Arunima Chaudhuri
23CSM1R04
M. Tech CSE - 1st Year

Advanced Operating Systems Lab - Assignment 1

ls

List files and directories in the current directory.

- **ls -l** List files and directories in long format.
- **ls -a** List all files and directories, including hidden ones.
- **ls -R** List files and directories recursively.
- **ls <dirname>** List files and directories in the specified directory.

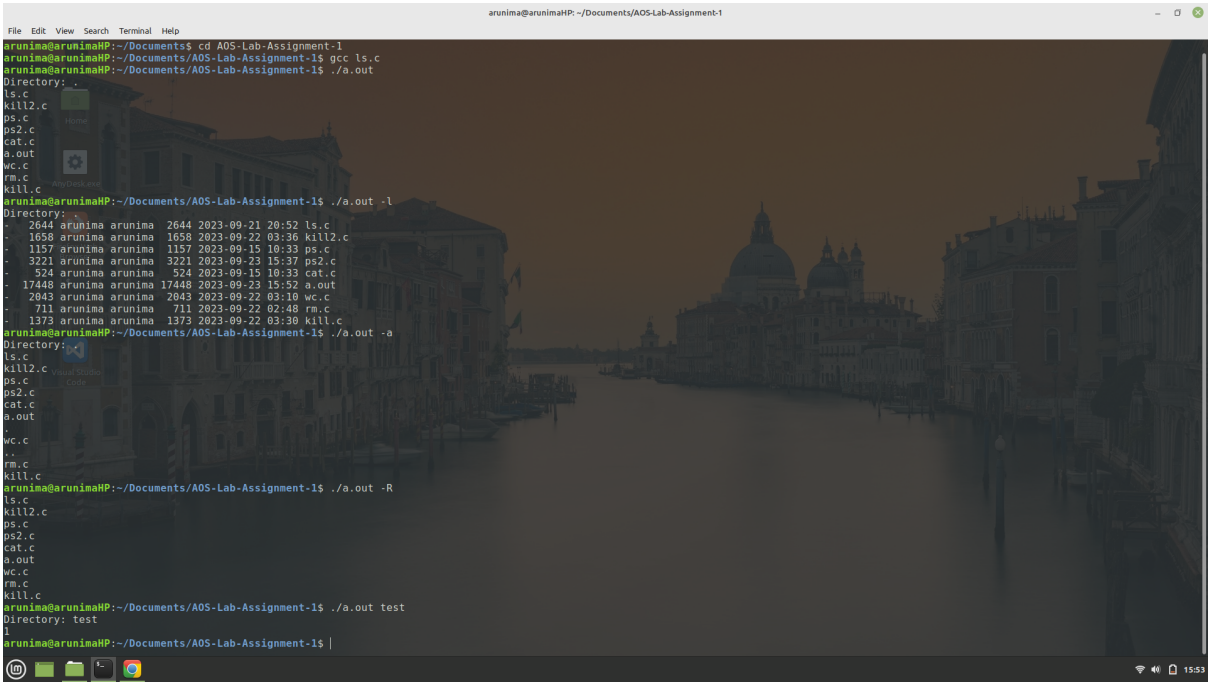
Implementation Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>

void list_directory(const char *path, int show_hidden, int long_format,
int recursive) {
    DIR *dir;
    struct dirent *entry;
    struct stat file_info;

    if ((dir = opendir(path)) == NULL) {
        perror("opendir");
        exit(1);
    }

    if (!recursive)
        printf("Directory: %s\n", path);

    while ((entry = readdir(dir)) != NULL) {
        if (!show_hidden && entry->d_name[0] == '.')
            continue;

        char full_path[1024];
        snprintf(full_path, sizeof(full_path), "%s/%s", path,
entry->d_name);

        if (lstat(full_path, &file_info) == -1) {
```

```
            perror("lstat");
            exit(1);
        }

        if (long_format) {
            struct passwd *pw = getpwuid(file_info.st_uid);
            struct group *gr = getgrgid(file_info.st_gid);
            struct tm *tm_info = localtime(&file_info.st_mtime);

            printf("%s %6ld %s %s %5ld %4d-%02d-%02d %02d:%02d %s\n",
                   (S_ISDIR(file_info.st_mode) ? "d" : "-"),
                   (long)file_info.st_size,
                   (pw ? pw->pw_name : "?"),
                   (gr ? gr->gr_name : "?"),
                   (long)file_info.st_size,
                   1900 + tm_info->tm_year,
                   tm_info->tm_mon + 1,
                   tm_info->tm_mday,
                   tm_info->tm_hour,
                   tm_info->tm_min,
                   entry->d_name);
        }
        else {
            printf("%s\n", entry->d_name);
         }

        if (recursive && S_ISDIR(file_info.st_mode) &&
strcmp(entry->d_name, ".") != 0 && strcmp(entry->d_name, "..") != 0) {
            list_directory(full_path, show_hidden, long_format,
recursive);
        }
    }

    closedir(dir);
}

int main(int argc, char *argv[]) {
    int show_hidden = 0;
    int long_format = 0;
    int recursive = 0;
    char *directory = ".";

