**ASSIGNMENT – 1**

i) **Write a program to create two processes. First process takes a string and passes it to second process through a pipe. The second process concatenates the received string with another string without using string function and sends it back to the first process for printing.**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/wait.h>

#define BUFFER\_SIZE 256

void concatenate\_without\_string\_functions(char \*dest, const char \*src) {

// Find the end of the dest string

while (\*dest) {

dest++;

}

// Copy the src string to the end of dest

while (\*src) {

\*dest++ = \*src++;

}

// Null-terminate the result

\*dest = '\0';

}

int main() {

int pipe1[2], pipe2[2];

pid\_t pid;

char input\_string[BUFFER\_SIZE];

char received\_string[BUFFER\_SIZE];

char final\_string[BUFFER\_SIZE] = " - concatenated string";

// Create the first pipe

if (pipe(pipe1) == -1) {

perror("pipe1");

exit(EXIT\_FAILURE);

}

// Create the second pipe

if (pipe(pipe2) == -1) {

perror("pipe2");

exit(EXIT\_FAILURE);

}

// Fork the process

pid = fork();

if (pid == -1) {

perror("fork");

exit(EXIT\_FAILURE);

}

if (pid == 0) {

// Child process (second process)

close(pipe1[1]); // Close write end of pipe1

close(pipe2[0]); // Close read end of pipe2

// Read the string from pipe1

read(pipe1[0], received\_string, BUFFER\_SIZE);

close(pipe1[0]);

// Concatenate the received string with final\_string

concatenate\_without\_string\_functions(received\_string, final\_string);

// Send the concatenated string to pipe2

write(pipe2[1], received\_string, strlen(received\_string) + 1);

close(pipe2[1]);

exit(EXIT\_SUCCESS);

} else {

// Parent process (first process)

close(pipe1[0]); // Close read end of pipe1

close(pipe2[1]); // Close write end of pipe2

// Get the input string from the user

printf("Enter a string: ");

fgets(input\_string, BUFFER\_SIZE, stdin);

// Remove newline character if present

size\_t len = strlen(input\_string);

if (len > 0 && input\_string[len - 1] == '\n') {

input\_string[len - 1] = '\0';

}

// Send the input string to the second process through pipe1

write(pipe1[1], input\_string, strlen(input\_string) + 1);

close(pipe1[1]);

// Wait for the child process to finish

wait(NULL);

// Read the result from pipe2

read(pipe2[0], received\_string, BUFFER\_SIZE);

close(pipe2[0]);

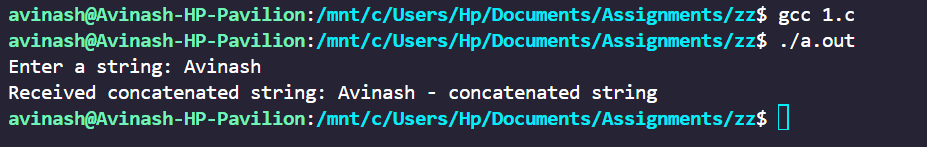
// Print the final concatenated string

printf("Received concatenated string: %s\n", received\_string);

exit(EXIT\_SUCCESS); }}



//output



**ii) Write a program in which the parent process sends two matrices to its child process through pipe and the child process returns the sum of the matrices to the parent through a pipe. The parent should print the result.**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#define MATRIX\_SIZE 3

void print\_matrix(int matrix[MATRIX\_SIZE][MATRIX\_SIZE]) {

for (int i = 0; i < MATRIX\_SIZE; i++) {

for (int j = 0; j < MATRIX\_SIZE; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

void sum\_matrices(int matrix1[MATRIX\_SIZE][MATRIX\_SIZE], int matrix2[MATRIX\_SIZE][MATRIX\_SIZE], int result[MATRIX\_SIZE][MATRIX\_SIZE]) {

for (int i = 0; i < MATRIX\_SIZE; i++) {

for (int j = 0; j < MATRIX\_SIZE; j++) {

result[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

}

int main() {

int pipe1[2], pipe2[2];

pid\_t pid;

int matrix1[MATRIX\_SIZE][MATRIX\_SIZE];

int matrix2[MATRIX\_SIZE][MATRIX\_SIZE];

int result\_matrix[MATRIX\_SIZE][MATRIX\_SIZE];

// Initialize matrices

printf("Enter Matrix1:\n");

for (int i = 0; i < MATRIX\_SIZE; i++) {

for (int j = 0; j < MATRIX\_SIZE; j++) {

scanf("%d", &matrix1[i][j]);

}

}

printf("Enter Matrix2:\n");

for (int i = 0; i < MATRIX\_SIZE; i++) {

for (int j = 0; j < MATRIX\_SIZE; j++) {

scanf("%d", &matrix2[i][j]);

}

}

// Create pipes

if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {

perror("pipe");

exit(EXIT\_FAILURE);

}

// Fork the process

pid = fork();

if (pid == -1) {

perror("fork");

exit(EXIT\_FAILURE);

}

if (pid == 0) {

// Child process

close(pipe1[1]); // Close write end of pipe1

close(pipe2[0]); // Close read end of pipe2

int recv\_matrix1[MATRIX\_SIZE][MATRIX\_SIZE];

int recv\_matrix2[MATRIX\_SIZE][MATRIX\_SIZE];

// Read matrices from pipe1

read(pipe1[0], recv\_matrix1, sizeof(recv\_matrix1));

read(pipe1[0], recv\_matrix2, sizeof(recv\_matrix2));

close(pipe1[0]);

// Compute the sum of matrices

sum\_matrices(recv\_matrix1, recv\_matrix2, result\_matrix);

// Send the result matrix back to parent via pipe2

write(pipe2[1], result\_matrix, sizeof(result\_matrix));

close(pipe2[1]);

exit(EXIT\_SUCCESS);

} else {

// Parent process

close(pipe1[0]); // Close read end of pipe1

close(pipe2[1]); // Close write end of pipe2

// Send matrices to the child process

write(pipe1[1], matrix1, sizeof(matrix1));

write(pipe1[1], matrix2, sizeof(matrix2));

close(pipe1[1]);

// Wait for child process to complete

wait(NULL);

// Read the result matrix from child process

read(pipe2[0], result\_matrix, sizeof(result\_matrix));

close(pipe2[0]);

// Print the result matrix

printf("Sum of matrices:\n");

print\_matrix(result\_matrix);

exit(EXIT\_SUCCESS);

}

}

//output

