**Inter-process communication-Named Pipes or FIFO**

Named pipes, also known as FIFOs (First-In, First-Out), are a form of inter-process communication (IPC) in Linux. Named pipes allow communication between unrelated processes by providing a named file-like interface. Here's an example code snippet that demonstrates the usage of named pipes for IPC:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/stat.h>

#define BUFFER\_SIZE 256

int main() {

 const char\* fifoPath = "/tmp/myfifo";

 char buffer[BUFFER\_SIZE];

 // Create the named pipe

 if (mkfifo(fifoPath, 0666) == -1) {

 perror("Failed to create named pipe");

 exit(EXIT\_FAILURE);

 }

 pid\_t childPid = fork();

 if (childPid == -1) {

 perror("Failed to fork");

 exit(EXIT\_FAILURE);

 }

 if (childPid == 0) {

 // Child process (reader)

 int fd = open(fifoPath, O\_RDONLY);

 if (fd == -1) {

 perror("Failed to open named pipe for reading");

 exit(EXIT\_FAILURE);

 }

 // Read data from the named pipe

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

 if (bytesRead == -1) {

 perror("Failed to read from named pipe");

 exit(EXIT\_FAILURE);

 }

 printf("Child process received: %s\n", buffer);

 // Close the named pipe

 close(fd);

 } else {

 // Parent process (writer)

 int fd = open(fifoPath, O\_WRONLY);

 if (fd == -1) {

 perror("Failed to open named pipe for writing");

 exit(EXIT\_FAILURE);

 }

 // Write data to the named pipe

 const char\* message = "Hello from the parent process!";

 ssize\_t bytesWritten = write(fd, message, sizeof(message));

 if (bytesWritten == -1) {

 perror("Failed to write to named pipe");

 exit(EXIT\_FAILURE);

 }

 // Close the named pipe

 close(fd);

 }

 // Remove the named pipe

 if (unlink(fifoPath) == -1) {

 perror("Failed to remove named pipe");

 exit(EXIT\_FAILURE);

 }

 return 0;

}

In this example, mkfifo() is used to create a named pipe (FIFO) with the specified path (fifoPath). The 0666 permission parameter allows read and write access to all users.

The program then forks into two processes: the parent process (writer) and the child process (reader).

In the child process, the named pipe is opened in read-only mode using open(), and data is read from the named pipe using read(). The received data is printed to the console, and the named pipe is closed.

In the parent process, the named pipe is opened in write-only mode using open(), and data (in this case, a message) is written to the named pipe using write(). The named pipe is then closed.

Finally, both processes exit. The named pipe is removed using unlink() after the communication is complete.Remember to compile and run this code on a Linux system with a C compiler (e.g., gcc). Include the necessary header files (stdio.h, stdlib.h, unistd.h, fcntl.h, sys/stat.h) and use the -Wall flag to enable warnings.

Regenerate response