOUT to the file which is given by user with command argument. If input_file and output_file pointers are not NULL, I open this files, make dup2 and close the file descriptors because we don't need them from now on.

```c
// redirect input to the input file
if (input_file != NULL) {
    int input_fd = open(input_file, O_RDONLY);
    if (input_fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }
    dup2(input_fd, STDIN_FILENO);
    close(input_fd);
}

// redirect output to the output file
if (output_file != NULL) {
    int output_fd = open(output_file, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (output_fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }
    dup2(output_fd, STDOUT_FILENO);
    close(output_fd);
}
```

In this code block, I create log file for each child process. Log file contains, pid and the command that processed by the child with this pid. Also name of the file has the time details.

14-04-2023-15-30-10_child_0.log

```
1   Child process 0 pid: 5986
2   Executed command: cat < file1.txt > file2.txt
3
```

At the end of the code block, I call execvp() syscall which takes two arguments the first is the name of the program to execute, and the second is a pointer to an array of pointers to null-terminated strings that represent the arguments to the program. Also make error checking and exit for termination of process.

```c
char filename[100];
time_t now = time(NULL);
struct tm *tm = localtime(&now);
sprintf(filename, "%02d-%02d-%04d-%02d-%02d-%02d_child_%d.log",
    tm->tm_mday, tm->tm_mon + 1, tm->tm_year + 1900,
    tm->tm_hour, tm->tm_min, tm->tm_sec, i);

FILE* log_file = fopen(filename, "w");
if(log_file == NULL) {
    fprintf(stderr, "Failed to create log file %s\n", filename);
    exit(1);
}

fprintf(log_file, "Child process %d pid: %d\n", i, getpid());

// Write the executed command to the log file
fprintf(log_file, "Executed command: ");
for (int j = 0; args[j] != NULL; j++) {
    fprintf(log_file, "%s ", args[j]);
}

// Log input and output files if they exist
if (input_file != NULL) {
    fprintf(log_file, "< %s ", input_file);
}
if (output_file != NULL) {
    fprintf(log_file, "> %s ", output_file);
}
fprintf(log_file, "\n");

fclose(log_file);

if (execvp(args[0], args) == -1) {
    perror("execvp failed : ");
}
exit(0);
```

So, when we come to the parent process part, I closed all pipes that I created before loop. The second loop waits for all of the child processes to complete using the waitpid system call. It also checks the status of each child process to determine whether it exited normally or was terminated due to an unhandled signal.

After all child processes have completed, the number_of_commands variable is reset to zero in preparation for the next iteration of the program.

```c
// Close pipes in the parent process
for (int i = 0; i < number_of_commands - 1; i++) {
    close(pipefds[i][0]);
    close(pipefds[i][1]);
}

// Wait for all child processes to complete
for (int i = 0; i < number_of_commands; i++) {

    waitpid(children[i], &status, 0);

    // check if child process exited normally
    if (WIFEXITED(status)) {
        printf("Child process exited with status %d\n", WEXITSTATUS(status));
    } else if (WIFSIGNALED(status)) {
        printf("Child process terminated due to unhandled signal %d\n", WTERMSIG(status));
    }
}

// Reset the number_of_commands for the next iteration
number_of_commands = 0;
```

==========

========== TEST CASES ==========

========== ==========

1 - ls -la | sort -k 5nr | head -n 10 > file2.txt

```
Enter a command: ls -la | sort -k 5nr | head -n 10 > file2.txt
Number of commands: 3
Child process exited with status 0
Child process exited with status 0
Child process exited with status 0
Enter a command:
```

file2.txt

```
1    -rwxr-xr-x 1 erselwsl erselwsl 21312 Apr 14 15:24 t21
2    -rwxr-xr-x 1 erselwsl erselwsl 21312 Apr 14 17:35 t22
3    -rwxr-xr-x 1 erselwsl erselwsl 21088 Apr 14 14:04 t20
4    -rwxr-xr-x 1 erselwsl erselwsl 20776 Apr 14 11:32 t17
5    -rwxr-xr-x 1 erselwsl erselwsl 20776 Apr 14 11:40 t18
6    -rwxr-xr-x 1 erselwsl erselwsl 16680 Apr 14 09:15 t15
7    -rwxr-xr-x 1 erselwsl erselwsl 16632 Apr 14 12:43 t19
8    -rwxr-xr-x 1 erselwsl erselwsl 16264 Apr 14 11:12 t1
9    -rw-r--r-- 1 erselwsl erselwsl 12950 Apr 14 11:40 test18.c
10   -rw-r--r-- 1 erselwsl erselwsl 12597 Apr 14 11:31 test17.c
11
```

