Operating Systems Lab Report Lab 1: Basic Shell Commands

Anuj Dangi
Roll Number:23MCCE16
Date: Aug 6, 2025

1. Basic Shell Commands and Observations

Command	Result	Observation
`date`	Displays the current system date and time.	Can be formatted using options like `+%d-%m-%Y`.
`cal`	Displays the current month's calendar.	Use `cal 2025` for the full year.
`echo Hello, World!`	Hello, World!	Prints the given string to standard output.
`man ls`	Manual for `ls` command.	Use / to search and `q` to quit.
`ls`	Lists files and directories.	Use `-l`, `-a` for more info.
`pwd`	Prints the current working directory.	Useful for knowing the current location.
`mkdir test_dir`	Creates a new directory named `test_dir`.	Use `-p` to create nested directories.
`cd test_dir`	Changes to the `test_dir` directory.	Use `cd ..` to go back.
`rmdir test_dir`	Removes an empty directory.	Fails if directory is not empty.
`cat file.txt`	Displays contents of `file.txt`.	Use >> to append and > to overwrite.
`sort file.txt`	Sorts lines of `file.txt` alphabetically.	Use `-r` for reverse order.
`cp file1.txt file2.txt`	Copies `file1.txt` to `file2.txt`.	Use `-r` to copy directories.
`mv old.txt new.txt`	Renames or moves a file.	Also used to move between directories.

Command	Result	Observation
`rm file.txt`	Deletes `file.txt`.	Use `-r` for directories and `-f` to force.
`wc file.txt`	Displays word, line, and byte count.	Use `-l`, `-w`, `-c` for specific stats.
`head file.txt`	Shows first 10 lines of the file.	Use `-n` to specify number of lines.
`tail file.txt`	Shows last 10 lines of the file.	Use `-f` to follow live changes.
`more file.txt`	Displays content page by page.	Press space to scroll, `q` to quit.
`cat file.txt | tr a-z A-Z`	Converts content to uppercase.	Demonstrates pipe and `tr` usage.

End of Lab 1 Report (L1_SA)

Operating Systems Lab Report
Lab 1: (Section B) Process Basics

Anuj Dangi
Roll Number: 23MCCE16
Date: Aug 6, 2025

1. Basic Process Commands and Observations

Command	Result	Observation
`ps`	Lists running processes for the user terminal	Use `ps aux` for all processes
`pstree`	Tree-like display of processes and their hierarchy	Visualizes parent-child relation
`top`	Real-time table of system processes and resources	Press q to quit, see top CPU/memory consumers
`pgrep firefox`	Shows PIDs for all running Firefox processes	Find PIDs of specific apps
`renice -n 10 -p 1234`	Changes priority of PID 1234 to 10	Only root can increase priority (negative value)
`kill 1234`	Sends SIGTERM to process 1234	Use `-9` for SIGKILL (force kill)
`xkill`	Mouse becomes tool to kill windows interactively	Useful for frozen GUI apps

2. C Program using `fork()`

```c
#include <stdio.h>
#include <unistd.h>
int main() {
    pid_t pid = fork();
    if (pid == 0) {
        printf("Child process. PID: %d, Parent PID: %d\n", getpid(),
getppid());
    } else if (pid > 0) {
        printf("Parent process. PID: %d, Child PID: %d\n", getpid(),
pid);
```

```
    } else {
        printf("Fork failed.\n");
    }
    return 0;
}
```

Observation: `fork()` creates a child process, both parent and child continue execution separately.

3. C Program using `exec()`

```
#include <stdio.h>
#include <unistd.h>
int main() {
    printf("Before exec\n");
    execlp("ls", "ls", "-l", NULL);
    printf("After exec (won’t print if exec succeeds)\n");
    return 0;
}
```

Observation: `exec()` replaces the process image; code after exec runs only if exec fails.

4. Program to Find Number of Processes and Their States

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>
#include <ctype.h>

int is_number(const char *str) {
    while (*str) {
        if (!isdigit(*str++)) return 0;
    }
    return 1;
}

void parse_stat(const char *path, char *state, unsigned long *utime,
unsigned long *stime) {
    FILE *file = fopen(path, "r");
    if (!file) return;
    int pid; char comm[256];
    fscanf(file, "%d %s %c", &pid, comm, state);
    for (int i=0; i<11; i++) fscanf(file, "%*s");
    fscanf(file, "%lu %lu", utime, stime);
    fclose(file);
}

void parse_status(const char *path, unsigned long *vmrss) {
    FILE *file = fopen(path, "r");
    if (!file) return;
    char line[256];
    while (fgets(line, sizeof(line), file)) {
        if (strncmp(line, "VmRSS:", 6) == 0) {
            sscanf(line+6, "%lu", vmrss);
            break;
        }
    }
    fclose(file);
}

int main() {
    DIR *proc = opendir("/proc");
    struct dirent *entry;

    printf("PID\tState\tCPU Time(Jiffies)\tVmRSS (kB)\n");
    while ((entry = readdir(proc)) != NULL) {
        if (is_number(entry->d_name)) {
            char stat_path[64], status_path[64], state = '?';
            unsigned long utime = 0, stime = 0, vmrss = 0;
            snprintf(stat_path, sizeof(stat_path), "/proc/%s/stat",
entry->d_name);
            snprintf(status_path, sizeof(status_path),
"/proc/%s/status", entry->d_name);
            parse_stat(stat_path, &state, &utime, &stime);
            parse_status(status_path, &vmrss);
            printf("%s\t%c\t%lu\t\t\t%lu\n", entry->d_name, state,
utime+stime, vmrss);
        }
    }
    closedir(proc);
    return 0;
}
```

5. Resource Assignment Validity Checker

```
#include <stdio.h>
#define N 5
#define M 3

int allocation[N][M] = {
    {1, 0, 1},
    {0, 1, 0},
    {0, 0, 1},
    {1, 1, 0},
    {0, 0, 0},
};
```

```c
void check_resource_validity() {
    int i, j, k;
    for (j = 0; j < M; j++) {
        int count = 0;
        for (i = 0; i < N; i++) {
            if (allocation[i][j]) count++;
        }
        if (count > 1)
            printf("Error: Resource %d assigned to %d processes\n", j,
count);
    }
    for (i = 0; i < N; i++) {
        for (j = 0; j < M; j++) {
            if (allocation[i][j] > 1)
                printf("Error: Process %d requests Resource %d
repeatedly\n", i, j);
        }
    }
}

int main() {
    check_resource_validity();
    return 0;
}
```

End of Lab 2 Report