Operating Systems

SIMPLE SHELL

(LAB 1)

Marwan ElSafty | 4690

Problem Statement:

It is required to implement a Unix shell program. A shell is simply a program that conveniently allows you to run other programs.

Your shell must support the following:

1. The internal shell command "exit" which terminates the shell

2. A command with no arguments

3. A command with arguments

4. A command, with or without arguments, executed in the background using &.

Used functions:

The code contains 7 functions in addition to the main function, all in one C source file.

1. getInput():
   function used to scan the input from user, if the user did not enter a command it returns 1, else it returns 0.

```c
int getInput()
{
    fgets(input,MAX_CHAR,stdin);

    //if there's no input return 1
    if(strlen(input) == 0)
        return 1;

    else
          return 0;

 }
```

2. splitInput():
   function that uses the string tokenizer to split the inserted string into words separated by a space.
   It also omits the newline symbol from the end of commands.

```c
void splitInput()
{

    const char s[2] = " ";
    char *token;
    token = strtok(input, s);

    int i=0;
    while( token != NULL )
      {
          tokenizedArgs[i] = token;
          token = strtok(NULL, s);
          //the upcoming line omits the \n found in the end of a command
          tokenizedArgs[i][strcspn(tokenizedArgs[i], "\n")] = 0;
          if (strlen(tokenizedArgs[i]) == 0)
              i--;
```

```c
        i++;
    }

    //Last string must be NULL for execvp() to execute
    tokenizedArgs[i] = NULL;

}
```

3. isBuiltInCommands:

   this function checks if the requested command is a built in command or one that needs to be implemented, like ‘exit’ and ‘cd’.
   It returns 0 if the command is built in, 1 if it’s an exit command and 2 if it’s a change dir command.

```c
int isBuiltInCommands()
{
    char * implementedCommands[2];
    int flag=0;
    implementedCommands[0] = "exit";
    implementedCommands[1] = "cd";

    for(int i=0 ; i < 2 ; i++)
    {
        if(strcmp(tokenizedArgs[0],implementedCommands[i]) == 0)
        {
            flag = i+1;
            break;
        }
    }

    //return 0 if it is a built in command
    if(flag == 0)
        return 0;
    else if(flag==1)
        return 1;
    else if(flag==2)
    {
        chdir(tokenizedArgs[1]);
        return 2;
    }


    return 0;
}
```

4. isBackgroundCommand:
   a function that checks for the existance of '&' in the command to know whether it is a background command that needs to run in parallel with the parent process or not.

```c
int isBackgroundCommand()
{
    for(int i=0; i < strlen(input); i++)
    {
        if(input[i] == '&')
            return 1;
    }
    return 0;
}
```

5. executeCommands:
   a function that execute the user inserted commands by forking a child.
   The child has a process id, if it's a -1 then the child failed to fork, if it's a 0 then the child is currently being executed, and if it's a positive number then the parent is currently being executed so we must use the wait() function to wait for the child to terminate.

```c
void executeCommands()
{
    // pid is the process ID
    int pid = fork();

    // when pid=-1 that means there is a failure in creating a child
    if(pid == -1)
    {
        printf("Failed to create a child!");
          return;
      }
      // when pid=0 that means the child is created and being executed
      else if(pid == 0)
      {
          if (execvp(tokenizedArgs[0], tokenizedArgs) < 0)
          {
              printf("\nFailed to execute Command..");
          }
          exit(0);
      }
      // when pid is +ve that means the execution is back to the parent
      else
      {
          wait(NULL);
          signal(SIGCHLD,sigHandler);
          return;
      }
 }
```

6. executeBackgroundCommand:

the same function as executeCommand() but the core difference is that the parent process doesn’t wait for the child to terminate, so I didn’t use the wait() function in the third if condition.