2 - ls -l | grep file | sort -rn > file2.txt

```
Enter a command: ls -l | grep file | sort -rn > file2.txt
Number of commands: 3
Child process exited with status 0
Child process exited with status 0
Child process exited with status 0
Enter a command:
```

file2.txt

```
-rw-r--r-- 1 erselwsl erselwsl   463 Apr 14 15:31 file_list.txt
-rw-r--r-- 1 erselwsl erselwsl   356 Apr 14 06:41 sorted_files.txt
-rw-r--r-- 1 erselwsl erselwsl    32 Apr 14 06:35 file4.txt
-rw-r--r-- 1 erselwsl erselwsl    31 Apr 14 17:33 merged_files.txt
-rw-r--r-- 1 erselwsl erselwsl    31 Apr 14 05:58 file3.txt
-rw-r--r-- 1 erselwsl erselwsl    14 Apr 14 08:27 file1.txt
-rw-r--r-- 1 erselwsl erselwsl     0 Apr 14 17:35 file2.txt
-rw-r--r-- 1 erselwsl erselwsl     0 Apr 14 06:50 file_sizes.txt
```

14-04-2023-17-35-40_child_0.log

```
Child process 0 pid: 11812
Executed command: ls -l
```

14-04-2023-17-35-40_child_1.log

```
Child process 1 pid: 11813
Executed command: grep file
```

14-04-2023-17-35-40_child_2.log

```
Child process 2 pid: 11814
Executed command: sort -rn > file2.txt
```

3 - sort < file1.txt > file2.txt

file1.txt

```
e
r
a
p
b
m
t
```

file2.txt

```
a
b
e
m
p
r
t
```

14-04-2023-17-37-08_child_0.log

```
Child process 0 pid: 12269
Executed command: sort < file1.txt > file2.txt
```

4 – ls

```
Enter a command: ls -l | grep file | sort -rn > file2.txt
Number of commands: 3
Child process exited with status 0
Child process exited with status 0
Child process exited with status 0
Enter a command: sort < file1.txt > file2.txt
Number of commands: 1
Child process exited with status 0
Enter a command: ls
Number of commands: 1
14-04-2023-15-26-55_child_0.log  14-04-2023-15-34-48_child_0.log  14-04-2023-15-42-01_child_2.log  14-04-2023-17-35-40_child_2.log  file3.txt          output.txt       t18     test15.c  test21.c
14-04-2023-15-28-06_child_0.log  14-04-2023-15-35-01_child_0.log  14-04-2023-17-33-38_child_0.log  14-04-2023-17-37-08_child_0.log  file4.txt          sorted_files.txt  t19     test16.c  test22.c
14-04-2023-15-29-11_child_0.log  14-04-2023-15-35-01_child_1.log  14-04-2023-17-33-38_child_1.log  14-04-2023-17-38-51_child_0.log  file_list.txt      sorted_lines.txt  t20     test17.c
14-04-2023-15-30-10_child_0.log  14-04-2023-15-35-01_child_2.log  14-04-2023-17-33-38_child_2.log  comm.txt                         file_sizes.txt     t1                t21     test18.c
14-04-2023-15-31-31_child_0.log  14-04-2023-15-42-01_child_0.log  14-04-2023-17-35-40_child_0.log  file1.txt                        important_lines.txt  t15             t22     test19.c
14-04-2023-15-31-51_child_0.log  14-04-2023-15-42-01_child_1.log  14-04-2023-17-35-40_child_1.log  file2.txt                        merged_files.txt   t17               test1.c  test20.c
Child process exited with status 0
Enter a command:
```

14-04-2023-17-38-51_child_0.log

```
1  Child process 0 pid: 12578
2  Executed command: ls
3
```

