Operating Systems (CS3000)

Lecture – 16

(Inter Process Communication - 2)

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING, KANCHEEPURAM

Dr. Jaishree Mayank

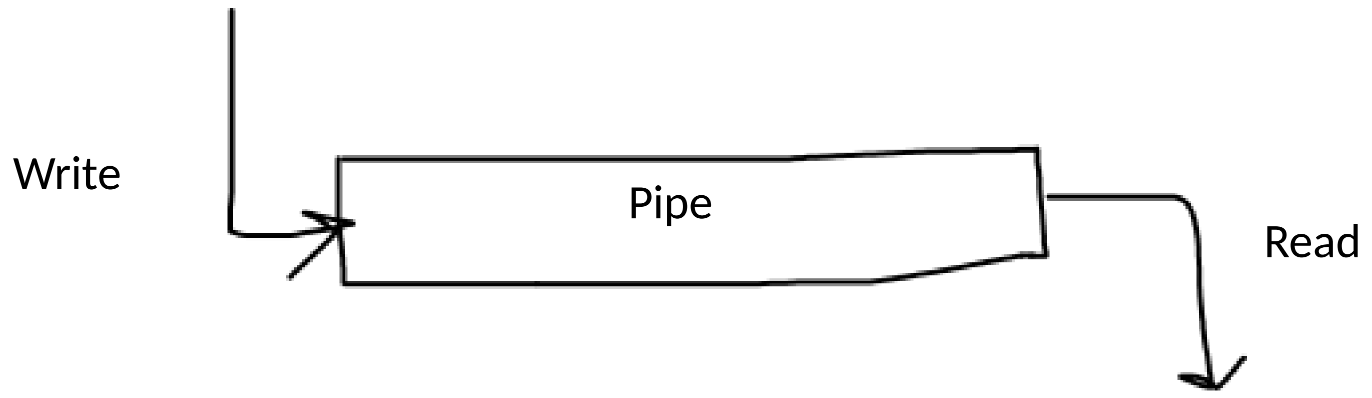
Assistant Professor

Department of Computer Sc. and Engg.

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING, KANCHEEPURAM

Message Passing using Pipes

- Pipe is a communication between parent and child process
- Communication is achieved by one process writing into the pipe and other reading from the pipe
- To achieve the pipe system call, create two descriptors, one to write into the file and another to read from the file.
- Patent [illegible] Child



Creating Pipe between Parent and Child process

- **int pipe**(int **pipefd[2]**);
- a file descriptor is used to access the two ends of the pipe: one end for reading and one end for writing
  - pipefd[0] is the file descriptor for reading.
  - pipefd[1] is the file descriptor for writing.
- Returns zero on success
- Returns -1 in case of failure

Message Passing using Pipes

- **ssize_t read**(int **fd**, void ***buf**, size_t **count**)

  - The file descriptor to read from.
  - A pointer to a buffer where the read data will be stored.
  - The maximum number of bytes to read.

- Returns the number of bytes read
- Returns -1 in case of failure

Message Passing using Pipes

- **ssize_t** **write**(int fd, void *buf, size_t **count**)

  - The file descriptor to write to.
  - A pointer to a buffer where the write data will be stored.
  - The maximum number of bytes to write.
- Return the number of bytes written
- Return zero in case nothing is written
- Return -1 in case of failure