    // Check command-line arguments
    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-a") == 0) {
            show_hidden = 1;
```

```
        } else if (strcmp(argv[i], "-l") == 0) {
            long_format = 1;
        } else if (strcmp(argv[i], "-R") == 0) {
            recursive = 1;
        } else if (argv[i][0] != '-') {
            // If not an option, assume it's a directory
            directory = argv[i];
        } else {
            fprintf(stderr, "Unknown option: %s\n", argv[i]);
            exit(1);
        }
    }

    list_directory(directory, show_hidden, long_format, recursive);

    return 0;
}
```

Output Screenshot



rm

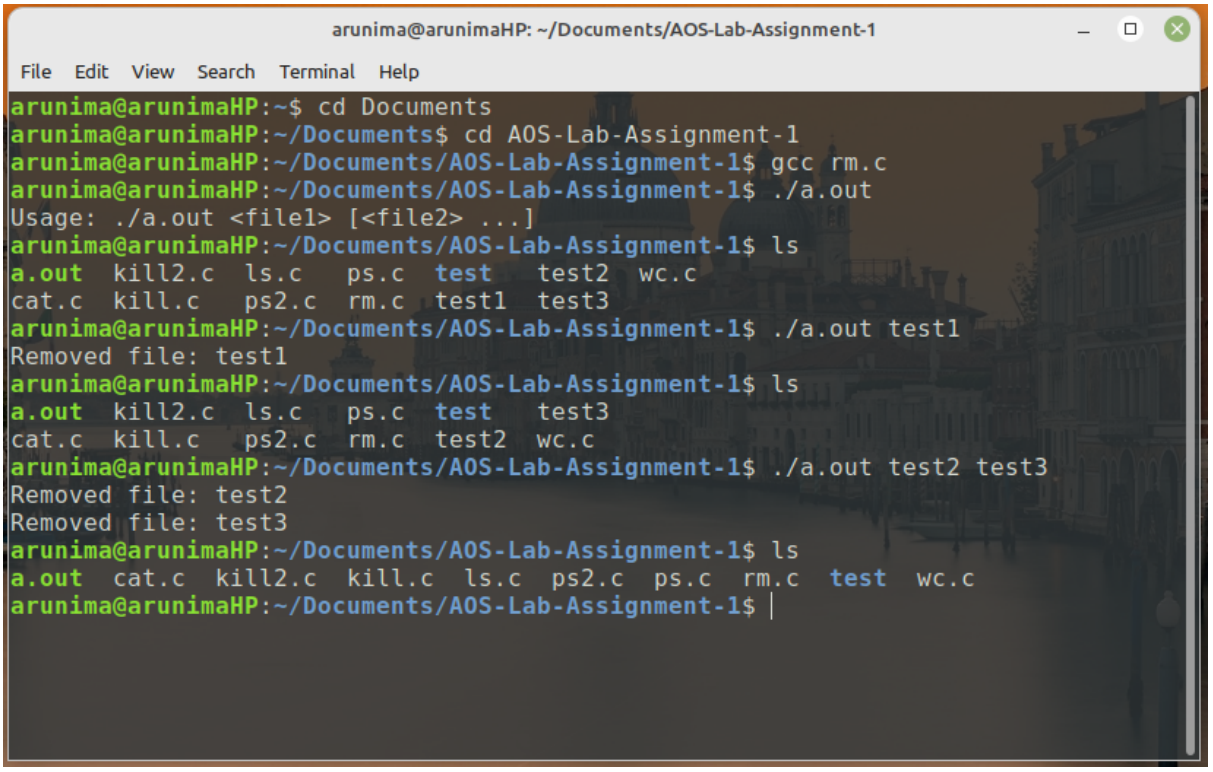
Remove files or directories.

- **rm <filename>** Remove a specific file.
- **rm -i <filename>** Interactively remove a file, prompting for confirmation.
- **rm <file1> <file2> ...** Remove multiple files at once.

Implementation Code

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <file1> [<file2> ...]\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    int c;
    if (argv[1][0] == '-') {
        printf("Do you want to remove file? (1=yes/0=no): ");
        scanf("%d", &c);
        if (c == 1) {
            remove(argv[2]);
        } else {
            printf("File not removed\n");
        }
    } else {
        for (int i = 1; i < argc; i++) {
            if (remove(argv[i]) == 0) {
                printf("Removed file: %s\n", argv[i]);
            } else {
                perror("Error deleting file");
            }
        }
    }
    return 0;
}
```

Output Screenshot



cat

Concatenate and display the content of files.