Here the screenshot that shows after running “ls | sort –r > file2.txt” and “ls > file_list.txt”, program is terminated with command “:q” and checks processes. There is no zombie process.

```
erselwsl@DESKTOP-T4MTM6P:~/VSCODE_SRC/System/HW2$ ./terminal

Enter a command: ls | sort -r > file2.txt
Number of commands: 2
Child process exited with status 0
Child process exited with status 0

Enter a command: ls > file_list.txt
Number of commands: 1
Child process exited with status 0

Enter a command: :q
erselwsl@DESKTOP-T4MTM6P:~/VSCODE_SRC/System/HW2$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0   1408   800 ?        Sl   03:57   0:01 /init
root       13693  0.0  0.0   1320   444 ?        Ss   17:44   0:00 /init
root       13694  0.0  0.0   1320   444 ?        S    17:44   0:00 /init
erselwsl 13695  0.0  0.3 592204 47796 pts/1    Ssl+ 17:44   0:00 /home/erselwsl/.vscode-server/bin/704ed70d4fd1c6bd6342c436f1ede30d1cff4710/node -e  const net = require('net'); process.stdin.pause(); const client = net.creat
root       13702  0.0  0.0   1320   444 ?        Ss   17:44   0:00 /init
root       13703  0.0  0.0   1320   444 ?        S    17:44   0:00 /init
erselwsl 13704  0.0  0.3 583704 40828 pts/2    Ssl+ 17:44   0:00 /home/erselwsl/.vscode-server/bin/704ed70d4fd1c6bd6342c436f1ede30d1cff4710/node -e  const net = require('net'); process.stdin.pause(); const client = net.creat
erselwsl 13711  0.0  0.3 835580 45272 pts/0    Sl+  17:44   0:00 /home/erselwsl/.vscode-server/bin/704ed70d4fd1c6bd6342c436f1ede30d1cff4710/node /home/erselwsl/.vscode-server/bin/704ed70d4fd1c6bd6342c436f1ede30d1cff4710/out/
erselwsl 13724  0.8  1.5 11551380 197084 pts/0 Sl+  17:44   0:06 /home/erselwsl/.vscode-server/bin/704ed70d4fd1c6bd6342c436f1ede30d1cff4710/node /home/erselwsl/.vscode-server/bin/704ed70d4fd1c6bd6342c436f1ede30d1cff4710/out/
erselwsl 13785  0.0  0.0   6232  5196 pts/5    Ss   17:45   0:00 /bin/bash --init-file /home/erselwsl/.vscode-server/bin/704ed70d4fd1c6bd6342c436f1ede30d1cff4710/out/vs/workbench/contrib/terminal/browser/media/shellIntegrati
erselwsl 13836  0.5  0.5 1506072 67260 pts/0   Sl+  17:49   0:02 /home/erselwsl/.vscode-server/extensions/ms-vscode.cpptools-1.14.5-linux-x64/bin/cpptools
erselwsl 14026  0.0  0.0   2772  1116 pts/5    T    17:51   0:00 ./terminal
erselwsl 14334  0.0  0.1 5541268 21032 pts/0   Sl+  17:53   0:00 /home/erselwsl/.vscode-server/extensions/ms-vscode.cpptools-1.14.5-linux-x64/bin/cpptools-srv 13836 {DA73D6CE-B9D9-4D62-967E-7773ADAB6B04}
erselwsl 14608  0.0  0.0   7476  3224 pts/5    R+   17:57   0:00 ps aux
root       20211  0.0  0.0   1060   184 ?        Ss   10:03   0:00 /init
root       20212  0.0  0.0   1060   184 ?        S    10:03   0:00 /init
erselwsl 20213  0.0  0.0   2884   980 pts/0    Ss+  10:03   0:00 sh -c "$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh" 704ed70d4fd1c6bd6342c436f1ede30d1cff4710 stable code-server .vscode-server --host=127.0.0.1 --port=0 --co
erselwsl 20214  0.0  0.0   2884   952 pts/0    S+   10:03   0:00 sh /mnt/c/Users/ersel/.vscode/extensions/ms-vscode-remote.remote-wsl-0.77.0/scripts/wslServer.sh 704ed70d4fd1c6bd6342c436f1ede30d1cff4710 stable code-server .v
erselwsl 20219  0.0  0.0   2884   952 pts/0    S+   10:03   0:00 sh /home/erselwsl/.vscode-server/bin/704ed70d4fd1c6bd6342c436f1ede30d1cff4710/bin/code-server --host=127.0.0.1 --port=0 --connection-token=2159723650-227612183
erselwsl 20223  0.1  0.6 950816 90288 pts/0    Rl+  10:03   0:30 /home/erselwsl/.vscode-server/bin/704ed70d4fd1c6bd6342c436f1ede30d1cff4710/node /home/erselwsl/.vscode-server/bin/704ed70d4fd1c6bd6342c436f1ede30d1cff4710/out/
erselwsl 20242  0.0  0.4 809580 64488 pts/0    Rl+  10:03   0:21 /home/erselwsl/.vscode-server/bin/704ed70d4fd1c6bd6342c436f1ede30d1cff4710/node /home/erselwsl/.vscode-server/bin/704ed70d4fd1c6bd6342c436f1ede30d1cff4710/out/
```

> Other tested inputs :

sort file1.txt > file2.txt

ls > file_list.txt

cat file1.txt file2.txt | sort > output.txt

ls | sort -r > file2.txt

head -n 3 file1.txt | sort > file2.txt

ls -l | grep t > output.txt

cat file1.txt > file2.txt

cat < file1.txt > file2.txt

sort -r < file1.txt > file2.txt

sort file2.txt > output.txt

cat file1.txt file2.txt | sort | uniq > merged_files.txt

cat file4.txt | sort -r | uniq > file2.txt

cat file1.txt file2.txt file3.txt | grep important | sort > important_lines.txt

To Run the Program :

> Write *“make”* . Makefile will compile *“terminal.c”* .

> Run the program with *“./terminal”*