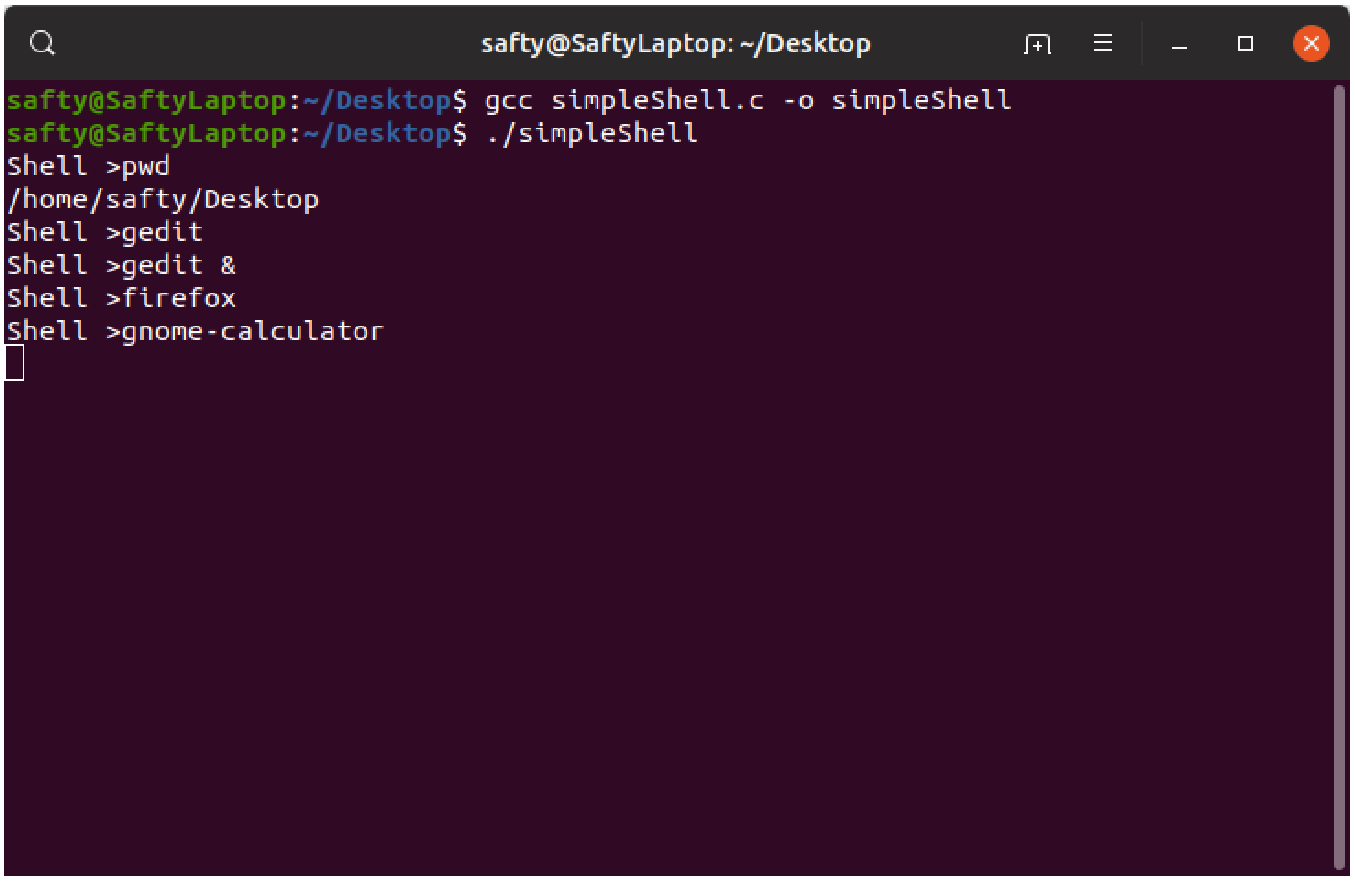
```c
void executeBackgroundCommand()
{
    // pid is the process ID
    int pid = fork();

    // when pid=-1 that means there is a failure in creating a child
    if(pid == -1)
    {
        printf("Failed to create a child!");
        return;
    }

    // when pid=0 that means the child is created and being executed
    else if(pid == 0)
    {
        if (execvp(tokenizedArgs[0], tokenizedArgs) < 0)
        {
            printf("\nFailed to execute Command..");
        }
        exit(0);
    }

    // when pid is +ve that means the execution is back to the parent
    else
    {
        signal(SIGCHLD,sigHandler);
        return;
    }
}
```

7. sigHandler:

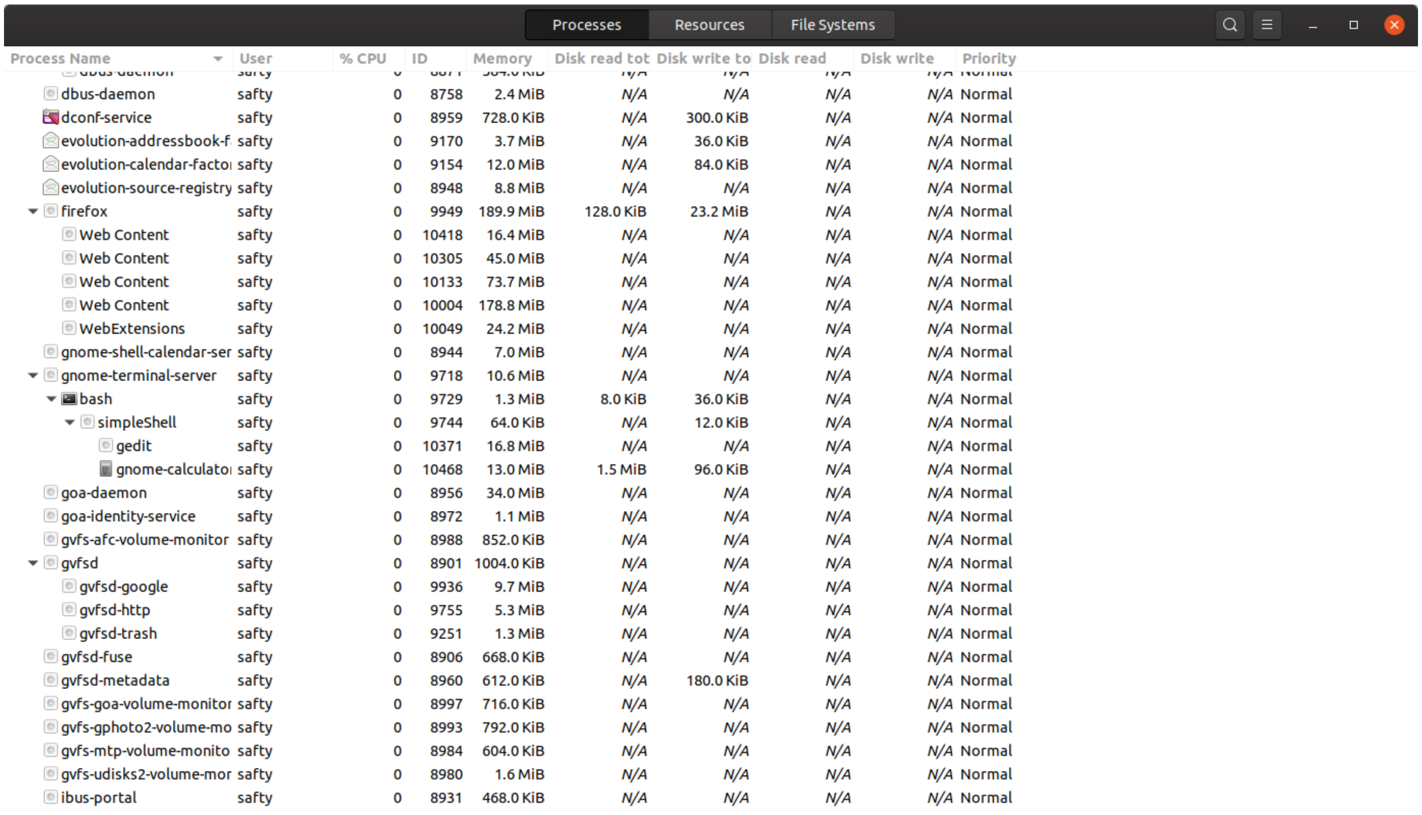
a function which prints to a log file that a child is terminated.
It is called by the signal() function whenever the parent receives a SIGCHLD signal which indicates that the child process has terminated.

```c
void sigHandler()
{
    FILE *fp ;
    fp = fopen("logFile.txt", "a");
    fprintf(fp, "%s\n", "Child process was terminated");
}
```