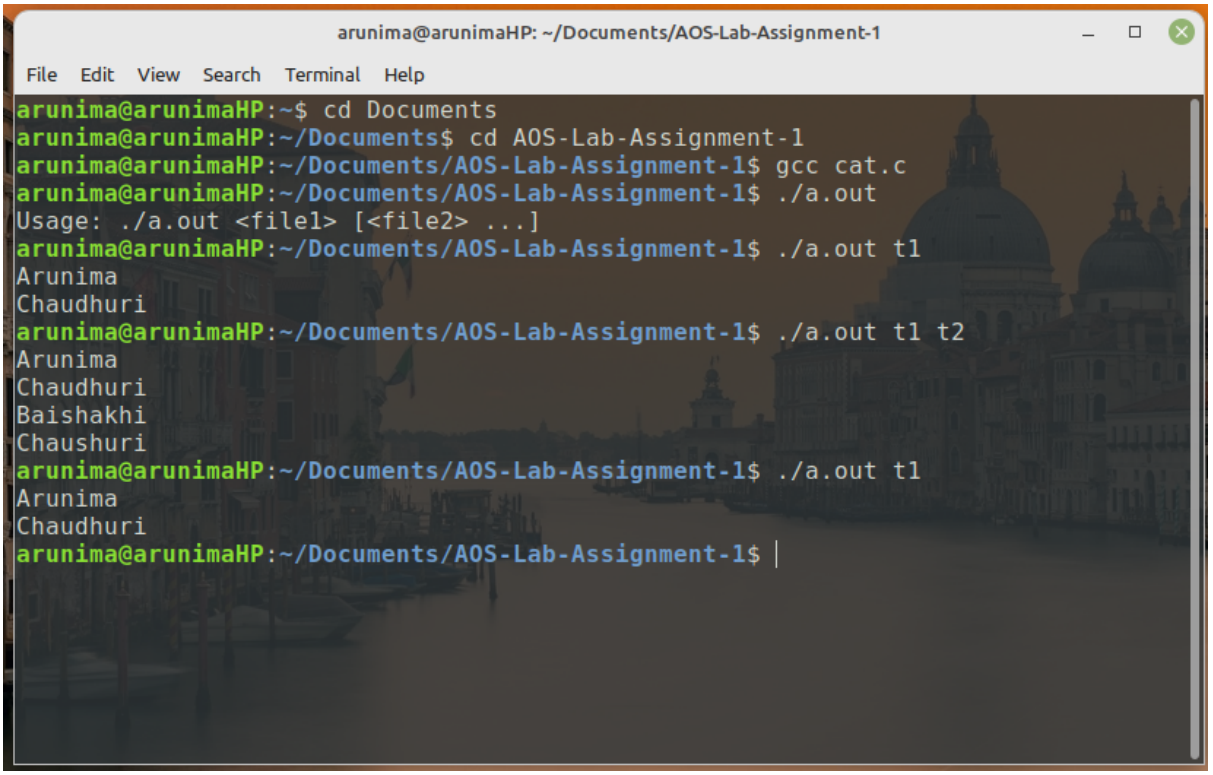
- **cat <file>** Display the content of a specific file.
- **cat <file1> <file2> ...** Concatenate and display the content of multiple files.

Implementation Code

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <file1> [<file2> ...]\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    for (int i = 1; i < argc; i++) {
        FILE *file = fopen(argv[i], "r");
        if (file == NULL) {
            perror("Error opening file");
            continue;
        }

