Mini Shell Project Report

Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam

Department of Computer Science & Engineering

M.Tech CSE - III Semester (2025–26)

Course: ICS1313 – Operating System Practices Laboratory

Experiment No: 1

Name: Simiyon vinscent Samuel L

Reg no:3122247001062

Title:

Implementing a Shell with Basic UNIX Command Functionality Using System Calls

Objective:

To design and implement a Mini Shell in C that supports basic UNIX commands using low-level system calls such as fork(), exec(), getpid(), exit(), wait(), close(), stat(), opendir(), and readdir() to enhance understanding of process and file system handling.

Commands Implemented:

- 1. ls <directory_path> – Lists the contents of the specified directory.
- 2. cat <file_path> – Displays the contents of the specified file.
- 3. exit – Exits the shell.

System Calls Used and Characteristics:

System Call	Purpose	Characteristics	Reason for Use
fork()	Creates a child process	Returns 0 to child, PID to parent	To run commands in parallel without interrupting main shell
exec()	Replaces current process image	Does not return on success	Used to execute external programs like ls, cat
getpid()	Returns process ID	Unique ID of calling process	Can be used to identify running process
exit()	Terminates process	Cleans up resources	Used to quit child and shell
wait()	Waits for child to finish	Blocks parent until child ends	To synchronize parent-child processes
close()	Closes file descriptor	Frees system resources	Used to properly close opened files
stat()	Gets file status	Fills structure with metadata	Used to get file properties like size

opendir()	Opens a directory stream	Returns DIR* pointer	Used for iterating through directories
readdir()	Reads next entry in dir	Returns pointer to dirent struct	Used to list files in a directory

Mini Shell Working:

1. List Directory (ls):

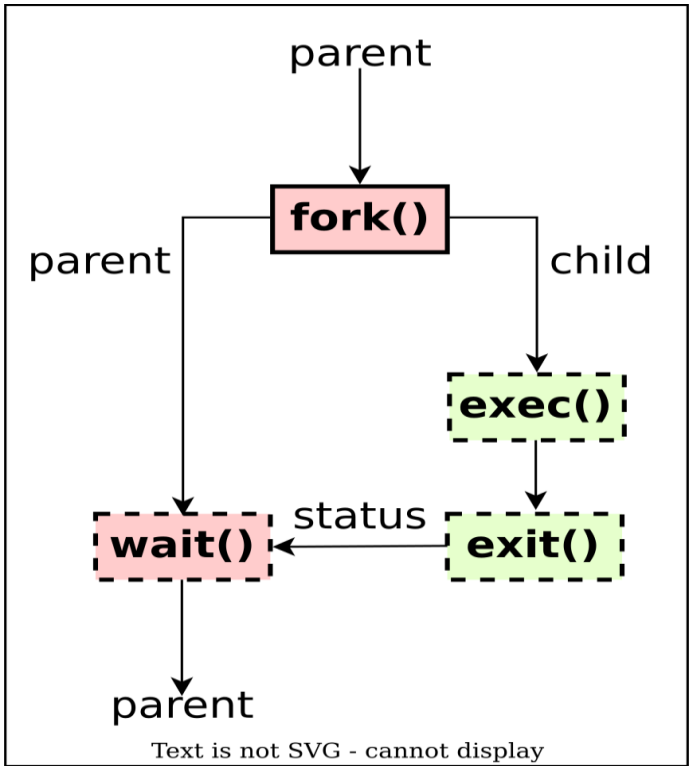
   Input: ls <directory_path> (If no directory is specified, list the current directory).

