The Prestige (2006), Christopher Nolan

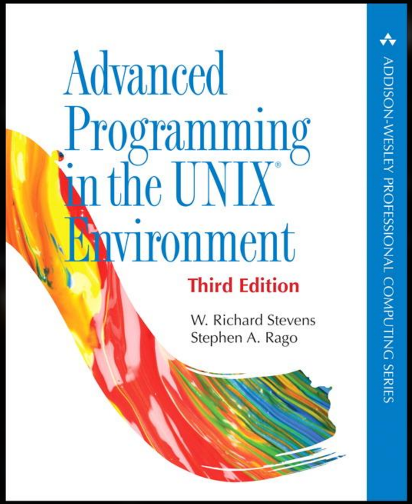
Lab08 and Lec08 → Marks & Keys

Lab09 and Lec09 → Nov. 24

Lab11 and Lec11 → Dec. 01

Final Exam → Dec. 14, 7:00 PM

https://www.uwindsor.ca/registrar/sites/uwindsor.ca.registrar/files/fall_2021_exam_schedule.pdf



Advanced Programming in the UNIX® Environment

Third Edition

W. Richard Stevens
Stephen A. Rago

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Chapter 15: Inter-Process Communication

Chapter 16: Network IPC (Sockets)

Multiprocessing

aka multiprogramming

Single Processor ~~Multiprocessor~~

IPC

Normal Communication

Can you do this for me? Yes, here is it. Anything else?

Parent ← Child

Passing the Results of Tasks
Passing Information

A) Share a Single File/Device (Lab09)
B) Share Part of Memory

Continuous Communication → Conversation

In previous examples, there exist a single communication.

The Prestige (2006), Christopher Nolan

Example III: Solution B

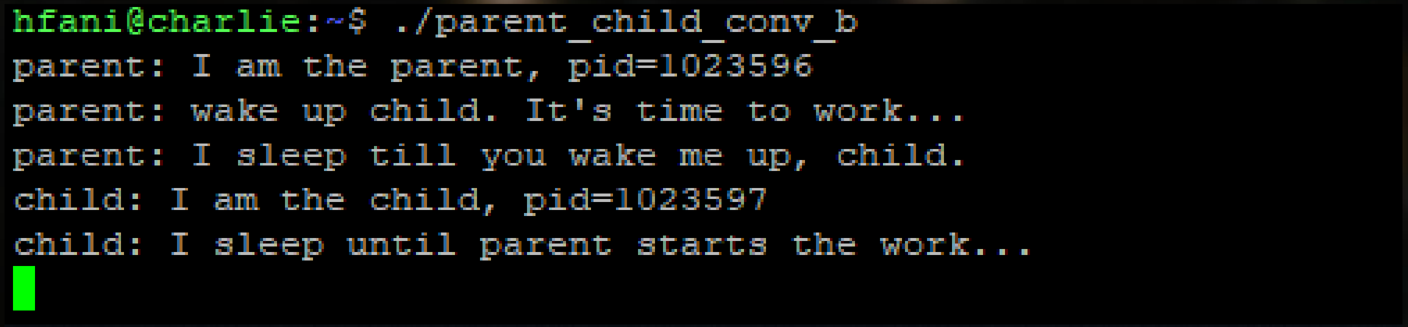
Same Child

IMPORTANT: the parent does NOT `wait()` for the child to `exit()`!
But `pause()` for the child for another round of conversation.

`sleep(int *second*)` cannot work because we depend on other process to wake up

Example III: Solution B

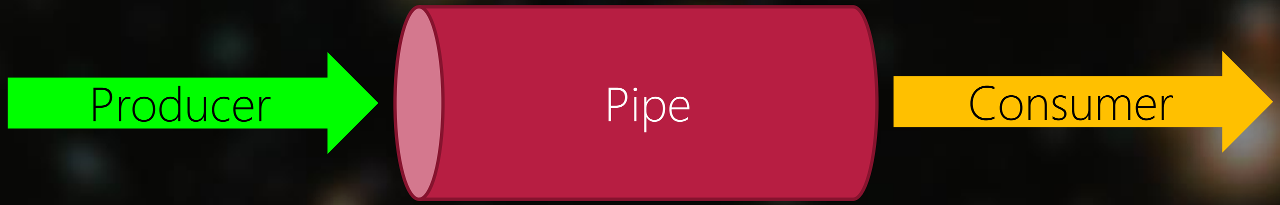
Does not work! Why?



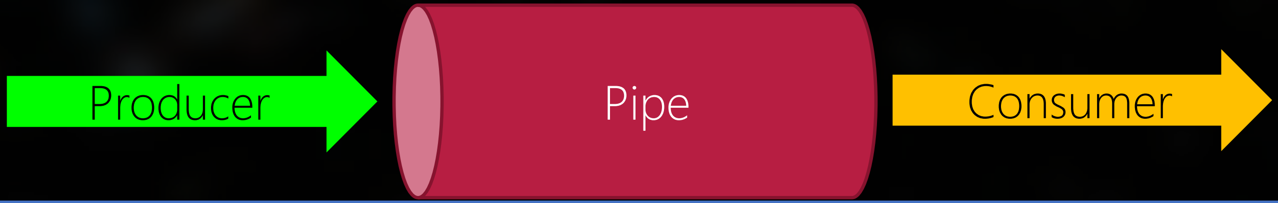
```
hfani@charlie:~$ ./parent_child_conv_b
parent: I am the parent, pid=1023596
parent: wake up child. It's time to work...
parent: I sleep till you wake me up, child.
child: I am the child, pid=1023597
child: I sleep until parent starts the work...
```

Unnamed File → Pipe

Handles all opening, closing, seeking, pauses, wakeups, ….
Temporary File, Memory, Device, …. (We don't know)

