**OS LAB 6**

**22F-3168**

**Mariam Fatima**

**Task 1:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main()

{

int num1, num2;

// Creating pipes for communication

int pipe1[2]; // C1 (add and sub)

int pipe2[2]; // C2 (mul)

int pipe3[2]; // C3 (div)

int pipe4[2]; // C4 (output)

pipe(pipe1);

pipe(pipe2);

pipe(pipe3);

pipe(pipe4);

//child process for add and sub

pid\_t c1 = fork();

if (c1 == 0)

{

// Child process C1

close(pipe1[1]);

int add\_result, sub\_result;

// Read numbers from the parent process

read(pipe1[0], &num1, sizeof(int));

read(pipe1[0], &num2, sizeof(int));

close(pipe1[0]);

//add and sub

add\_result = num1 + num2;

sub\_result = num1 - num2;

// Creating child process C4

pid\_t c4 = fork();

if (c4 == 0) {

// Child process C4

close(pipe4[1]);

int c1\_add, c1\_sub, c2\_mul, c3\_div;

// Read results from C1

read(pipe4[0], &c1\_add, sizeof(int));

read(pipe4[0], &c1\_sub, sizeof(int));

// Read results from C2

read(pipe4[0], &c2\_mul, sizeof(int));

// Read results from C3

read(pipe4[0], &c3\_div, sizeof(int));

close(pipe4[0]); // Close reading end after reading data

// Display the results

printf("Sum: %d\n", c1\_add);

printf("Difference: %d\n", c1\_sub);

printf("Product: %d\n", c2\_mul);

printf("Quotient: %d\n", c3\_div);

exit(0);

} else {

// Send results from C1 to C4

close(pipe4[0]); // Close unused read end of pipe4

write(pipe4[1], &add\_result, sizeof(int));

write(pipe4[1], &sub\_result, sizeof(int));

close(pipe4[1]); // Close write end after sending data

wait(NULL);

}

exit(0);

}

// Creating child process for mul

pid\_t c2 = fork();

if (c2 == 0)

{

// Child process C2

close(pipe2[1]);

int mul\_result;

// Read numbers from the parent process

read(pipe2[0], &num1, sizeof(int));

read(pipe2[0], &num2, sizeof(int));

close(pipe2[0]);

// Perform mul

mul\_result = num1 \* num2;

// Send result to C4

close(pipe4[0]); // Close unused read end of pipe4

write(pipe4[1], &mul\_result, sizeof(int));

close(pipe4[1]); // Close write end after sending data

exit(0);

}

// Create child process for division

pid\_t c3 = fork();

if (c3 == 0) {

// Child process C3

close(pipe3[1]);

int div\_result;

// Read numbers from the parent process

read(pipe3[0], &num1, sizeof(int));

read(pipe3[0], &num2, sizeof(int));

close(pipe3[0]);

// Perform division

if (num2 != 0) {

div\_result = num1 / num2;

} else {

div\_result = 0; // Handle division by zero

}

// Send result to C4

close(pipe4[0]);

write(pipe4[1], &div\_result, sizeof(int));

close(pipe4[1]);

exit(0);

}

// Get user input

printf("Enter two numbers for addition and subtraction: ");

scanf("%d %d", &num1, &num2);

// Send numbers to C1

close(pipe1[0]);

write(pipe1[1], &num1, sizeof(int));

write(pipe1[1], &num2, sizeof(int));

close(pipe1[1]);

// Get user input for the second set of numbers

printf("Enter two numbers for multiplication and division: ");

scanf("%d %d", &num1, &num2);

// Send numbers to C2 and C3

close(pipe2[0]);

write(pipe2[1], &num1, sizeof(int));

write(pipe2[1], &num2, sizeof(int));

close(pipe2[1]);

close(pipe3[0]);

write(pipe3[1], &num1, sizeof(int));

write(pipe3[1], &num2, sizeof(int));

close(pipe3[1]);

// Wait for all children to finish

wait(NULL);

wait(NULL);

wait(NULL);

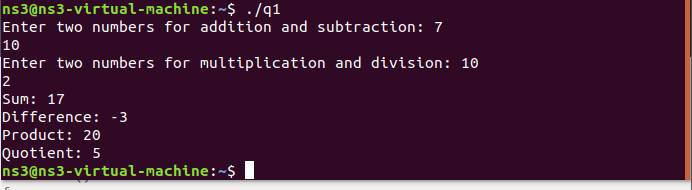
// Close pipes

close(pipe4[0]);

close(pipe4[1]);

return 0;

}



**Task 2:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/wait.h>

#define BUFFER\_SIZE 1024

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

{

    if (argc != 2) {

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

        return 1;

    }

    int pipefd[2];

    if (pipe(pipefd) == -1) {

        perror("pipe");

        return 1;

    }

    pid\_t pid = fork();

    if (pid < 0) {

        perror("fork");

        return 1;

    }

    if (pid == 0) {

        close(pipefd[0]);

        dup2(pipefd[1], STDOUT\_FILENO);

        close(pipefd[1]);

        execl(argv[1], argv[1], NULL);

        perror("execl"); // If execl fails

        exit(1);

    } else {

        close(pipefd[1]); // Close unused write end

        FILE \*file = fopen("result.txt", "w");

        if (!file) {

            perror("fopen");

            return 1;