Message Passing using Pipes

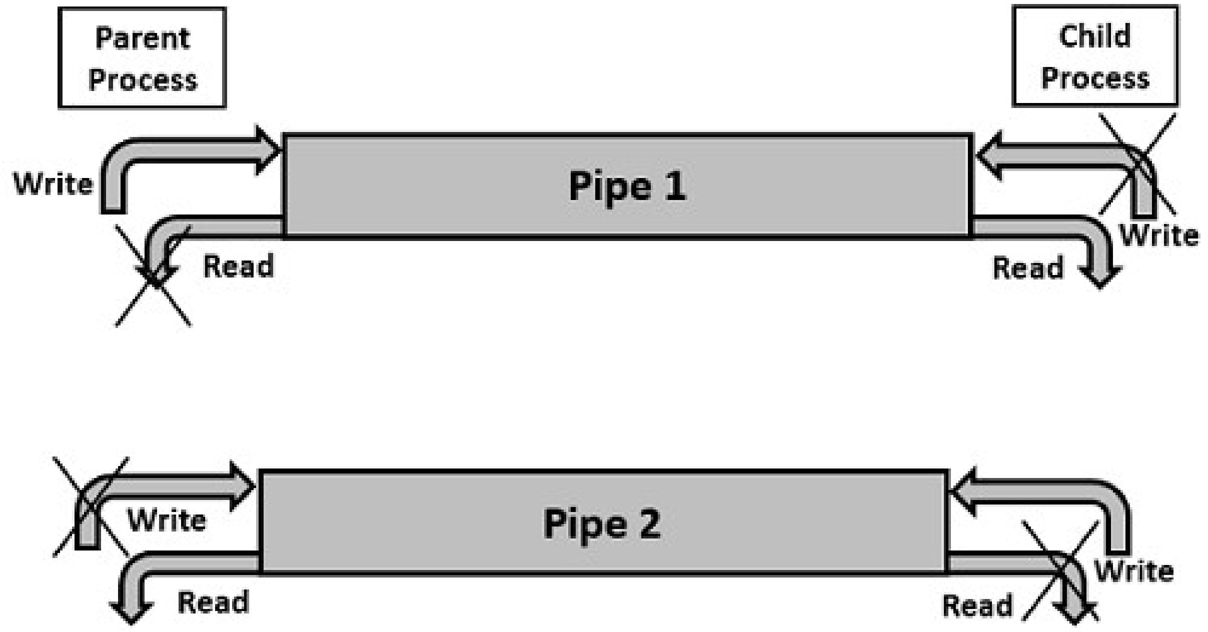
- **_int close**(int fd)
  - Closing the pipe end.
- Return zero on success
- Return -1 in case of failure

Message Passing using Pipes

- Algorithm
- **Step 1** − Create a pipe
- **Step 2** − Create a child process
- **Step 3** − Parent process writes to the pipe
