**3. Introduction to pipes and related system calls for pipe management**

**1. Understand the Concept**

A **pipe** is a unidirectional communication channel:

* Data written to the pipe by one process can be read by another.
* A pipe can be created using the pipe() system call in Linux.

**2. Use the pipe() System Call**

* The pipe() system call creates a pipe.
* It returns two file descriptors:
  + **fd[0]**: Read end of the pipe.
  + **fd[1]**: Write end of the pipe.

**3. Fork a Child Process**

* Use the fork() system call to create a child process.
* Parent and child processes can communicate through the pipe.

**4. Close Unused Ends**

* In the parent process, close the read end of the pipe if it's only writing.
* In the child process, close the write end of the pipe if it's only reading.

**5. Write and Read Data**

* The parent process writes data to the pipe.
* The child process reads the data from the pipe.

**6. Code Implementation**

Below is an example in C:

Copy code

#include <stdio.h>

#include <unistd.h>

#include <string.h>

int main() {

int fd[2]; // File descriptors for the pipe

pid\_t pid;

char write\_msg[] = "Hello from parent!";

char read\_msg[100];

// Step 2: Create the pipe

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

perror("Pipe failed");

return 1;

}

// Step 3: Fork a child process

pid = fork();

if (pid < 0) {

perror("Fork failed");

return 1;

}

if (pid > 0) { // Parent process

// Step 4: Close unused read end

close(fd[0]);

// Step 5: Write to the pipe

write(fd[1], write\_msg, strlen(write\_msg) + 1);

close(fd[1]); // Close write end after writing

} else { // Child process

// Step 4: Close unused write end

close(fd[1]);

// Step 5: Read from the pipe

read(fd[0], read\_msg, sizeof(read\_msg));

printf("Child received: %s\n", read\_msg);

close(fd[0]); // Close read end after reading

}

return 0;

}

**7. Explanation**

1. **pipe(fd)**: Creates a pipe with fd[0] for reading and fd[1] for writing.
2. **fork()**: Creates a child process.
3. **Parent Process**:
   * Closes the read end (fd[0]).
   * Writes data to the write end (fd[1]).
4. **Child Process**:
   * Closes the write end (fd[1]).
   * Reads data from the read end (fd[0]).
5. Communication is complete, and ends are closed.

**8. Compile and Run**

bash

Copy code

gcc -o pipe\_example pipe\_example.c

./pipe\_example

**Expected Output**

bash

Copy code

Child received: Hello from parent!