10. **Write a C program to emulate the Unix ls – l command.**

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

int main(int argc, char \*\*argv) {

    DIR \*dirop;

    struct dirent \*dired;

    // Check if the correct number of arguments is provided

    if (argc != 2) {

        printf("Invalid number of arguments. Usage: %s <directory\_name>\n", argv[0]);

        exit(1); // Exit if the number of arguments is incorrect

    }

    // Try to open the directory specified in the argument

    if ((dirop = opendir(argv[1])) == NULL) {

        printf("Cannot open directory %s\n", argv[1]);

        exit(1); // Exit if the directory cannot be opened

    }

    // Read and print directory entries

    while ((dired = readdir(dirop)) != NULL) {

        // Print inode number and filename

        printf("%10lu %s\n", dired->d\_ino, dired->d\_name);

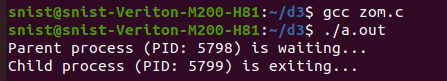
    }

    // Close the directory stream

    closedir(dirop);

    return 0;

}



5. C Programming examples using Linux operating systems.

a. wc

#include <stdio.h>

#include <ctype.h>

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

FILE \*file;

int c, lines = 0, words = 0, chars = 0;

if (argc != 2) {

fprintf(stderr, "Usage: %s <filename>\n", argv[0]);

return 1;

}

file = fopen(argv[1], "r");

if (!file) {

perror("Error opening file");

return 1;

}

while ((c = fgetc(file)) != EOF) {

chars++;

if (c == '\n') {

lines++;

}

if (isspace(c)) {

words++;

}

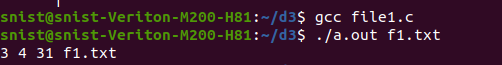
}

fclose(file);

printf("%d %d %d %s\n", lines, words, chars, argv[1]);

return 0;

}



b.Cat

File1.c

#include <stdio.h>

#include <stdlib.h>

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

    FILE \*file;

    char ch;

    // Check if at least one file name is provided

    if (argc < 2) {

        printf("Usage: %s <file1> <file2> ...\n", argv[0]);

        return 1;

    }

    // Loop through each file provided in the command line

    for (int i = 1; i < argc; i++) {

        file = fopen(argv[i], "r");  // Open the file in read mode

        if (file == NULL) {

            perror("Error opening file");

            continue;  // Move to the next file

        }

        // Read the file character by character and print it to the console

        while ((ch = fgetc(file)) != EOF) {

            putchar(ch);  // Print the character to stdout

        }

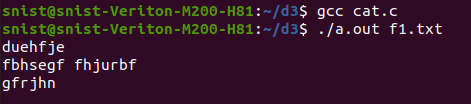
        fclose(file);  // Close the file after reading

        printf("\n");   // Print a newline after each file's content

    }

    return 0;

}



c.cp

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#define BUF\_SIZE 1024

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

    int src, dest, n;

    char buf[BUF\_SIZE];

    if (argc != 3) {

        write(2, "Usage: cp <source> <destination>\n", 33);

        return 1;

    }

    src = open(argv[1], O\_RDONLY);

    if (src == -1) return 1;

    dest = open(argv[2], O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

    if (dest == -1) return 1;

    while ((n = read(src, buf, BUF\_SIZE)) > 0)

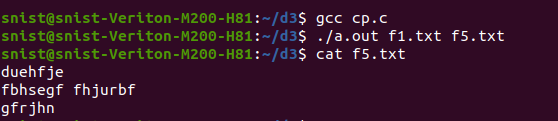
        write(dest, buf, n);

    close(src);

    close(dest);

    return 0;

}



**9.Implement in C the following Unix commands using System calls**

* 1. **rename**

#include <stdio.h>

#include <stdlib.h>

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

    // Check if correct number of arguments is provided

    if (argc != 3) {

        fprintf(stderr, "Usage: %s <old\_name> <new\_name>\n", argv[0]);

        return 1;

    }

    // Rename the file using the rename() system call

    if (rename(argv[1], argv[2]) == 0) {

        printf("File renamed successfully: %s -> %s\n", argv[1], argv[2]);

    } else {

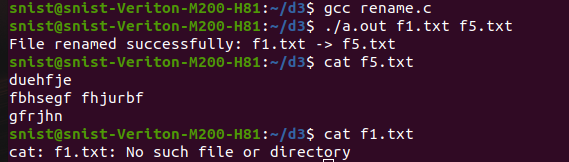
        perror("Error renaming file");

        return 1;

    }

    return 0;

}



b.link

#include <stdio.h>

#include <unistd.h>

#include <errno.h>

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

    if (argc != 3) {

        // Check if the correct number of arguments is provided

        fprintf(stderr, "Usage: %s <source\_file> <link\_name>\n", argv[0]);

        return 1;

    }

    char \*source\_file = argv[1];

    char \*link\_name = argv[2];

    // Use the link system call to create a hard link

    if (link(source\_file, link\_name) == -1) {

        // If the link creation fails, print an error

        perror("Error creating link");