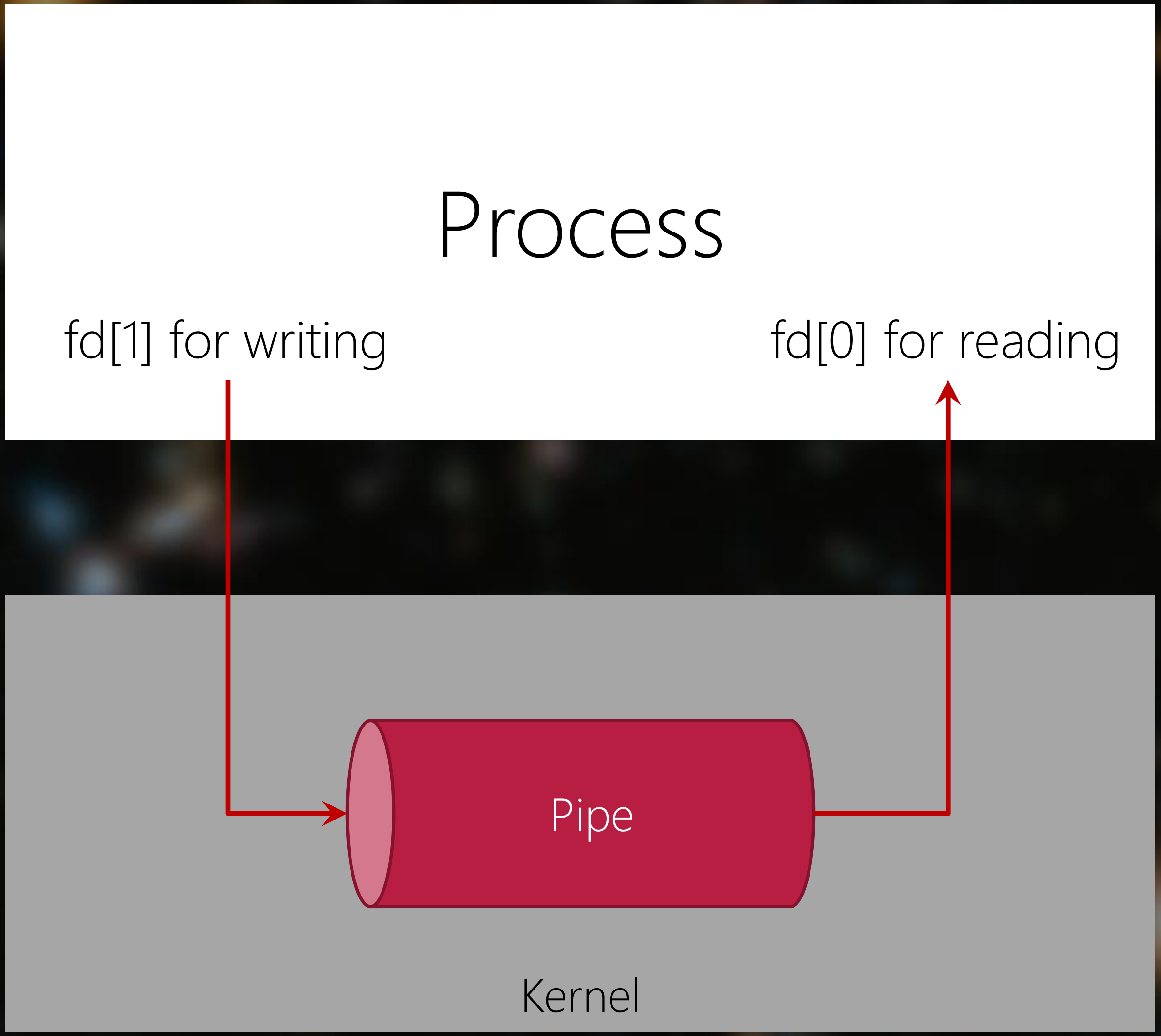


Unnamed File → Pipe

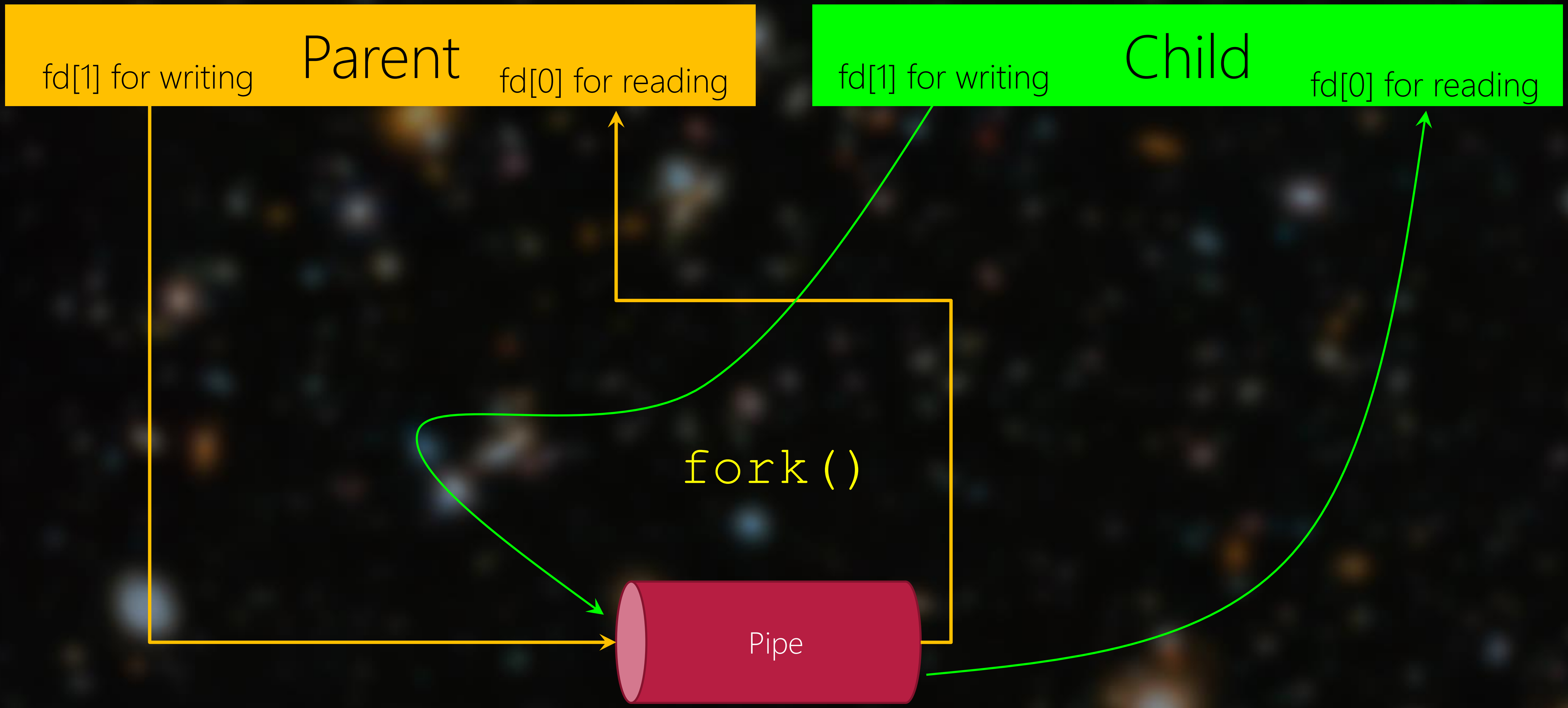


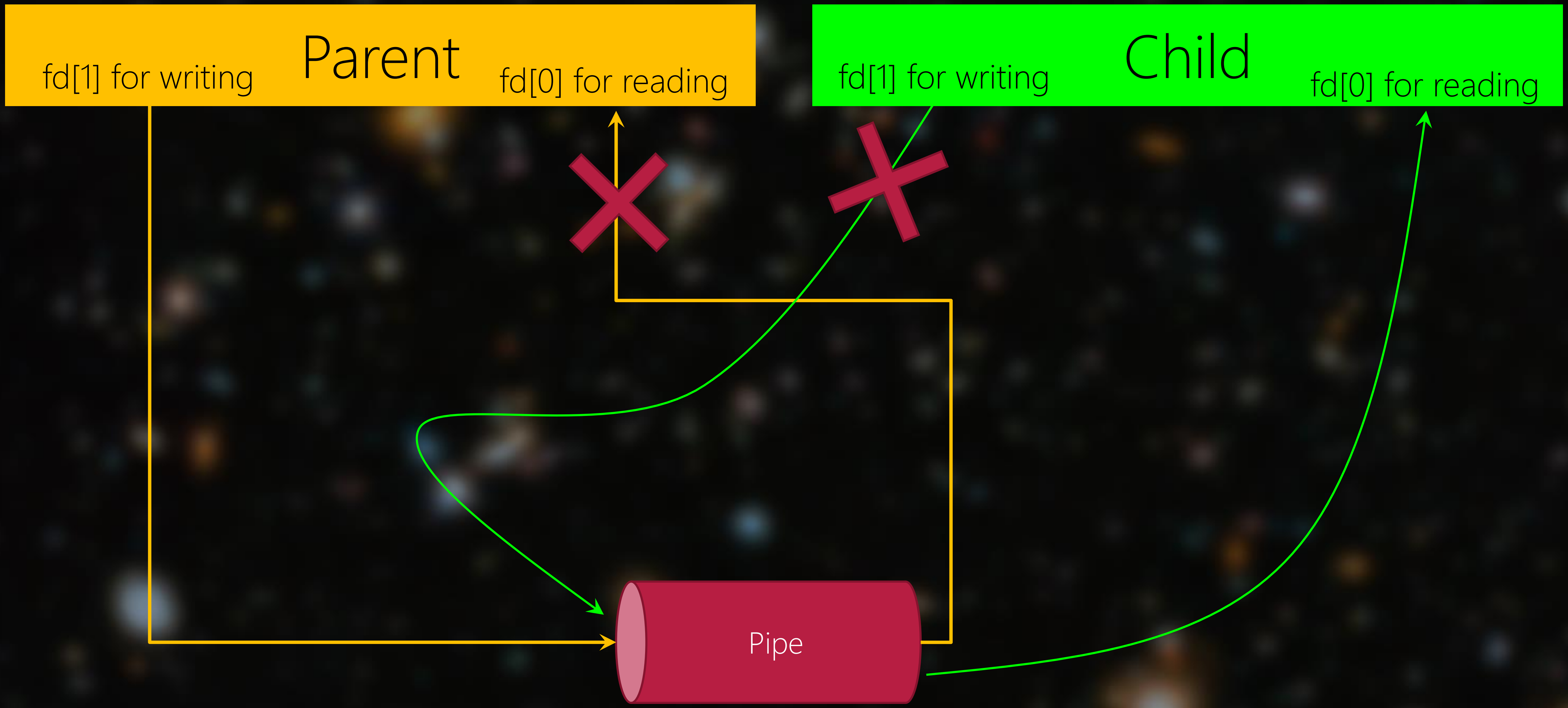
```
#include <unistd.h>
int pipe(int fd[2]);