   Output: List of files and subdirectories within the specified or current directory.

```c
void ls(){
    char dict[255];
    printf("enter the directory path:");
    scanf("%s",dict);
    pid_t pid=fork();
    if(pid==0)
    {
        printf("child is running...\n");
        if(execl("/bin/ls","ls","-l",dict,NULL)==-1)
        {
        perror("invalid pathway using home path");
        execl("/bin/ls","ls","-l","/home/mtech1",NULL);
        }
        exit(0);
    }
    else if(pid>0)
    {
        wait(NULL);
        printf("successfully system process executed\n");
    }
    else
    printf("child creation failed!\n");
}
```

```
web@samsamuel:/mnt/c/Users/sam/OneDrive/one drive back up/OneDrive - S
SN-Institute/Documents/projects/os/a1$ ./a.out
1.ls
2.cat
3.exit
enter your choice:1
enter the directory path:/
child is running...
total 2832
drwxr-xr-x   3 root root    4096 Apr 15 18:14 Docker
lrwxrwxrwx   1 root root       7 Nov 23  2023 bin -> usr/bin
drwxr-xr-x   2 root root    4096 Apr 18  2022 boot
drwxr-xr-x  15 root root    3860 Jul 14 21:31 dev
drwxr-xr-x  94 root root    4096 Jul 14 23:13 etc
drwxr-xr-x   7 root root    4096 Oct 12  2024 home
-rwxrwxrwx   1 root root 2724480 Jun 10 00:02 init
lrwxrwxrwx   1 root root       7 Nov 23  2023 lib -> usr/lib
lrwxrwxrwx   1 root root       9 Nov 23  2023 lib32 -> usr/lib32
lrwxrwxrwx   1 root root       9 Nov 23  2023 lib64 -> usr/lib64
lrwxrwxrwx   1 root root      10 Nov 23  2023 libx32 -> usr/libx32
drwx------   2 root root   16384 Apr 10  2019 lost+found
drwxr-xr-x   2 root root    4096 Nov 23  2023 media
drwxr-xr-x   5 root root    4096 Sep 15  2024 mnt
drwxr-xr-x   2 root root    4096 Nov 23  2023 opt
dr-xr-xr-x 669 root root       0 Jul 14 21:31 proc
drwx------   6 root root    4096 Jun  8 19:24 root
drwxr-xr-x  23 root root     640 Jul 14 21:31 run
lrwxrwxrwx   1 root root       8 Nov 23  2023 sbin -> usr/sbin
drwxr-xr-x   8 root root    4096 Nov 23  2023 snap
drwxr-xr-x   2 root root    4096 Nov 23  2023 srv
dr-xr-xr-x  13 root root       0 Jul 14 21:31 sys
drwxrwxrwt   8 root root    4096 Jul 14 22:35 tmp
drwxr-xr-x  14 root root    4096 Nov 23  2023 usr
drwxr-xr-x  14 root root    4096 Dec 13  2024 var
drwx------   2 root root    4096 Dec 11  2024 wslDEBJME
drwx------   2 root root    4096 Feb 26 18:05 wslDIeiGp
drwx------   2 root root    4096 Dec 11  2024 wslDOFnhF
drwx------   2 root root    4096 Jan  7  2025 wslDpkpdl
drwx------   2 root root    4096 Dec 11  2024 wslGPkDIF
```

Explanation:



- **Prompts for Directory**: The C code asks the user for a directory path and uses fork() to create a child process to run ls -l on it.
- **Executes ls Command**: The child process executes /bin/ls -l with the provided path, falling back to /home/mtech1 if the path is invalid.
- **Parent Waits**: The parent process waits for the child to complete and prints a success message, or reports an error if fork() fails.

2. Concatenate and Display File (cat):

Input: cat <file_path> (Display the content of the specified file).

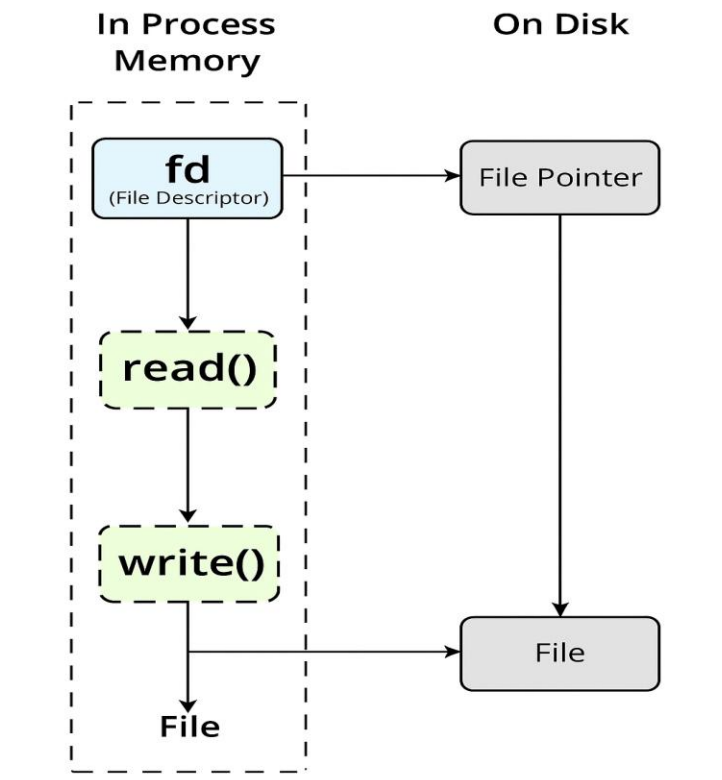
Output: Content of the file displayed on the terminal.

```c
void cat()
{
    char file[20];
    char buffer[1024];
    ssize_t bytes_read;
    printf("enter a file name:");
    scanf("%s",file);
    int fd = open(file, O_RDONLY, 0644);
    if (fd == -1)
    {
    perror("open failed");
    }
printf("File opened, descriptor: %d\n", fd);
    while ((bytes_read=read(fd,buffer,sizeof(buffer)))>0)
    {
        write(STDOUT_FILENO,buffer,bytes_read);
    }


close(fd);
}
```

```
web@samsamuel:/mnt/c/Users/sam/OneDrive/one drive back up/OneDrive - SSN
-Institute/Documents/projects/os/a1$ ./a.out
1.ls
2.cat
3.exit
enter your choice:2
enter a file name:hi.txt
File opened, descriptor: 3
hi!!!!!!!!!!!!
1.ls
2.cat
3.exit
enter your choice:
```