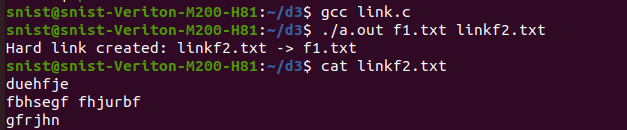
        return 1;

    }

    printf("Hard link created: %s -> %s\n", link\_name, source\_file);

    return 0;

}



10. **Write a C program to emulate the Unix ls – l command.**

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

int main(int argc, char \*\*argv) {

    DIR \*dirop;

    struct dirent \*dired;

    // Check if the correct number of arguments is provided

    if (argc != 2) {

        printf("Invalid number of arguments. Usage: %s <directory\_name>\n", argv[0]);

        exit(1); // Exit if the number of arguments is incorrect

    }

    // Try to open the directory specified in the argument

    if ((dirop = opendir(argv[1])) == NULL) {

        printf("Cannot open directory %s\n", argv[1]);

        exit(1); // Exit if the directory cannot be opened

    }

    // Read and print directory entries

    while ((dired = readdir(dirop)) != NULL) {

        // Print inode number and filename

        printf("%10lu %s\n", dired->d\_ino, dired->d\_name);

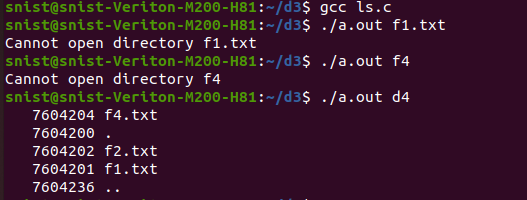
    }

    // Close the directory stream

    closedir(dirop);

    return 0;

}



11.**Write a C program on zombie process**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

    pid\_t pid;

    // Create a child process

    pid = fork();

    if (pid < 0) {

        // If fork() fails

        perror("Fork failed");

        return 1;

    }

    else if (pid == 0) {

        // Child process

        printf("Child process (PID: %d) is exiting...\n", getpid());

        \_exit(0);  // Child exits without the parent collecting the exit status

    }

    else {

        // Parent process

        printf("Parent process (PID: %d) is waiting...\n", getpid());

        // Parent does not call wait() immediately, so the child becomes a zombie

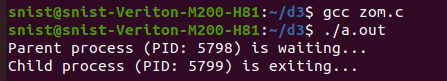
        sleep(10);  // Sleep for 10 seconds to allow the child to become a zombie

        printf("Parent process (PID: %d) exiting...\n", getpid());

    }

    return 0;

}



12. Write a C program that illustrates the following. a) Creating a message queue. b) Writing to a message queue. c) Reading from a message queue.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

// Define the structure for the message queue

struct msgbuf {

    long mtype;      // Message type (must be a positive integer)

    char mtext[100]; // Message text (content)

};

int main() {

    int msqid;          // Message queue ID

    int len, ret;       // Length of the message and return values

    struct msgbuf msgsend = {0, "\0"}, msgrecv; // Message buffers for sending and receiving

    // Step 1: Create a message queue (with key 1234)

    msqid = msgget((key\_t)1234, IPC\_CREAT | 0666);  // Create the message queue with read-write permissions for all users

    if (msqid == -1) {

        perror("msgget:");

        exit(1);

    }

    printf("Message Queue Created with ID: %d\n", msqid);

    // Step 2: Write a message to the message queue

    printf("Enter message type (positive integer): ");

    if (scanf("%ld", &msgsend.mtype) != 1) {

        fprintf(stderr, "Invalid message type input.\n");

        exit(1);

    }

    printf("Enter message text: ");

    if (scanf("%s", msgsend.mtext) != 1) {

        fprintf(stderr, "Invalid message text input.\n");

        exit(1);

    }

    len = strlen(msgsend.mtext);  // Calculate the length of the message text

    ret = msgsnd(msqid, &msgsend, len, 0);  // Send the message to the queue

    if (ret == -1) {

        perror("msgsnd:");

        exit(1);

    }

    printf("Message Sent: %s\n", msgsend.mtext);

    // Step 3: Read the message from the message queue

    ret = msgrcv(msqid, &msgrecv, sizeof(msgrecv.mtext), msgsend.mtype, 0);  // Receive the message with the same type

    if (ret == -1) {

        perror("msgrcv:");

        exit(1);

    }

    printf("Message Received: %s\n", msgrecv.mtext);

    // Step 4: Clean up by removing the message queue (optional)

    ret = msgctl(msqid, IPC\_RMID, NULL);  // Remove the message queue

    if (ret == -1) {

        perror("msgctl:");

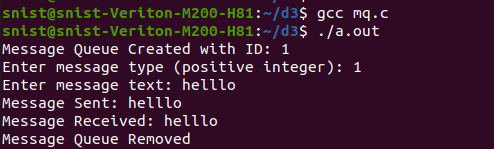
        exit(1);

    }

    printf("Message Queue Removed\n");

    return 0;