```

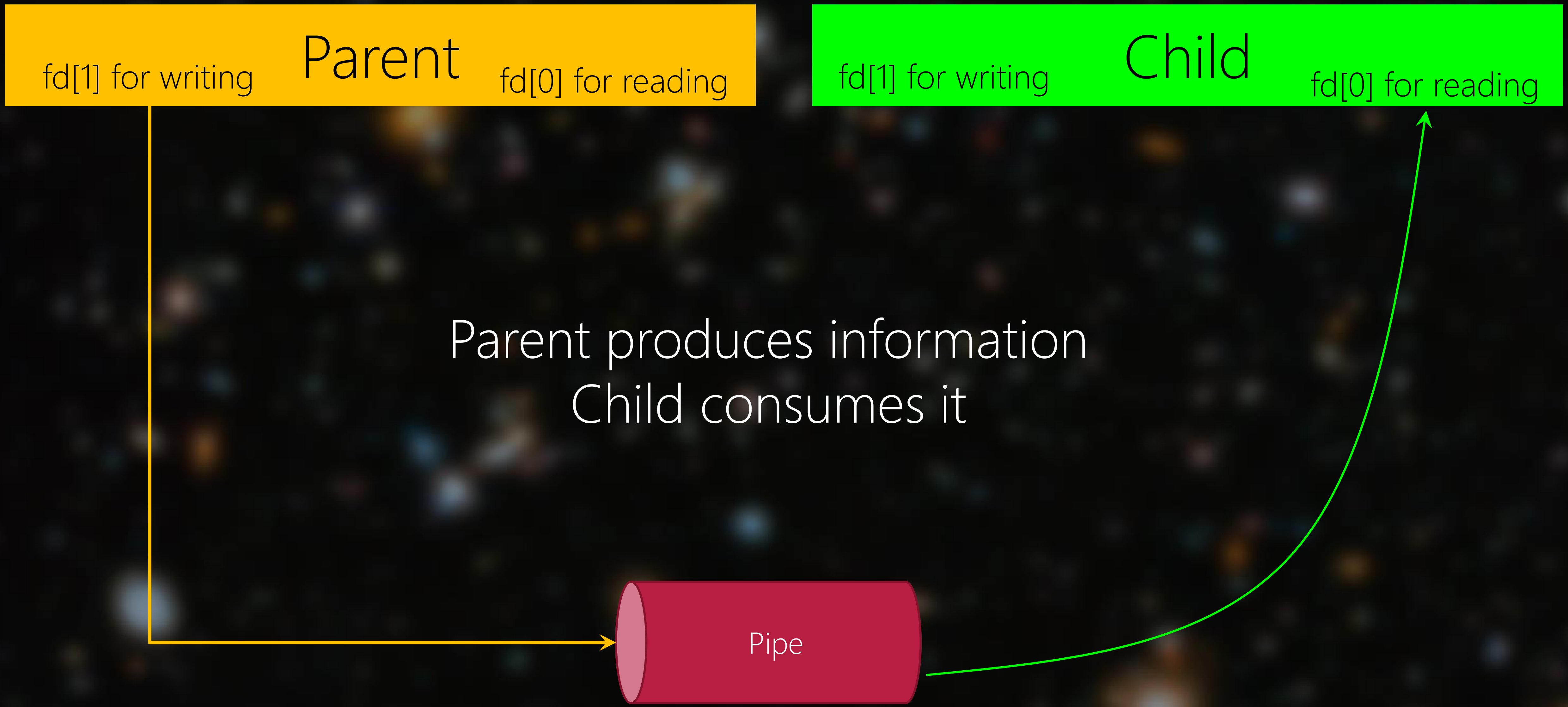
Returns 0 if OK, -1 on error



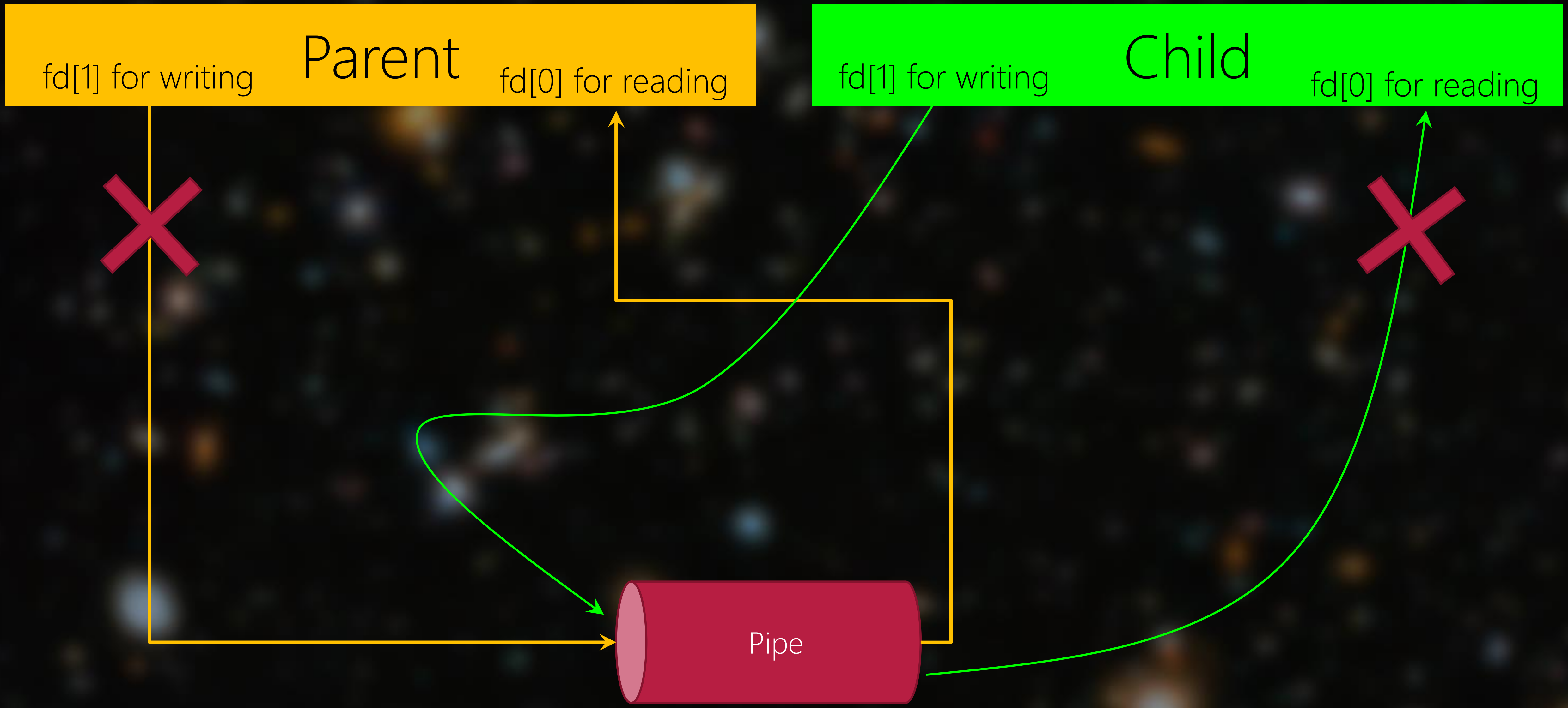
Process
fd[1] for writing
fd[0] for reading
Pipe
Kernel