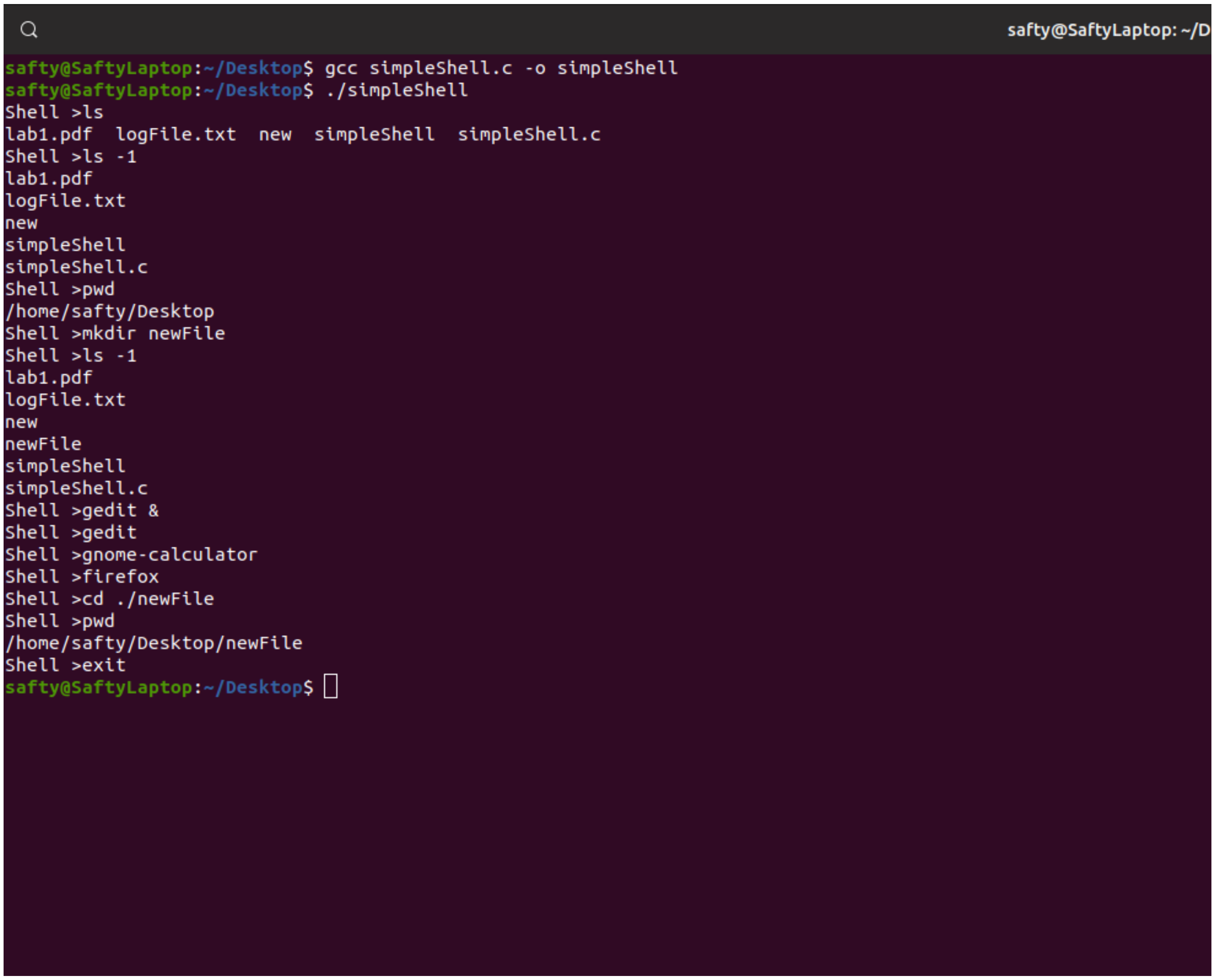
Screenshots of sample runs:



```
safty@SaftyLaptop:~/Desktop$ gcc simpleShell.c -o simpleShell
safty@SaftyLaptop:~/Desktop$ ./simpleShell
Shell >pwd
/home/safty/Desktop
Shell >gedit
Shell >gedit &
Shell >firefox
Shell >gnome-calculator
```



