**Pipes in Inter-process Communication**

Pipes are a form of inter-process communication (IPC) in Linux that allow one-way communication between two related processes. A pipe can be used to connect the standard output of one process to the standard input of another process. Here's an example code snippet that demonstrates how to create and use pipes for IPC in Linux:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#define BUFFER\_SIZE 256

int main() {

 int pipefd[2]; // Pipe file descriptors

 char buffer[BUFFER\_SIZE];

 // Create the pipe

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

 perror("Failed to create pipe");

 exit(EXIT\_FAILURE);

 }

 pid\_t childPid = fork();

 if (childPid == -1) {

 perror("Failed to fork");

 exit(EXIT\_FAILURE);

 }

 if (childPid == 0) {

 // Child process (reader)

 // Close the write end of the pipe

 close(pipefd[1]);

 // Read data from the pipe

 ssize\_t bytesRead = read(pipefd[0], buffer, BUFFER\_SIZE);

 if (bytesRead == -1) {

 perror("Failed to read from pipe");

 exit(EXIT\_FAILURE);

 }

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

 // Close the read end of the pipe

 close(pipefd[0]);

 } else {

 // Parent process (writer)

 // Close the read end of the pipe

 close(pipefd[0]);

 // Write data to the pipe

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

 ssize\_t bytesWritten = write(pipefd[1], message, BUFFER\_SIZE);

 if (bytesWritten == -1) {

 perror("Failed to write to pipe");

 exit(EXIT\_FAILURE);

 }

 // Close the write end of the pipe

 close(pipefd[1]);

 }

 return 0;

}

In this example, a pipe is created using the pipe() function, which returns two file descriptors: pipefd[0] for reading (the read end of the pipe) and pipefd[1] for writing (the write end of the pipe).

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

In the child process, the write end of the pipe (pipefd[1]) is closed, and data is read from the read end of the pipe (pipefd[0]) using the read() function. The received data is printed to the console, and the read end of the pipe is closed.

In the parent process, the read end of the pipe (pipefd[0]) is closed, and data (in this case, a message) is written to the write end of the pipe (pipefd[1]) using the write() function. The write end of the pipe is then closed.

Finally, both processes exit.

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) and use the -Wall flag to enable warnings.