Parent
fd[1] for writing
fd[0] for reading
Child
fd[1] for writing
fd[0] for reading
fork()
Pipe


Parent
fd[1] for writing
fd[0] for reading
Child
fd[1] for writing
fd[0] for reading
Pipe



Parent produces information
Child consumes it



Parent

fd[1] for writing

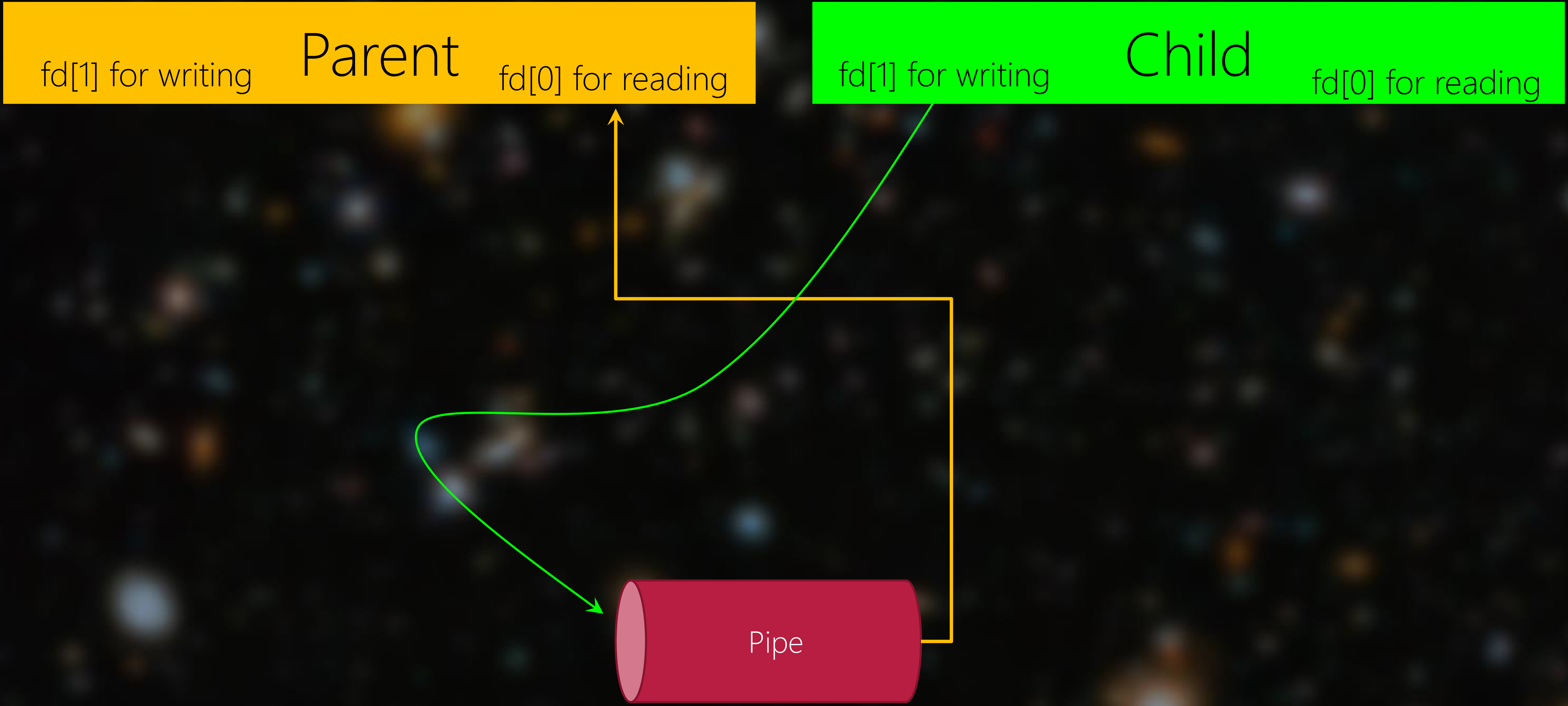
fd[0] for reading

Child

fd[1] for writing

fd[0] for reading

Pipe



Parent

fd[1] for writing

fd[0] for reading

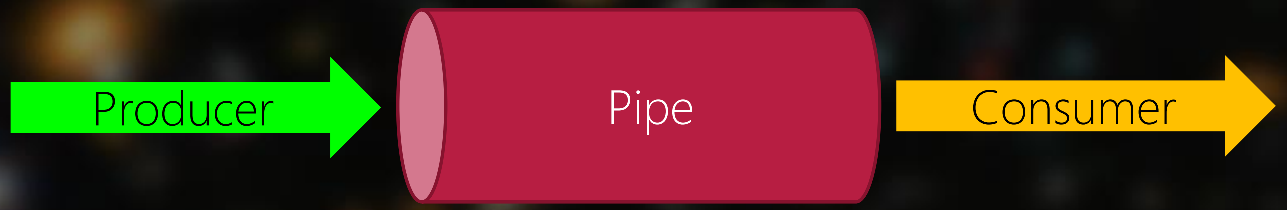
Child

fd[1] for writing

fd[0] for reading

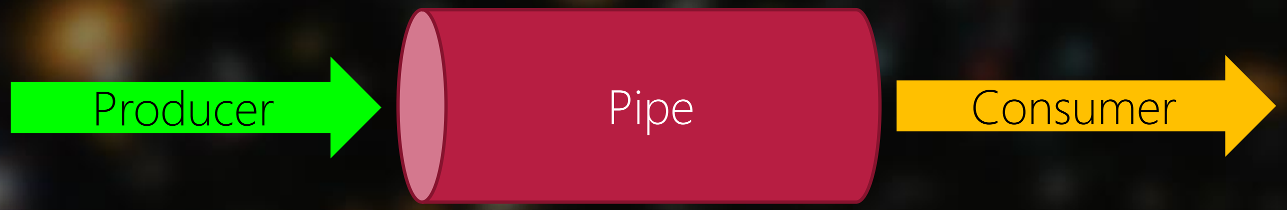
Pipe

Child produces information
Parent consumes it



Situations:

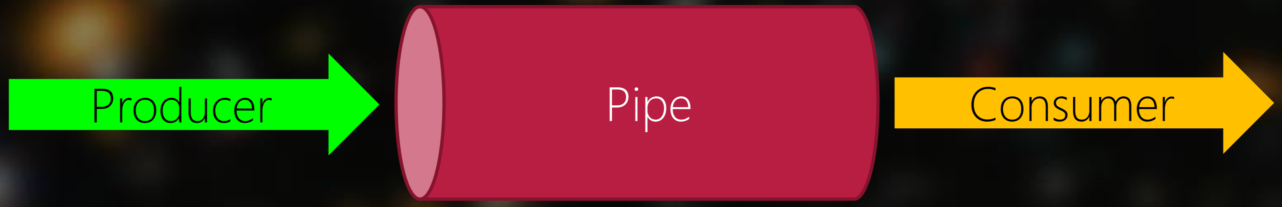
1) If the consumer wants to `read()` N bytes but there less data
2) If the consumer wants to `read()` but there is no data (empty pipe)
3) If the consumer wants to `read()` but there is no producer anymore
4) If the producer wants to `write()` but there is no consumer
5) If the producer wants to `write()` but pipe is full



Situations:

1) If the consumer wants to `read()` N bytes but there less data

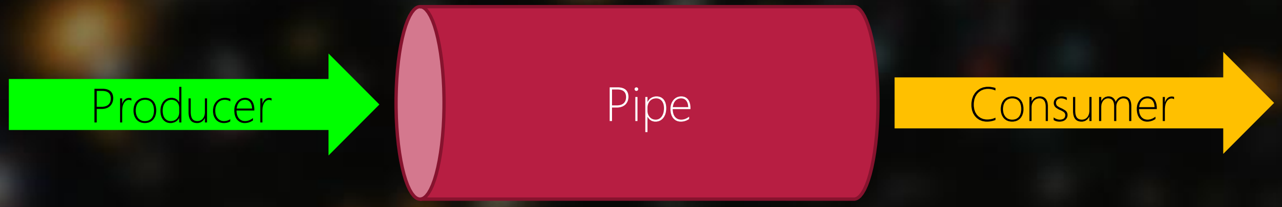
We already saw this when reading from a file while giving large buffer
Only the available data will be read



Situations:

2) If the consumer wants to `read()` but there is no data (empty pipe)

If a producer exists, the consumer `pause()` till the kernel SIGNALs it when at least 1 byte become available



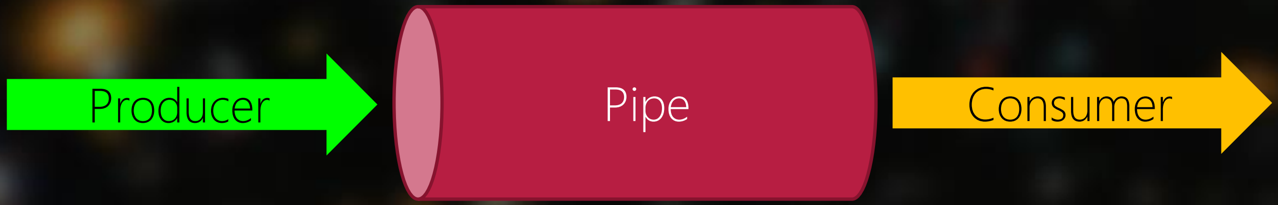
Situations:

3) If the consumer wants to `read()` but there is no producer anymore

The consumer can continue to read until there is no information left

The consumer does NOT `pause()`

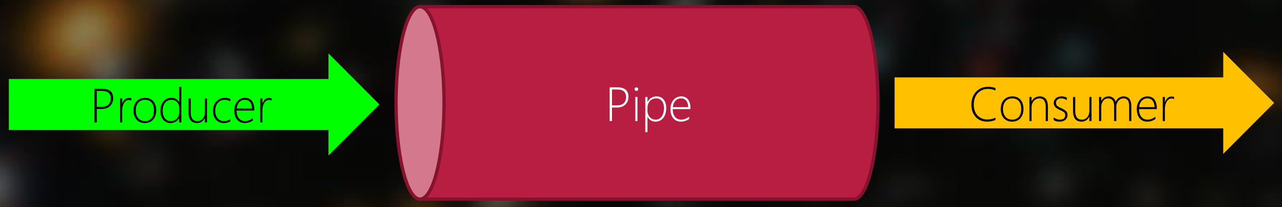
The last read returns 0 (EOF) and consumer decides to exit



Situations:

4) If the producer wants to `write()` but there is no consumer

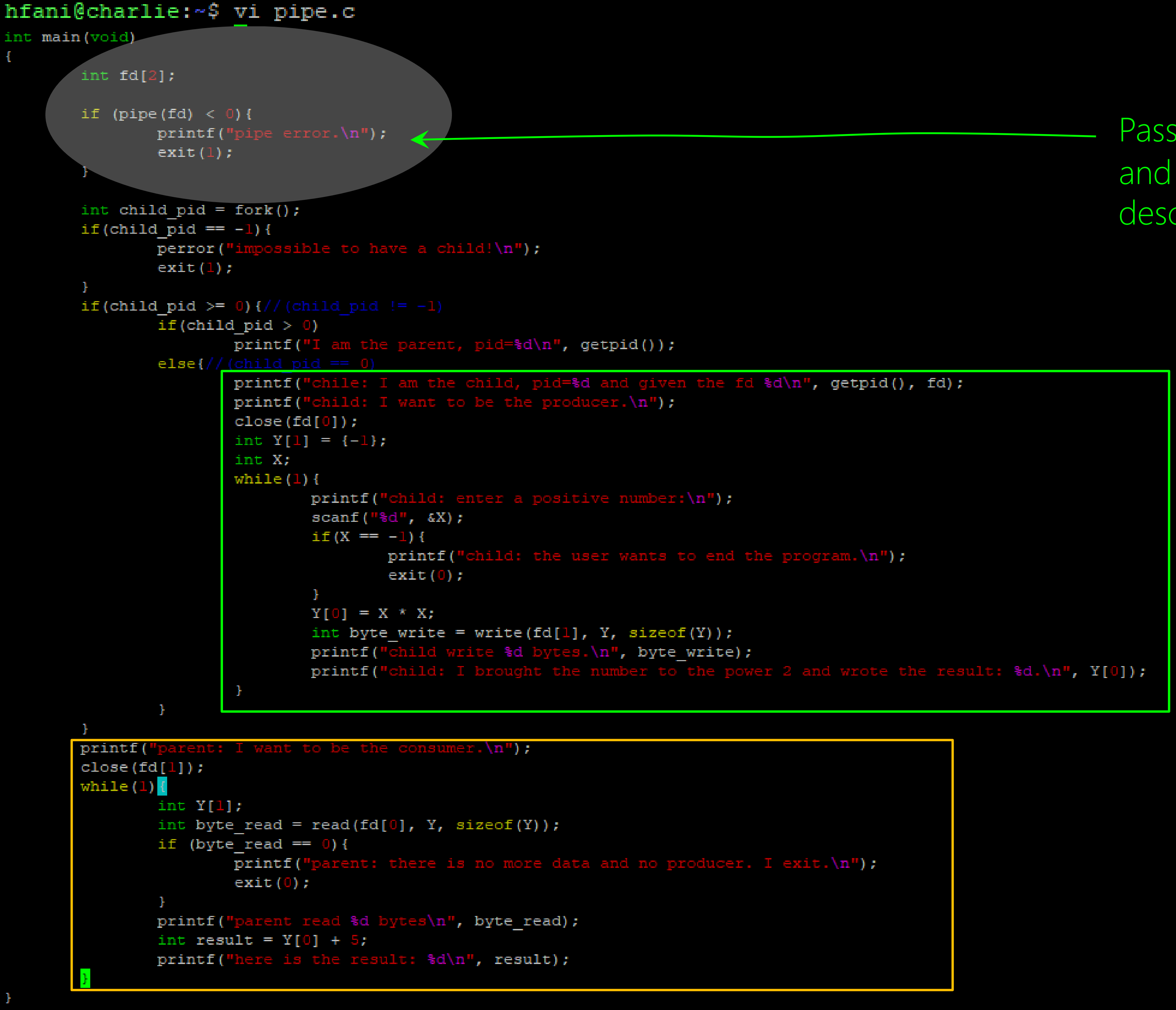
The producer fails and receives SIGPIPE by the kernel