Process Name	User	% CPU	ID	Memory	Disk read tot	Disk write to	Disk read	Disk write	Priority
dbus-daemon	safty	0	8758	2.4 MiB	N/A	N/A	N/A	N/A	Normal
dconf-service	safty	0	8959	728.0 KiB	N/A	300.0 KiB	N/A	N/A	Normal
evolution-addressbook-f	safty	0	9170	3.7 MiB	N/A	36.0 KiB	N/A	N/A	Normal
evolution-calendar-factoi	safty	0	9154	12.0 MiB	N/A	84.0 KiB	N/A	N/A	Normal
evolution-source-registry	safty	0	8948	8.8 MiB	N/A	N/A	N/A	N/A	Normal
firefox	safty	0	9949	189.9 MiB	128.0 KiB	23.2 MiB	N/A	N/A	Normal
Web Content	safty	0	10418	16.4 MiB	N/A	N/A	N/A	N/A	Normal
Web Content	safty	0	10305	45.0 MiB	N/A	N/A	N/A	N/A	Normal
Web Content	safty	0	10133	73.7 MiB	N/A	N/A	N/A	N/A	Normal
Web Content	safty	0	10004	178.8 MiB	N/A	N/A	N/A	N/A	Normal
WebExtensions	safty	0	10049	24.2 MiB	N/A	N/A	N/A	N/A	Normal
gnome-shell-calendar-ser	safty	0	8944	7.0 MiB	N/A	N/A	N/A	N/A	Normal
gnome-terminal-server	safty	0	9718	10.6 MiB	N/A	N/A	N/A	N/A	Normal
bash	safty	0	9729	1.3 MiB	8.0 KiB	36.0 KiB	N/A	N/A	Normal
simpleShell	safty	0	9744	64.0 KiB	N/A	12.0 KiB	N/A	N/A	Normal
gedit	safty	0	10371	16.8 MiB	N/A	N/A	N/A	N/A	Normal
gnome-calculatoi	safty	0	10468	13.0 MiB	1.5 MiB	96.0 KiB	N/A	N/A	Normal
goa-daemon	safty	0	8956	34.0 MiB	N/A	N/A	N/A	N/A	Normal
goa-identity-service	safty	0	8972	1.1 MiB	N/A	N/A	N/A	N/A	Normal
gvfs-afc-volume-monitor	safty	0	8988	852.0 KiB	N/A	N/A	N/A	N/A	Normal
gvfsd	safty	0	8901	1004.0 KiB	N/A	N/A	N/A	N/A	Normal
gvfsd-google	safty	0	9936	9.7 MiB	N/A	N/A	N/A	N/A	Normal
gvfsd-http	safty	0	9755	5.3 MiB	N/A	N/A	N/A	N/A	Normal
gvfsd-trash	safty	0	9251	1.3 MiB	N/A	N/A	N/A	N/A	Normal
gvfsd-fuse	safty	0	8906	668.0 KiB	N/A	N/A	N/A	N/A	Normal
gvfsd-metadata	safty	0	8960	612.0 KiB	N/A	180.0 KiB	N/A	N/A	Normal
gvfs-goa-volume-monitor	safty	0	8997	716.0 KiB	N/A	N/A	N/A	N/A	Normal
gvfs-gphoto2-volume-mo	safty	0	8993	792.0 KiB	N/A	N/A	N/A	N/A	Normal
gvfs-mtp-volume-monito	safty	0	8984	604.0 KiB	N/A	N/A	N/A	N/A	Normal
gvfs-udisks2-volume-mor	safty	0	8980	1.6 MiB	N/A	N/A	N/A	N/A	Normal
ibus-portal	safty	0	8931	468.0 KiB	N/A	N/A	N/A	N/A	Normal



```
safty@SaftyLaptop:~/Desktop$ gcc simpleShell.c -o simpleShell
safty@SaftyLaptop:~/Desktop$ ./simpleShell
Shell >ls
lab1.pdf  logFile.txt  new  simpleShell  simpleShell.c
Shell >ls -1
lab1.pdf
logFile.txt
new
simpleShell
simpleShell.c
Shell >pwd
/home/safty/Desktop
Shell >mkdir newFile
Shell >ls -1
lab1.pdf
logFile.txt
new
newFile
simpleShell
simpleShell.c
Shell >gedit &
Shell >gedit
Shell >gnome-calculator
Shell >firefox
Shell >cd ./newFile
Shell >pwd
/home/safty/Desktop/newFile
Shell >exit
safty@SaftyLaptop:~/Desktop$ 
```