Practical-3

**Aim:-** Implement TWO-WAY *Inter-process communication* using "pipe()" system call between:

1. Child and parent process
2. Two child processes

This communication must continue until a specific key is pressed or any process sends a STOP message.

# Theory:-

Inter-process communication using pipes is a common method for allowing two processes to communicate with each other in a Unix-like operating system. A pipe is a unidirectional communication channel that allows data to ﬂow from one process to another. To establish two-way IPC using pipes, you typically need to create two pipes: one for each direction of communication.

# Code:-

* 1. Child and parent process

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

#define READ\_END 0

#define WRITE\_END 1

int main() {

int pipe\_fd1[2], pipe\_fd2[2]; pid\_t child\_pid;

char buffer[100]; int bytes\_read;

if (pipe(pipe\_fd1) == -1 || pipe(pipe\_fd2) == -1) { perror("Pipe creation failed"); exit(EXIT\_FAILURE);

}

child\_pid = fork();

if (child\_pid == -1) { perror("Fork failed"); exit(EXIT\_FAILURE);

}

if (child\_pid == 0) { close(pipe\_fd1[READ\_END]); close(pipe\_fd2[WRITE\_END]);

while (1) {

char message[100];

printf("Enter a message for parent: "); fgets(message, sizeof(message), stdin);

write(pipe\_fd1[WRITE\_END], message, strlen(message) + 1);

bytes\_read = read(pipe\_fd2[READ\_END], buffer, sizeof(buffer)); if (bytes\_read <= 0) {

break;

}

if (strcmp(buffer, "STOP\n") == 0) { printf("Communication break !!! \n"); break;

} else {

printf("Received message from parent: %s", buffer);

}

}

close(pipe\_fd1[WRITE\_END]); close(pipe\_fd2[READ\_END]); exit(EXIT\_SUCCESS);

} else { close(pipe\_fd1[WRITE\_END]); close(pipe\_fd2[READ\_END]);

while (1) {

bytes\_read = read(pipe\_fd1[READ\_END], buffer, sizeof(buffer)); if (bytes\_read <= 0) {

break;

}

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

printf("Communication break !!! \n"); break;

} else {

printf("Received message from child: %s", buffer);

}

char message[100];

printf("Enter a message for child: "); fgets(message, sizeof(message), stdin);

write(pipe\_fd2[WRITE\_END], message, strlen(message) + 1);

if (strcmp(message, "STOP\n") == 0) { printf("Communication break !!! \n"); break;

}

}

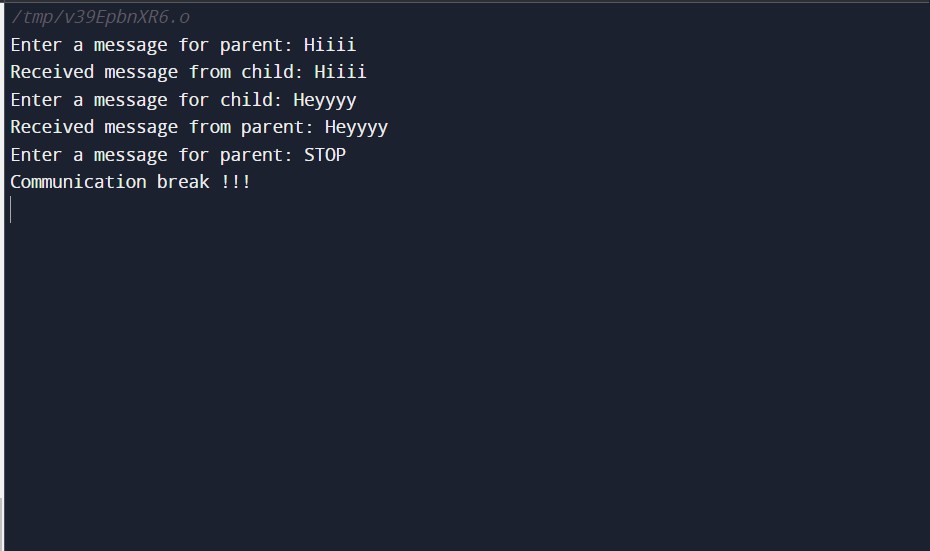
close(pipe\_fd1[READ\_END]); close(pipe\_fd2[WRITE\_END]);