Situations:

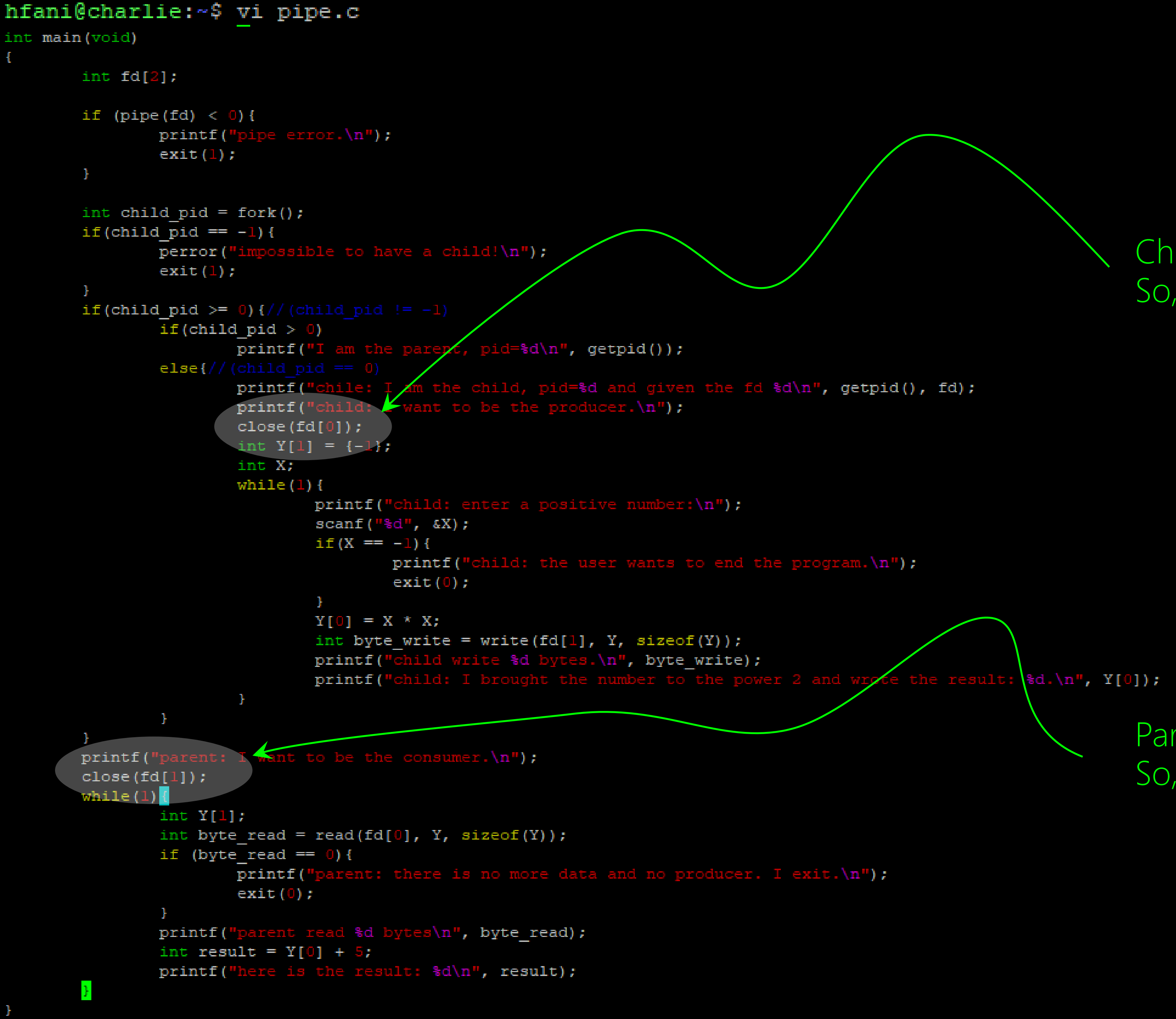
5) If the producer wants to `write()` but pipe is full

Any idea?



```
hfani@charlie:~$ vi pipe.c
int main(void)
{
        int fd[2];

        if (pipe(fd) < 0){
                printf("pipe error.\n");
                exit(1);
        }

        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!\n");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("chile: I am the child, pid=%d and given the fd %d\n", getpid(), fd);
                        printf("child: I want to be the producer.\n");
                        close(fd[0]);
                        int Y[1] = {-1};
                        int X;
                        while(1){
                                printf("child: enter a positive number:\n");
                                scanf("%d", &X);
                                if(X == -1){
                                        printf("child: the user wants to end the program.\n");
                                        exit(0);
                                }
                                Y[0] = X * X;
                                int byte_write = write(fd[1], Y, sizeof(Y));
                                printf("child write %d bytes.\n", byte_write);
                                printf("child: I brought the number to the power 2 and wrote the result: %d.\n", Y[0]);
                        }
                }
        }
        printf("parent: I want to be the consumer.\n");
        close(fd[1]);
        while(1){
                int Y[1];
                int byte_read = read(fd[0], Y, sizeof(Y));
                if (byte_read == 0){
                        printf("parent: there is no more data and no producer. I exit.\n");
                        exit(0);
                }
                printf("parent read %d bytes\n", byte_read);
                int result = Y[0] + 5;
                printf("here is the result: %d\n", result);
        }
}
```