        int ch;
        while ((ch = fgetc(file)) != EOF) {
```

```
            putchar(ch);
        }

        fclose(file);
    }

    return 0;
}
```

Output Screenshot



kill

Terminate processes by their process ID (PID).

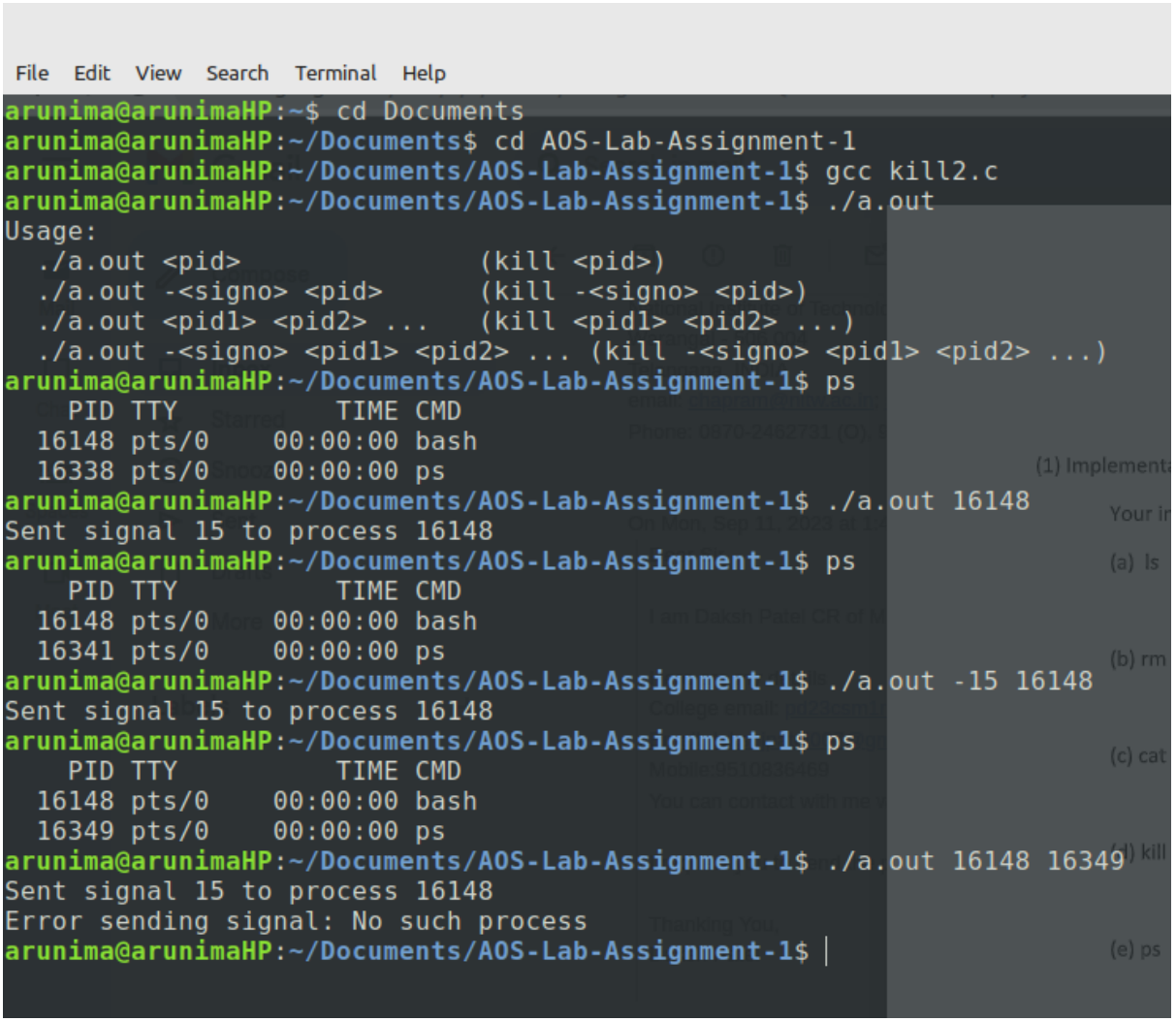
- **kill <pid>** Send a default termination signal to a process.
- **kill -<signo> <pid>** Send a specific signal to a process.
- **kill <pid1> <pid2> ...** Terminate multiple processes.

Implementation Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>

int main(int argc, char *argv[]) {
```

```
    if (argc < 2) {
        fprintf(stderr, "Usage:\n");
        fprintf(stderr, "  %s <pid>               (kill <pid>)\n",
argv[0]);
        fprintf(stderr, "  %s -<signo> <pid>      (kill -<signo>
<pid>)\n", argv[0]);
        fprintf(stderr, "  %s <pid1> <pid2> ...   (kill <pid1> <pid2>
...)\n", argv[0]);
        fprintf(stderr, "  %s -<signo> <pid1> <pid2> ... (kill -<signo>
<pid1> <pid2> ...)\n", argv[0]);
        return 1;
    }

    // Determine the signal number and PIDs based on command-line
arguments
    int signo = SIGTERM; // Default signal is SIGTERM
    int arg_start = 1;   // Index of the first argument

    if (argv[1][0] == '-') {
        // If the first argument starts with a '-', it's a signal option
        if (strcmp(argv[1], "-k") == 0) {
            // If the signal option is "-k", we treat it as a custom
signal
            signo = SIGKILL; // Change the signal to SIGKILL
            arg_start = 2;   // Start from the next argument
        } else {
            // If it's not "-k", extract the signal number
            signo = atoi(argv[1] + 1); // Extract the signal number
            arg_start = 2;             // Start from the next argument
        }
    }

    // Iterate through the PIDs and send the signal to each process
    for (int i = arg_start; i < argc; i++) {
        int pid = atoi(argv[i]);
        if (kill(pid, signo) == 0) {
            printf("Sent signal %d to process %d\n", signo, pid);
        } else {
            perror("Error sending signal");
        }
    }

    return 0;
}
```