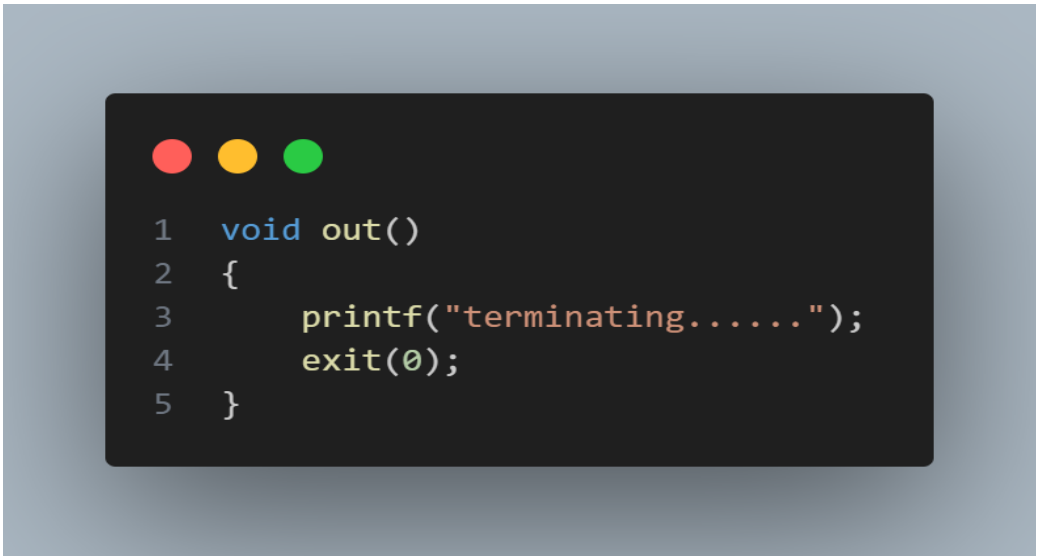
- **Prompts for File Path**: The C code prompts the user to enter a file path, stored in a 20-character array file.
- **Opens File**: It opens the specified file in read-only mode using open(), returning a file descriptor or reporting an error if it fails.
- **Reads and Displays Content**: The code reads the file in chunks (up to 1024 bytes) using read() and writes them to the terminal using write().
- **Closes File**: After reading, the file descriptor is closed with close() to free system resources.



3. Exit Shell (exit):

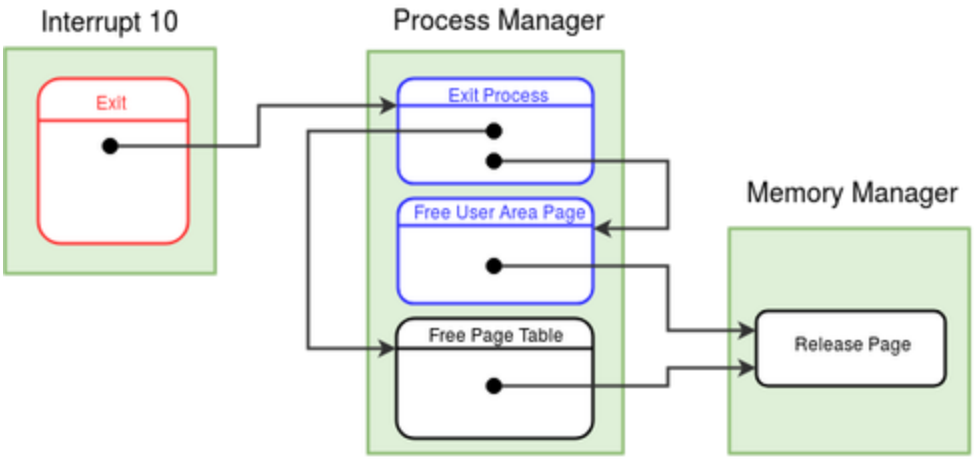
   Input: exit (Command to exit the shell).

   Output: Terminates the mini shell



```c
void out()
{
    printf("terminating......");
    exit(0);
}
```

```
web@samsamuel:/mnt/c/Users/sam/OneDrive/one drive back up/OneDrive - SSN-Institute/Documents/projects/os/a1$ ./a.out
1.ls
2.cat
3.exit
enter your choice:3
terminating......
web@samsamuel:/mnt/c/Users/sam/OneDrive/one drive back up/OneDrive - SSN-Institute/Documents/projects/os/a1$
```



- **Terminates Process**: The exit system call immediately terminates the calling process, freeing its resources and stopping execution.
- **Returns Status**: It sends an exit status code to the parent process (e.g., exit(0) for success, non-zero for failure).
- **Cleanup**: Ensures proper cleanup of the process, including closing open file descriptors and flushing I/O buffers.

Conclusion:

- This project provided hands-on experience with system-level programming using C.
- The implementation of a custom shell enhanced understanding of processes, file handling, and interaction with the UNIX operating system through direct use of system calls.