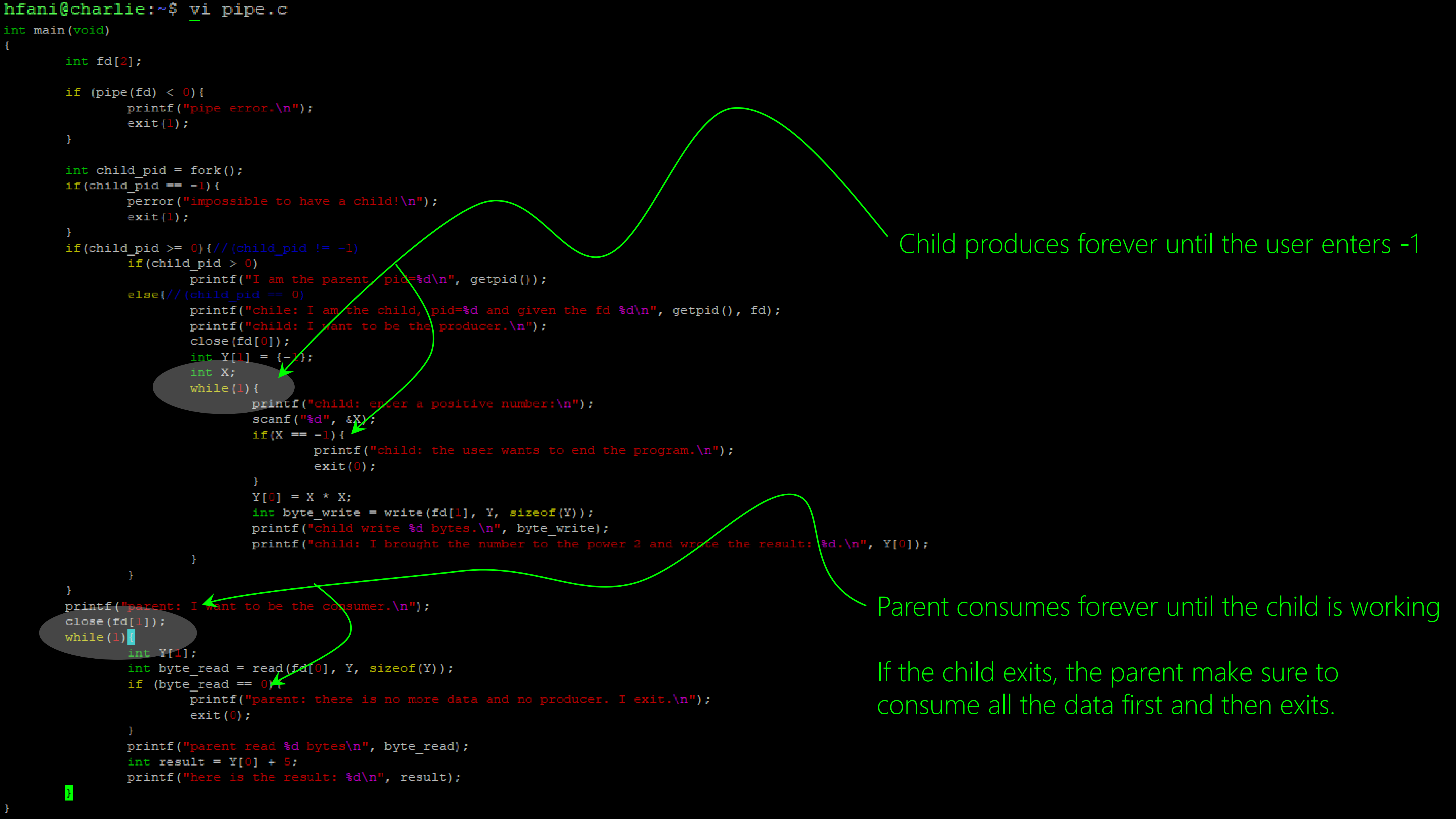
Passing an array of `fd[2]` to `pipe()` and receiving separate read and write file descriptors.



```
hfani@charlie:~$ vi pipe.c
int main(void)
{
        int fd[2];

        if (pipe(fd) < 0){
                printf("pipe error.\n");
                exit(1);
        }

        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!\n");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("chile: I am the child, pid=%d and given the fd %d\n", getpid(), fd);
                        printf("child: I want to be the producer.\n");
                        close(fd[0]);
                        int Y[1] = {-1};
                        int X;
                        while(1){
                                printf("child: enter a positive number:\n");
                                scanf("%d", &X);
                                if(X == -1){
                                        printf("child: the user wants to end the program.\n");
                                        exit(0);
                                }
                                Y[0] = X * X;
                                int byte_write = write(fd[1], Y, sizeof(Y));
                                printf("child write %d bytes.\n", byte_write);
                                printf("child: I brought the number to the power 2 and wrote the result: %d.\n", Y[0]);
                        }
                }
        }
        printf("parent: I want to be the consumer.\n");
        close(fd[1]);
        while(1){
                int Y[1];
                int byte_read = read(fd[0], Y, sizeof(Y));
                if (byte_read == 0){
                        printf("parent: there is no more data and no producer. I exit.\n");
                        exit(0);
                }
                printf("parent read %d bytes\n", byte_read);
                int result = Y[0] + 5;
                printf("here is the result: %d\n", result);
        }
}
```

Child is the producer.
So, it closes the read descriptor `fd[0]`

Parent is the consumer.
So, it closes the write descriptor `fd[1]`

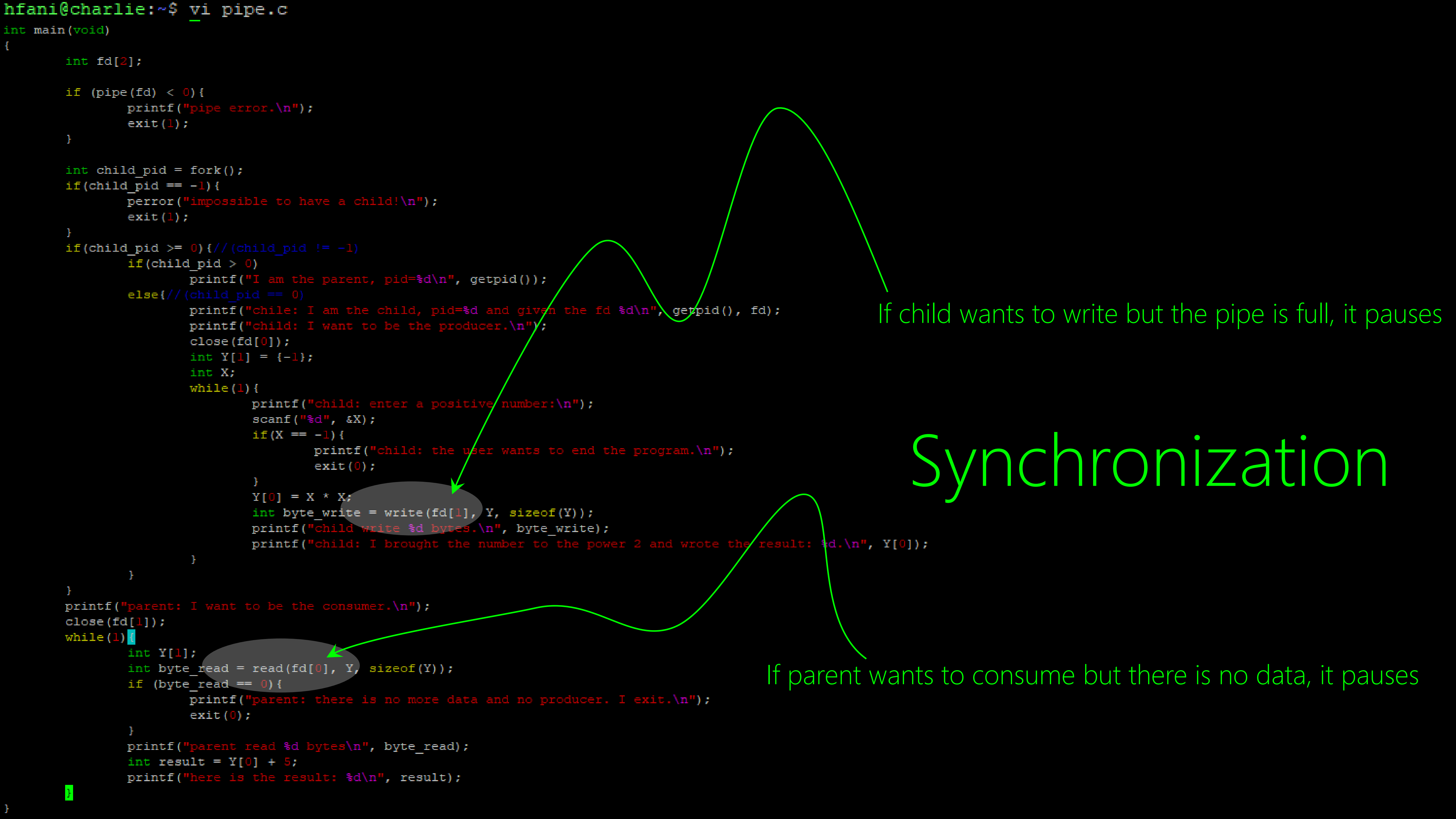


```
hfani@charlie:~$ vi pipe.c
int main(void)
{
        int fd[2];

        if (pipe(fd) < 0){
                printf("pipe error.\n");
                exit(1);
        }

        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!\n");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("chile: I am the child, pid=%d and given the fd %d\n", getpid(), fd);
                        printf("child: I want to be the producer.\n");
                        close(fd[0]);
                        int Y[1] = {-1};
                        int X;
                        while(1){
                                printf("child: enter a positive number:\n");
                                scanf("%d", &X);
                                if(X == -1){
                                        printf("child: the user wants to end the program.\n");
                                        exit(0);
                                }
                                Y[0] = X * X;
                                int byte_write = write(fd[1], Y, sizeof(Y));
                                printf("child write %d bytes.\n", byte_write);
                                printf("child: I brought the number to the power 2 and wrote the result: %d.\n", Y[0]);
                        }
                }
        }
        printf("parent: I want to be the consumer.\n");
        close(fd[1]);
        while(1){
                int Y[1];
                int byte_read = read(fd[0], Y, sizeof(Y));
                if (byte_read == 0){
                        printf("parent: there is no more data and no producer. I exit.\n");
                        exit(0);
                }
                printf("parent read %d bytes\n", byte_read);
                int result = Y[0] + 5;
                printf("here is the result: %d\n", result);
        }
}
```