- **Step 4** − Child process retrieves the message from the pipe and writes it to the standard output
- **Step 5** − Repeat step 3 and step 4 once again

Two-way Communication Using Pipes

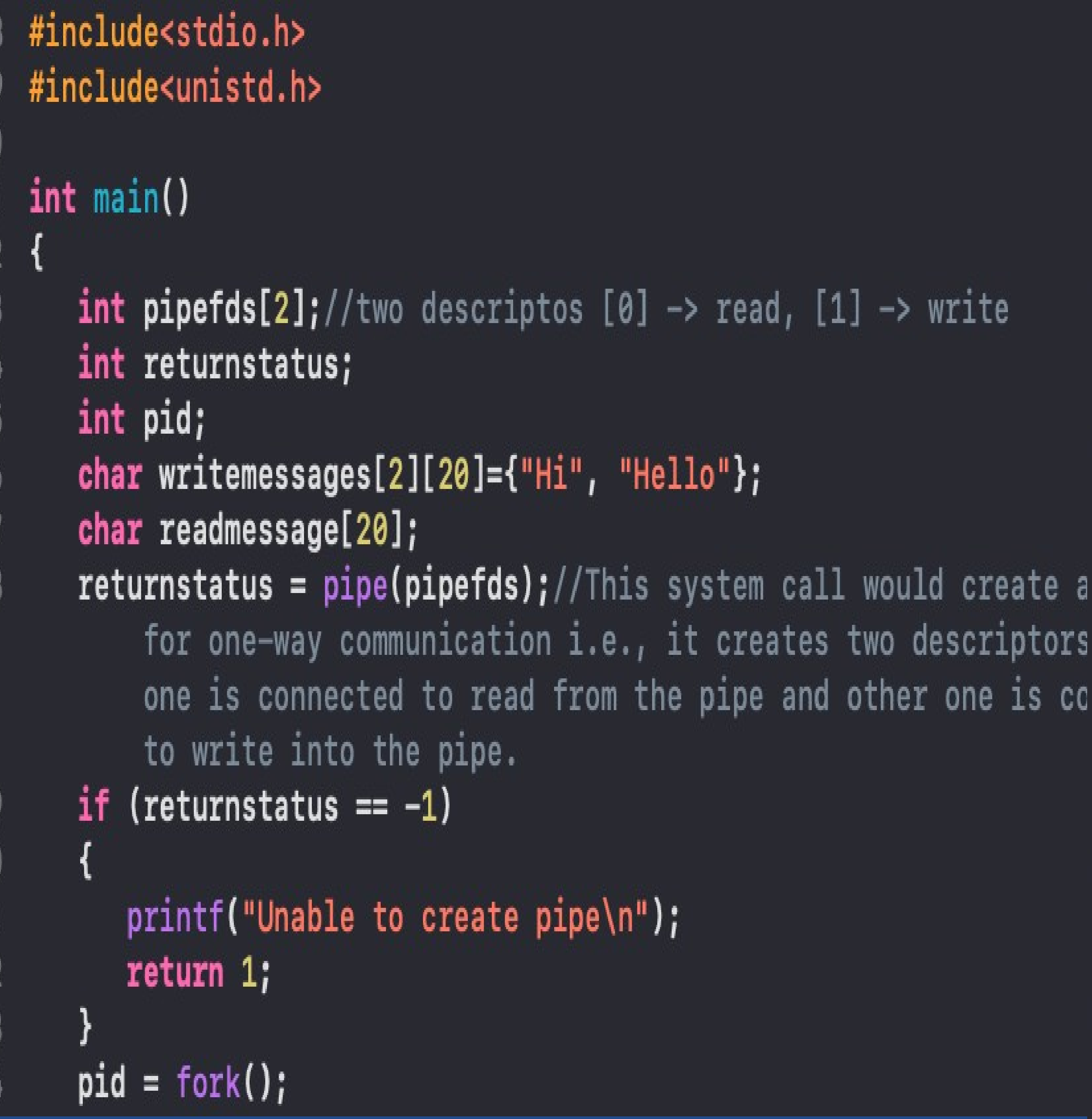
- If both the parent and the child needs to write and read from the pipes simultaneously
  - Two pipes are required