        }

        char buffer[BUFFER\_SIZE];

        ssize\_t bytesRead;

        size\_t totalBytes = 0;

        while ((bytesRead = read(pipefd[0], buffer, sizeof(buffer))) > 0) {

            fwrite(buffer, 1, bytesRead, file);

            totalBytes += bytesRead;

        }

        close(pipefd[0]);

        fclose(file);

        wait(NULL);

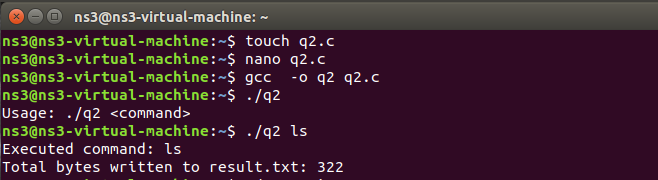
        printf("Executed command: %s\n", argv[1]);

        printf("Total bytes written to result.txt: %zu\n", totalBytes);

    }

    return 0;

}



**Task 3:**

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

#include <string.h>

#define FIFO\_NAME "my\_fifo"

int main() {

int fd;

char buffer[100];

if (mkfifo(FIFO\_NAME, 0666) == -1) {

perror("mkfifo");

exit(EXIT\_FAILURE);

}

fd = open(FIFO\_NAME, O\_RDONLY);

if (fd == -1) {

perror("open");

exit(EXIT\_FAILURE);

}

while (1) {

ssize\_t bytesRead = read(fd, buffer, sizeof(buffer) - 1);

if (bytesRead > 0) {

buffer[bytesRead] = '\0';

printf("Received: %s", buffer);

if (strcmp(buffer, "quit\n") == 0) {

break;

}

} else if (bytesRead == 0) {

break;

} else {

perror("read");

break;

}

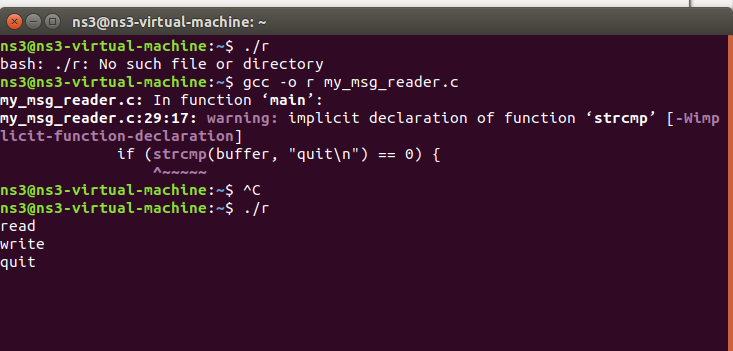
}

close(fd);

unlink(FIFO\_NAME);

return 0;

}



**Writter code**  
#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#define FIFO\_NAME "my\_fifo"

int main() {

int fd;

char buffer[100];

fd = open(FIFO\_NAME, O\_WRONLY);

while (1) {

printf("Enter message: ");

if (fgets(buffer, sizeof(buffer), stdin) == NULL) {

perror("fgets");

break;

}

if (write(fd, buffer, sizeof(buffer)) == -1) {

perror("write");

break;

}

if (strcmp(buffer, "quit\n") == 0) {

break;

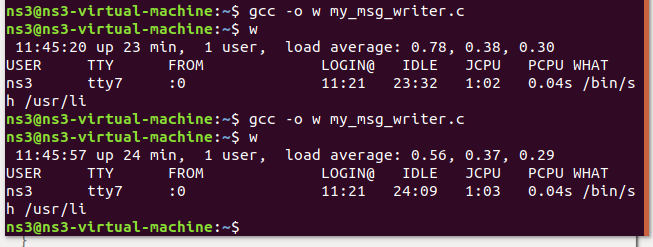
}

}

close(fd);

return 0;

}



**Task 4:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/shm.h>

#include <string.h>

#define SHM\_SIZE 256 // Size of shared memory segment

struct shared\_memory {

char str1[50];

char str2[50];

};

int main() {

int shmid;

struct shared\_memory \*shm;

// Create a shared memory segment

shmid = shmget(IPC\_PRIVATE, SHM\_SIZE, IPC\_CREAT | 0666);

if (shmid < 0) {

perror("shmget");

exit(1);

}

// Attach the shared memory segment

shm = (struct shared\_memory \*)shmat(shmid, NULL, 0);

if (shm == (void \*)-1) {

perror("shmat");

exit(1);

}

// Fork the first child process

pid\_t pid1 = fork();

if (pid1 == 0) {

// Child process 1

strcpy(shm->str1, "Welcome");

exit(0);

}

// Fork the 2nd child process

pid\_t pid2 = fork();

if (pid2 == 0) {

// Child process 2

sprintf(shm->str1, "%d", 42);

strcpy(shm->str2, "to the class");

exit(0);

}

// Fork the third child process

pid\_t pid3 = fork();

if (pid3 == 0) {

// Child process 3

sprintf(shm->str1, "%d", 1001);

strcpy(shm->str2, "student ID");

exit(0);

}

wait(NULL);

wait(NULL);

wait(NULL);

// Print the contents of shared memory

printf("Contents of shared memory:\n");

printf("String 1: %s\n", shm->str1);

printf("String 2: %s\n", shm->str2);

if (shmdt(shm) == -1) {

perror("shmdt");

exit(1);

}

if (shmctl(shmid, IPC\_RMID, NULL) == -1) {

perror("shmctl");

exit(1);

}

return 0;

}