}

return 0;

}

# Output:-



* 1. Two child processes

#include <signal.h> #include <stdbool.h> #include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h>

#define READ\_END 0

#define WRITE\_END 1

int main() {

int pipe\_child1\_to\_child2[2]; int pipe\_child2\_to\_child1[2]; pid\_t child1\_pid, child2\_pid; char buffer[100];

int bytes\_read;

bool stop\_communication = false;

if (pipe(pipe\_child1\_to\_child2) == -1 || pipe(pipe\_child2\_to\_child1) == -1) { perror("Pipe creation failed");

exit(EXIT\_FAILURE);

}

child1\_pid = fork();

if (child1\_pid == -1) { perror("Fork failed"); exit(EXIT\_FAILURE);

}

if (child1\_pid == 0) { // Child process 1 close(pipe\_child1\_to\_child2[READ\_END]); close(pipe\_child2\_to\_child1[WRITE\_END]);

while (!stop\_communication) { char message[100];

printf("Enter a message for Child 2 (or type 'STOP' to end "

"communication): "); fgets(message, sizeof(message), stdin);

write(pipe\_child1\_to\_child2[WRITE\_END], message, strlen(message) + 1); if (strcmp(message, "STOP\n") == 0) {

stop\_communication = true;

break;

}

bytes\_read =

read(pipe\_child2\_to\_child1[READ\_END], buffer, sizeof(buffer)); if (bytes\_read > 0) {

printf("Message from Child 2: %s", buffer);

}

if (strcmp(buffer, "STOP\n") == 0) { stop\_communication = true; printf("Communication break !!! \n"); break;

}

}

close(pipe\_child1\_to\_child2[WRITE\_END]); close(pipe\_child2\_to\_child1[READ\_END]); exit(EXIT\_SUCCESS);

} else { close(pipe\_child1\_to\_child2[WRITE\_END]); close(pipe\_child2\_to\_child1[READ\_END]);

child2\_pid = fork();

if (child2\_pid == -1) { perror("Fork failed"); exit(EXIT\_FAILURE);

}

if (child2\_pid == 0) { // Child process 2 close(pipe\_child1\_to\_child2[WRITE\_END]); close(pipe\_child2\_to\_child1[READ\_END]);

while (!stop\_communication) { char message[100];

bytes\_read =

read(pipe\_child1\_to\_child2[READ\_END], buffer, sizeof(buffer)); if (bytes\_read > 0) {

printf("Message from Child 1: %s", buffer);

}

if (strcmp(buffer, "STOP\n") == 0) { stop\_communication = true; printf("Communication break !!! \n"); break;

}

printf("Enter a message for Child 1 (or type 'STOP' to end " "communication): ");

fgets(message, sizeof(message), stdin); write(pipe\_child2\_to\_child1[WRITE\_END], message, strlen(message) + 1);

if (strcmp(message, "STOP\n") == 0) { stop\_communication = true;

break;

}

}

close(pipe\_child1\_to\_child2[READ\_END]); close(pipe\_child2\_to\_child1[WRITE\_END]); exit(EXIT\_SUCCESS);

} else { // Parent process close(pipe\_child1\_to\_child2[READ\_END]); close(pipe\_child2\_to\_child1[WRITE\_END]);

wait(NULL); wait(NULL);

close(pipe\_child1\_to\_child2[WRITE\_END]); close(pipe\_child2\_to\_child1[READ\_END]);

}

}

return 0;

}

# Output:-