Two-way Communication Using Pipes

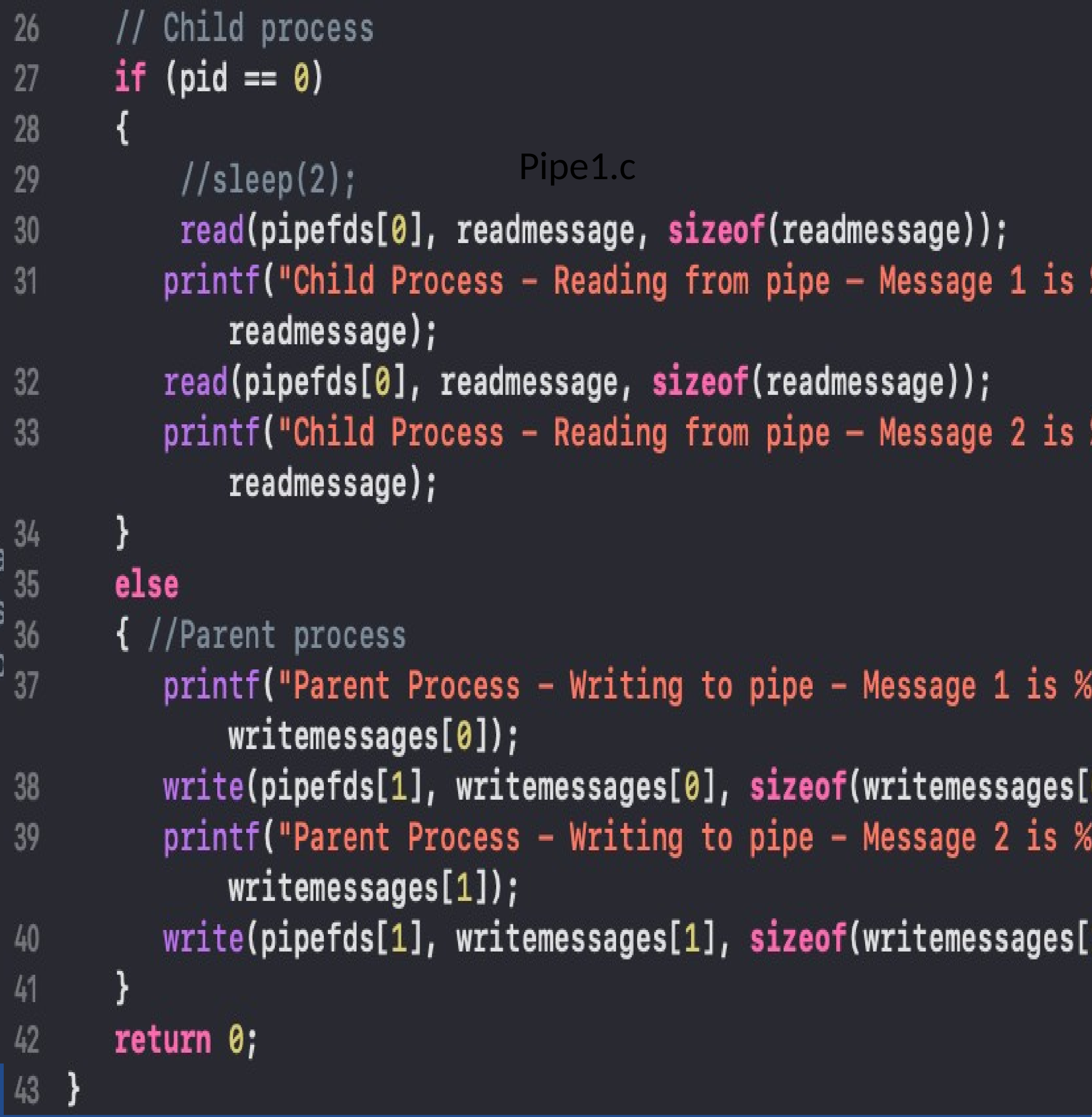
- Algorithm
- **Step 1** − Create pipe1 for the parent process to write and the child process to read
- **Step 2** − Create pipe2 for the child process to write and the parent process to read
- **Step 3** − Close the unwanted ends of the pipe from the parent and child side
- **Step 4** − Parent process to write a message and child process to read and display on the screen
- **Step 5** − Child process to write a message and parent process to read and display on the screen

