MARMARA UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE-3033

OPERATING SYSTEMS

PROJECT 2

Number	First Name	Last Name
150119566	Müslim	Yılmaz
150119037	Ömer	Kibar
150119036	Serkan	Korkut

Problem Description

- We will design a simple shell with the given details in project document. Shell will focus on the three main themes which are:
    - It will take the command as input and will execute that in a new process.
    - It will support the given internal commands in project documents. (history, ^Z, fg %num, exit)
    - It will support the I/O-redirection for stdin, stdout and stderr.

Detailed Code Description

```
typedef struct ProcessLL
{
    pid_t processId;
    struct ProcessLL *next;
    struct ProcessLL *previous;
} ProcessLL;
```

- In order to keep track of background processes we implemented a doubly linked list **ProcessLL**. Each time user enters a command to be executed in background using & sign that process is added to end of the background processes linked list. When user types fg command with the parameter process id. First, we found the process in that linked list then put the process to the foreground.

```
typedef struct HistoryLL{
    char* str;
    char inputBuffer[MAX_LINE];
    int background;
    char *args[MAX_LINE];
    struct HistoryLL* next;
}HistoryLL;
```

- In order to list the last 10 commands entered by user we implemented linked list called **HistoryLL**. When a new command is executed, we create a history node and add this node to the beginning of HistoryLL. Every history node contains inputBuffer, args, background and a pointer to next history node. These members are necessary in order to re-execute that command in case user enters history with option i.

```c
static void stopSignalHandler(int signo)
{
    printf("\nmyshell: ");
    fflush(stdout);
}
```

- If ctrl-z signal arrives to the shell, that means there is no foreground process in the shell. In this case ctrl-z should have no effect. So in the beginning of the main we set signal handler for ctrl-z to stopSignalHandler that only goes to next line.

```c
ProcessLL *bgProcessesHead = (ProcessLL *)malloc(sizeof(ProcessLL));
bgProcessesHead->previous = NULL;
bgProcessesHead->next = NULL;
ProcessLL *bgProcessesTail = bgProcessesHead;
```

- In the beginning of the main function, we initialize the linked list for background processes.

Command Execution Details

- After we get input from the user using setup function we check if the given command is a shell built-in function like history or fg.
    - If entered command is a history command with no option, we just print the history linked list's first 10 entries.
    - If entered command is a history command with -i option, we find the specified node in the history linked list using index given in the third argument and execute that command with parameters we already saved.
    - If entered command is fg %num, we first check if the process id exists in the background processes linked list. If it exists, we put it in the foreground and wait for it.
    - If entered command is exit, we first check if there is a background process in the background processes linked list. If there is no background process the program will terminate otherwise prints a message and doesn't terminate.
- If it isn't a shell built-in command, we first put that command in to the history linked list. In main, inputBuffer, args and background variables contain information about that command. So, we keep these variables as members in a newly created node. Then we add the node we created to the beginning of the history linked list. This way we can use the history linked list later for history commands.
- After that, we get the command path using getCommandPath function. Then using createChildProcess function we fork a new child that will execute the entered command. If the command is entered with & sign, we do not wait for the newly created process otherwise we wait for the newly created process.

```
if (!background)
{
    int status;
    waitpid(childProcess, &status, WUNTRACED);
    if (WIFSTOPPED(status))
    {
        printf("\n");
        killpg(getpgid(childProcess), SIGKILL);
        waitpid(childProcess, NULL, 0);
    }
    tcsetpgrp(STDIN_FILENO, getpgid(getpid()));
}
```

We used waitpid function with WUNTRACED option which doesn't block execution of the parent process when the child stopped with the ctrl-z signal. So after waitpid we check the status if it is stopped meaning ctrl-z signal arrived to the children so we kill the children process group using killpg in order to also kill any descendants of child.

- WIFSTOPPED (status): returns true if the child process was stopped by delivery of a signal; this is only possible if the call was done using **WUNTRACED** or when the child is being traced[1]

```
else
{
    bgProcessesTail->processId = childProcess;
    bgProcessesTail->next = (ProcessLL *)malloc(sizeof(ProcessLL));
    bgProcessesTail->next->previous = bgProcessesTail;
    bgProcessesTail = bgProcessesTail->next;
    bgProcessesTail->next = NULL;
}
```

- If it is a background process it adds the created processes to the background process linked list

```
char * getCommandPath(char *command)
```

- Search the PATH for the executable file of the command given as an argument.

```
pid_t createChildProcess(char *path,char **args,int background)
```

- This function analyzes the arguments and determines the stdin, stdout and stderr file of the child process then forks the child assign it is file descriptors and also put the child into a new process group and gives the control of the terminal to the child if child isn't a background process. Then child executes the command using execv function.

[1] https://linux.die.net/man/2/waitpid