Output Screenshot



ps

Display information about running processes.

- **ps** List information about currently running processes.
- **ps -a** Display information about all processes.
- **ps -ae** List detailed information about all processes.
- **ps -u <username>** Display processes associated with a specific user.

Implementation Code

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

void execute_ps(const char *options) {
    char command[256];
    snprintf(command, sizeof(command), "ps %s", options);

    FILE *fp = popen(command, "r");
```

```
    if (fp == NULL) {
        perror("Error executing ps command");
        exit(EXIT_FAILURE);
    }

    char buffer[1024];
    while (fgets(buffer, sizeof(buffer), fp) != NULL) {
        printf("%s", buffer);
    }

    pclose(fp);
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s [-a] [-ae] [-u <username>]\n",
argv[0]);
        exit(EXIT_FAILURE);
    }

    if (strcmp(argv[1], "-a") == 0) {
        execute_ps("a");
    } else if (strcmp(argv[1], "-ae") == 0) {
        execute_ps("ae");
    } else if (strcmp(argv[1], "-u") == 0 && argc >= 3) {
        if (strlen(argv[2]) > 0) {
            char options[256];
            snprintf(options, sizeof(options), "u %s", argv[2]);
            execute_ps(options);
        } else {
            fprintf(stderr, "Invalid username argument\n");
            exit(EXIT_FAILURE);
        }
    } else {
        execute_ps("");
    }

    return 0;
}
```

Output Screenshot

wc

Count words, lines, and characters in files.

- **wc <file1> <file2> ...** Count words, lines, and characters in specified files.
- **wc -c <file1> <file2> ...** Count only characters in specified files.
- **wc -l <file1> <file2> ...** Count only lines in specified files.
- **wc -w <file1> <file2> ...** Count only words in specified files.

Implementation Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void count_wc(const char *filename, int count_chars, int count_lines,
int count_words) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        perror("Error opening file");
        return;
    }

    int char_count = 0;
    int line_count = 0;
    int word_count = 0;
    int in_word = 0;
    int ch;

    while ((ch = fgetc(file)) != EOF) {
        char_count++;

        if (count_lines && ch == '\n') {
            line_count++;
        }

        if (count_words) {
            if (ch == ' ' || ch == '\t' || ch == '\n') {
                if (in_word) {
                    word_count++;
                    in_word = 0;
                }
            } else {
                in_word = 1;
            }
        }
    }

    // Increment word count for the last word if necessary
    if (count_words && in_word) {
        word_count++;
```

```
    }

    fclose(file);

    // Print counts based on options
    if (count_chars) {
        printf("Characters in %s: %d\n", filename, char_count);
    }
    if (count_lines) {
        printf("Lines in %s: %d\n", filename, line_count);
    }
    if (count_words) {
        printf("Words in %s: %d\n", filename, word_count);
    }
}

int main(int argc, char *argv[]) {
    int count_chars = 0;
    int count_lines = 0;
    int count_words = 0;

    if (argc < 2) {
        fprintf(stderr, "Usage: %s [-c] [-l] [-w] <file1> [<file2>
...]\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    // Check for optional options
    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-c") == 0) {
            count_chars = 1;
        } else if (strcmp(argv[i], "-l") == 0) {
            count_lines = 1;
        } else if (strcmp(argv[i], "-w") == 0) {
            count_words = 1;
        } else {
      if(i==1){
            count_chars=count_lines=count_words=1;
          count_wc(argv[i], count_chars,count_lines,count_words);
      }
      else{
            count_wc(argv[i], count_chars,count_lines,count_words);
            }
        }

      }
```

```
    return 0;
}
```

Output Screenshot