Pipe1.c



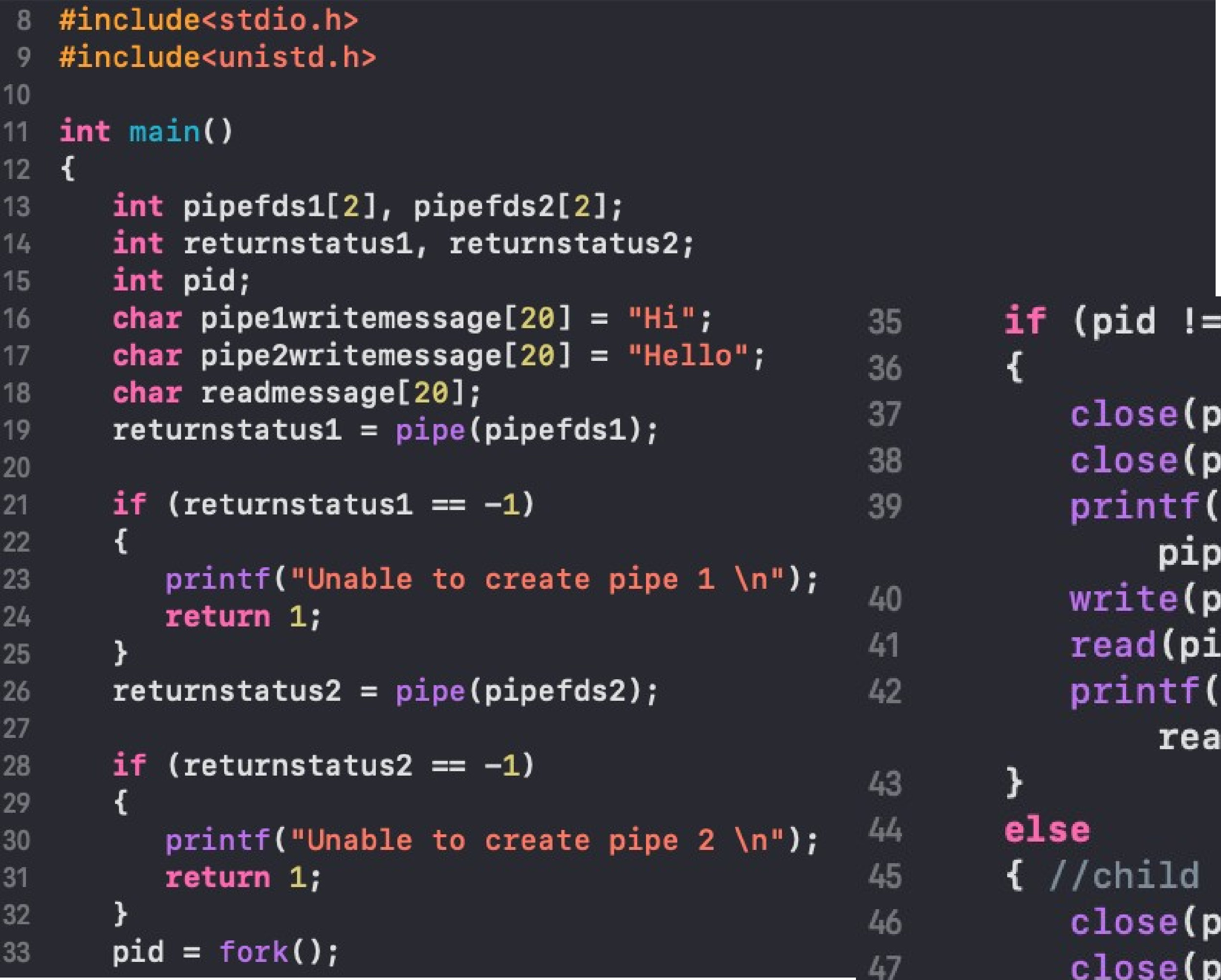
```
#include<stdio.h>
#include<unistd.h>

int main()
{
   int pipefds[2];//two descriptos [0] -> read, [1] -> write
   int returnstatus;
   int pid;
   char writemessages[2][20]={"Hi", "Hello"};
   char readmessage[20];
   returnstatus = pipe(pipefds);//This system call would create a
      for one-way communication i.e., it creates two descriptors
      one is connected to read from the pipe and other one is co
      to write into the pipe.
   if (returnstatus == -1)
   {
      printf("Unable to create pipe\n");
      return 1;
   }
   pid = fork();
```



```
   // Child process
   if (pid == 0)
   {
       //sleep(2);
       read(pipefds[0], readmessage, sizeof(readmessage));
      printf("Child Process - Reading from pipe - Message 1 is 
         readmessage);
      read(pipefds[0], readmessage, sizeof(readmessage));
      printf("Child Process - Reading from pipe - Message 2 is 
         readmessage);
   }
   else
   { //Parent process
      printf("Parent Process - Writing to pipe - Message 1 is %
         writemessages[0]);
      write(pipefds[1], writemessages[0], sizeof(writemessages[
      printf("Parent Process - Writing to pipe - Message 2 is %
         writemessages[1]);
      write(pipefds[1], writemessages[1], sizeof(writemessages[
   }
   return 0;
}
```