}



13. **Write a C program that illustrates file locking using semaphores.**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <unistd.h>

// Semaphore union for operations

union semun {

    int val;

    struct semid\_ds \*buf;

    unsigned short \*array;

    struct seminfo \*\_\_buf;

};

#define SEM\_KEY 1234  // Semaphore key

// Locking function using semaphore

void lock\_file(int semid) {

    struct sembuf sb = {0, -1, 0};  // P (wait) operation

    if (semop(semid, &sb, 1) == -1) {

        perror("semop - lock");

        exit(1);

    }

    printf("File locked.\n");

}

// Unlocking function using semaphore

void unlock\_file(int semid) {

    struct sembuf sb = {0, 1, 0};   // V (signal) operation

    if (semop(semid, &sb, 1) == -1) {

        perror("semop - unlock");

        exit(1);

    }

    printf("File unlocked.\n");

}

int main() {

    int semid;

    union semun sem\_union;

    // Create or get the semaphore set

    semid = semget(SEM\_KEY, 1, IPC\_CREAT | 0666);

    if (semid == -1) {

        perror("semget");

        exit(1);

    }

    // Initialize the semaphore to 1 (unlocked)

    sem\_union.val = 1;

    if (semctl(semid, 0, SETVAL, sem\_union) == -1) {

        perror("semctl");

        exit(1);

    }

    // Lock and unlock the file (simulated by print and sleep)

    lock\_file(semid);

    sleep(3);  // Simulate file access

    unlock\_file(semid);

    // Clean up semaphore

    if (semctl(semid, 0, IPC\_RMID, sem\_union) == -1) {

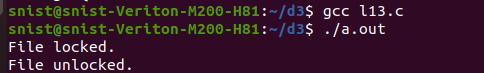
        perror("semctl - IPC\_RMID");

        exit(1);

    }

    return 0;

}



14. write a C program to implement record locking.

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <errno.h>

#define FILENAME "record\_file.txt"

int main() {

    int fd;

    struct flock lock;

    // Open the file

    fd = open(FILENAME, O\_RDWR | O\_CREAT, 0666);

    if (fd == -1) {

        perror("Failed to open file");

        exit(1);

    }

    // Initialize the lock structure

    lock.l\_type = F\_WRLCK;  // Set the lock type to write lock

    lock.l\_whence = SEEK\_SET;  // Lock is from the beginning of the file

    lock.l\_start = 0;   // Start of the file (you can set it to a specific record offset)

    lock.l\_len = 100;   // Length of the record to lock (here locking 100 bytes)

    // Try to acquire the lock

    if (fcntl(fd, F\_SETLK, &lock) == -1) {

        if (errno == EACCES || errno == EAGAIN) {

            printf("Record is already locked by another process.\n");

        } else {

            perror("Failed to lock record");

        }

        close(fd);

        exit(1);

    }

    printf("Record locked successfully.\n");

    // Simulate file access (sleep for 5 seconds)

    sleep(5);

    // Release the lock

    lock.l\_type = F\_UNLCK;  // Change the lock type to unlock

    if (fcntl(fd, F\_SETLK, &lock) == -1) {

        perror("Failed to unlock record");

        close(fd);

        exit(1);

    }

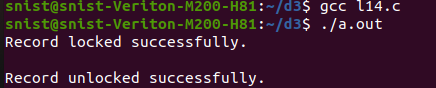
    printf("Record unlocked successfully.\n");

    // Close the file

    close(fd);

    return 0;

}



15.  Write a C program that illustrates two processes communicating using PIPE

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <sys/wait.h>  
#include <time.h>  
  
int main(int argc, char\* argv[]) {  
    int p1[2];  
    int p2[2];  
     
    if (pipe(p1) == -1) {  
        return 1;  
    }  
    if (pipe(p2) == -1) {  
        return 2;  
    }  
    int pid = fork();  
    if (pid == -1) {  
        return 3;  
    }  
     
    if (pid == 0) {  
        // Child process  
        close(p1[1]);  
        close(p2[0]);  
         
        int x;  
        if (read(p1[0], &x, sizeof(int)) == -1) {  
            return 3;  
        }  
        printf("Received %d\n", x);  
         
        x \*= 4;  
         
        if (write(p2[1], &x, sizeof(int)) == -1) {  
            return 4;  
        }  
        printf("Wrote %d\n", x);  
        close(p1[0]);  
        close(p2[1]);  
    } else {  
        // Parent process  
        close(p1[0]);  
        close(p2[1]);  
         
        srand(time(NULL));  
        int y = rand() % 10;  
         
         
        if (write(p1[1], &y, sizeof(y)) == -1) {  
            return 5;  
        }  
        printf("Wrote %d\n", y);  
        if (read(p2[0], &y, sizeof(y)) == -1) {  
            return 6;  
        }  
        printf("Result is %d\n", y);  
         
        close(p1[1]);  
        close(p2[0]);  
        wait(NULL);  
    }  
     
    return 0;  
}

