**Practical - 3**

**Aim: Implement IPC with sockets**

**Code:**

***client.c***

#include <sys/socket.h>

#include <sys/types.h>

#include <sys/un.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#define BACKLOG 2

#define BUFFER\_SIZE 50

#define SOCKET\_NAME "/tmp/utsav.socket"

int main()

{

printf("Client started...\n");

struct sockaddr\_un addr;

int i;

int ret;

int client\_socket;

char buffer[BUFFER\_SIZE];

// Create socket

client\_socket = socket(AF\_UNIX, SOCK\_SEQPACKET, 0);

if(client\_socket < 0) {

perror("socket");

exit(EXIT\_FAILURE);

}

// Setup socket address family

memset(&addr, 0, sizeof(struct sockaddr\_un));

// Connect socket to socket address

addr.sun\_family = AF\_UNIX;

strncpy(addr.sun\_path, SOCKET\_NAME, sizeof(addr.sun\_path)-1);

ret = connect(client\_socket, (const struct sockaddr\*)&addr, sizeof(struct sockaddr\_un));

if(ret < 0) {

perror("connect");

exit(EXIT\_FAILURE);

}

while(1)

{

// Write data to buffer

sprintf(buffer, "@client--> ping");

// Send data to server

ret = write(client\_socket, buffer, BUFFER\_SIZE);

if(ret < 0) {

perror("write");

exit(EXIT\_FAILURE);

}

// Receive response from server

ret = read(client\_socket, buffer, BUFFER\_SIZE);

if(ret < 0) {

perror("read");

exit(EXIT\_FAILURE);

}

// Ensure buffer is 0-terminated

buffer[BUFFER\_SIZE-1] = 0;

// Print content in buffer

printf("Response from server: %s\n", buffer);

sleep(1);

}

// Close socket

close(client\_socket);

exit(EXIT\_SUCCESS);

return 0;

}

***server.c***

#include <sys/socket.h>

#include <sys/types.h>

#include <sys/un.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#define BACKLOG 2

#define BUFFER\_SIZE 50

#define SOCKET\_NAME "/tmp/utsav.socket"

int main()

{

printf("server starting...\n");

struct sockaddr\_un addr;

int listen\_socket;

int client\_socket;

int ret;

char buffer[BUFFER\_SIZE];

// Unlink socket if something occured during last run

unlink(SOCKET\_NAME);

// Create socket

listen\_socket = socket(AF\_UNIX, SOCK\_SEQPACKET, 0);

if(listen\_socket < 0) {

perror("socket");

exit(EXIT\_FAILURE);

}

// Setup socket address family

memset(&addr, 0, sizeof(struct sockaddr\_un));

// Bind socket

addr.sun\_family = AF\_UNIX;

strncpy(addr.sun\_path, SOCKET\_NAME, sizeof(addr.sun\_path)-1);

ret = bind(listen\_socket, (const struct sockaddr\*)&addr, sizeof(struct sockaddr\_un));

if(ret < 0) {

perror("bind");

exit(EXIT\_FAILURE);

}

// Listen for client connection

printf("Server is listening...\n");

ret = listen(listen\_socket, BACKLOG);

if(ret < 0){

perror("listen");

exit(EXIT\_FAILURE);

}

while(1)

{

// Accept client connection

client\_socket = accept(listen\_socket, NULL, NULL);

if(client\_socket < 0) {

perror("accept");

exit(EXIT\_FAILURE);

}

// Handle client connection

while(1)

{

// Read data from client

ret = read(client\_socket, buffer, BUFFER\_SIZE);

if(ret < 0) {

perror("read");

exit(EXIT\_FAILURE);

}

// Make sure buffer is 0-terminated

buffer[BUFFER\_SIZE-1] = 0;

// Print content of buffer

printf("Received data: %s\n", buffer);

// Write content to buffer

sprintf(buffer, "@server--> pong");

// Send data to client

ret = write(client\_socket, buffer, BUFFER\_SIZE);

if(ret < 0) {

perror("write");

exit(EXIT\_FAILURE);

}

sleep(1);

}

close(listen\_socket);

unlink(SOCKET\_NAME);

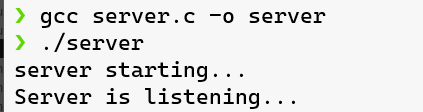
exit(EXIT\_SUCCESS);

}

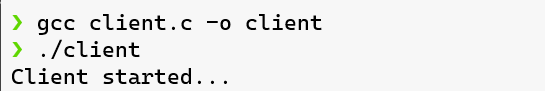
return 0;

}

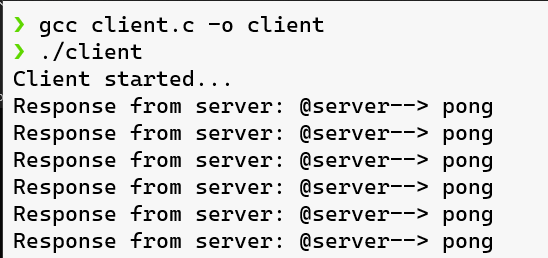
**# Server (start)**

****

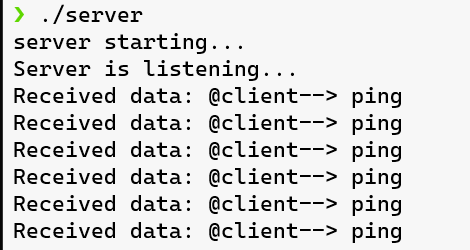
**# Client (start)**

****

**# Client receiving response from server**

****

**# Server listening from client**

****