Pipe2.c

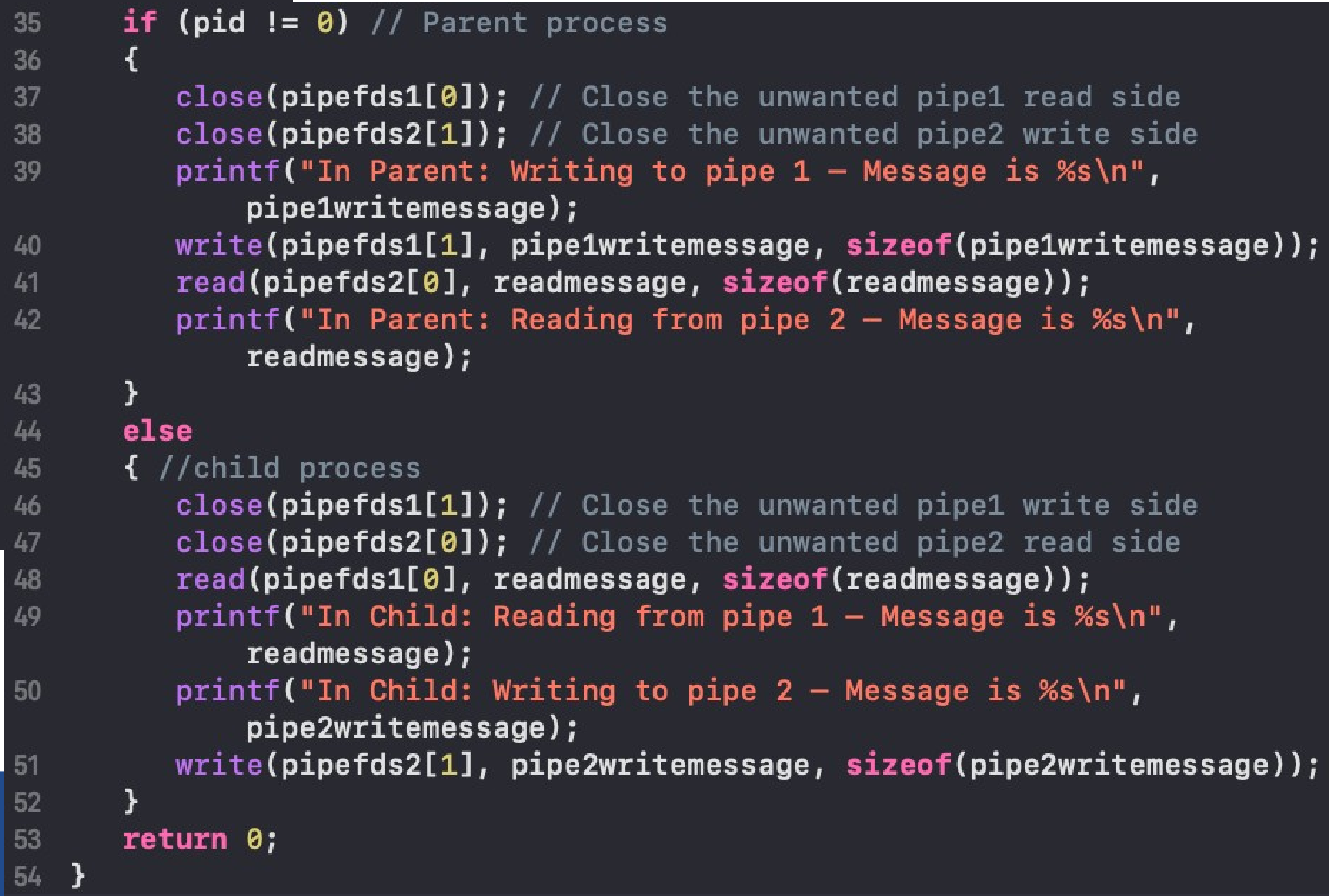


```c
#include<stdio.h>
#include<unistd.h>

int main()
{
   int pipefds1[2], pipefds2[2];
   int returnstatus1, returnstatus2;
   int pid;
   char pipe1writemessage[20] = "Hi";
   char pipe2writemessage[20] = "Hello";
   char readmessage[20];
   returnstatus1 = pipe(pipefds1);

   if (returnstatus1 == -1)
   {
      printf("Unable to create pipe 1 \n");
      return 1;
   }
   returnstatus2 = pipe(pipefds2);

   if (returnstatus2 == -1)
   {
      printf("Unable to create pipe 2 \n");
      return 1;
   }
   pid = fork();
```



```c
   if (pid != 0) // Parent process
   {
      close(pipefds1[0]); // Close the unwanted pipe1 read side
      close(pipefds2[1]); // Close the unwanted pipe2 write side
      printf("In Parent: Writing to pipe 1 – Message is %s\n",
          pipe1writemessage);
      write(pipefds1[1], pipe1writemessage, sizeof(pipe1writemessage));
      read(pipefds2[0], readmessage, sizeof(readmessage));
      printf("In Parent: Reading from pipe 2 – Message is %s\n",
          readmessage);
   }
   else
   { //child process
      close(pipefds1[1]); // Close the unwanted pipe1 write side
      close(pipefds2[0]); // Close the unwanted pipe2 read side
      read(pipefds1[0], readmessage, sizeof(readmessage));
      printf("In Child: Reading from pipe 1 – Message is %s\n",
          readmessage);
      printf("In Child: Writing to pipe 2 – Message is %s\n",
          pipe2writemessage);
      write(pipefds2[1], pipe2writemessage, sizeof(pipe2writemessage));
   }
   return 0;
}
```

Thank You

Any Questions?