Child produces forever until the user enters -1

Parent consumes forever until the child is working

If the child exits, the parent make sure to consume all the data first and then exits.

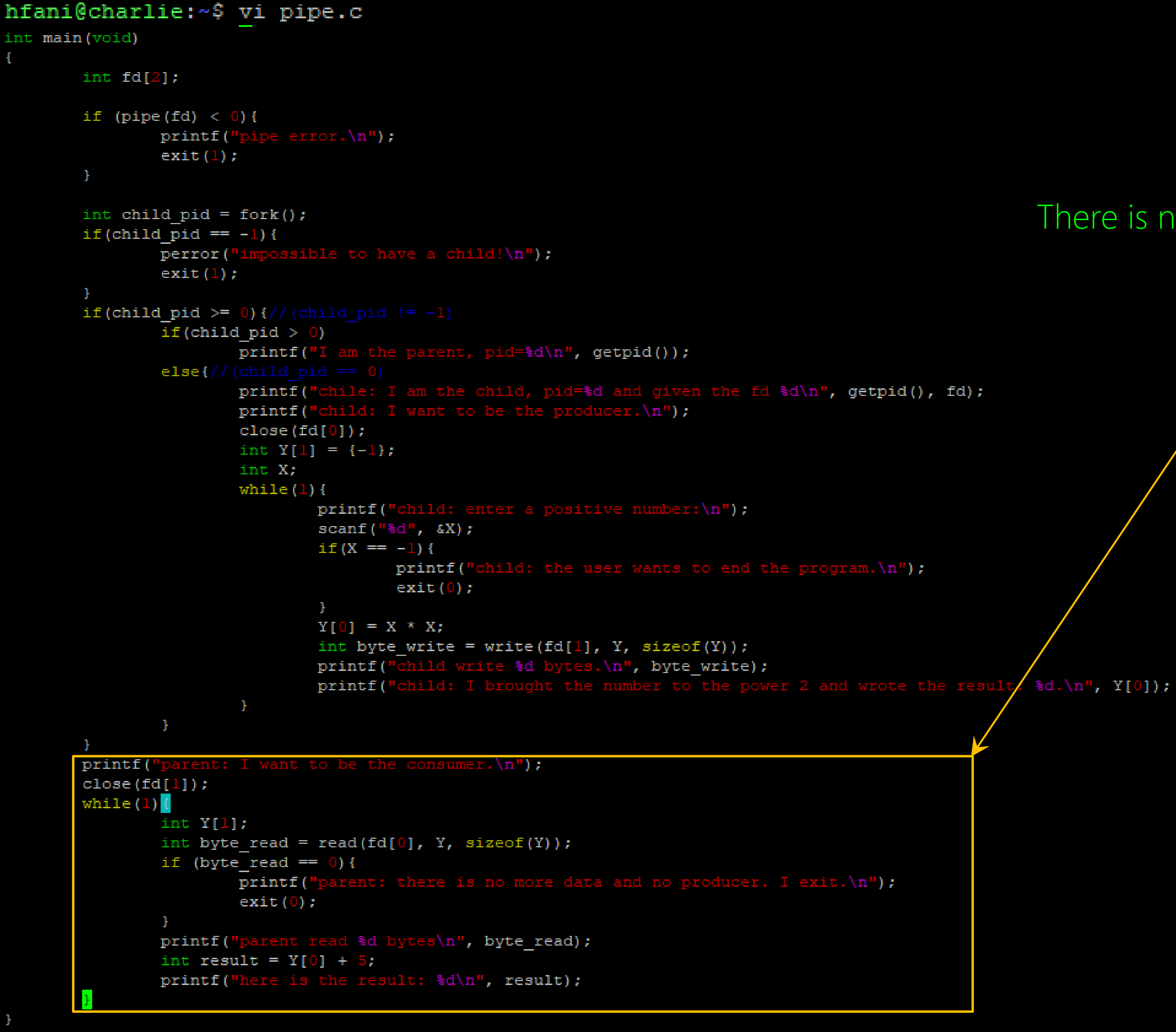


```
hfani@charlie:~$ vi pipe.c
int main(void)
{
        int fd[2];

        if (pipe(fd) < 0){
                printf("pipe error.\n");
                exit(1);
        }

        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!\n");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("chile: I am the child, pid=%d and given the fd %d\n", getpid(), fd);
                        printf("child: I want to be the producer.\n");
                        close(fd[0]);
                        int Y[1] = {-1};
                        int X;
                        while(1){
                                printf("child: enter a positive number:\n");
                                scanf("%d", &X);
                                if(X == -1){
                                        printf("child: the user wants to end the program.\n");
                                        exit(0);
                                }
                                Y[0] = X * X;
                                int byte_write = write(fd[1], Y, sizeof(Y));
                                printf("child write %d bytes.\n", byte_write);
                                printf("child: I brought the number to the power 2 and wrote the result: %d.\n", Y[0]);
                        }
                }
        }
        printf("parent: I want to be the consumer.\n");
        close(fd[1]);
        while(1){
                int Y[1];
                int byte_read = read(fd[0], Y, sizeof(Y));
                if (byte_read == 0){
                        printf("parent: there is no more data and no producer. I exit.\n");
                        exit(0);
                }
                printf("parent read %d bytes\n", byte_read);
                int result = Y[0] + 5;
                printf("here is the result: %d\n", result);
        }
}
```

If child wants to write but the pipe is full, it pauses

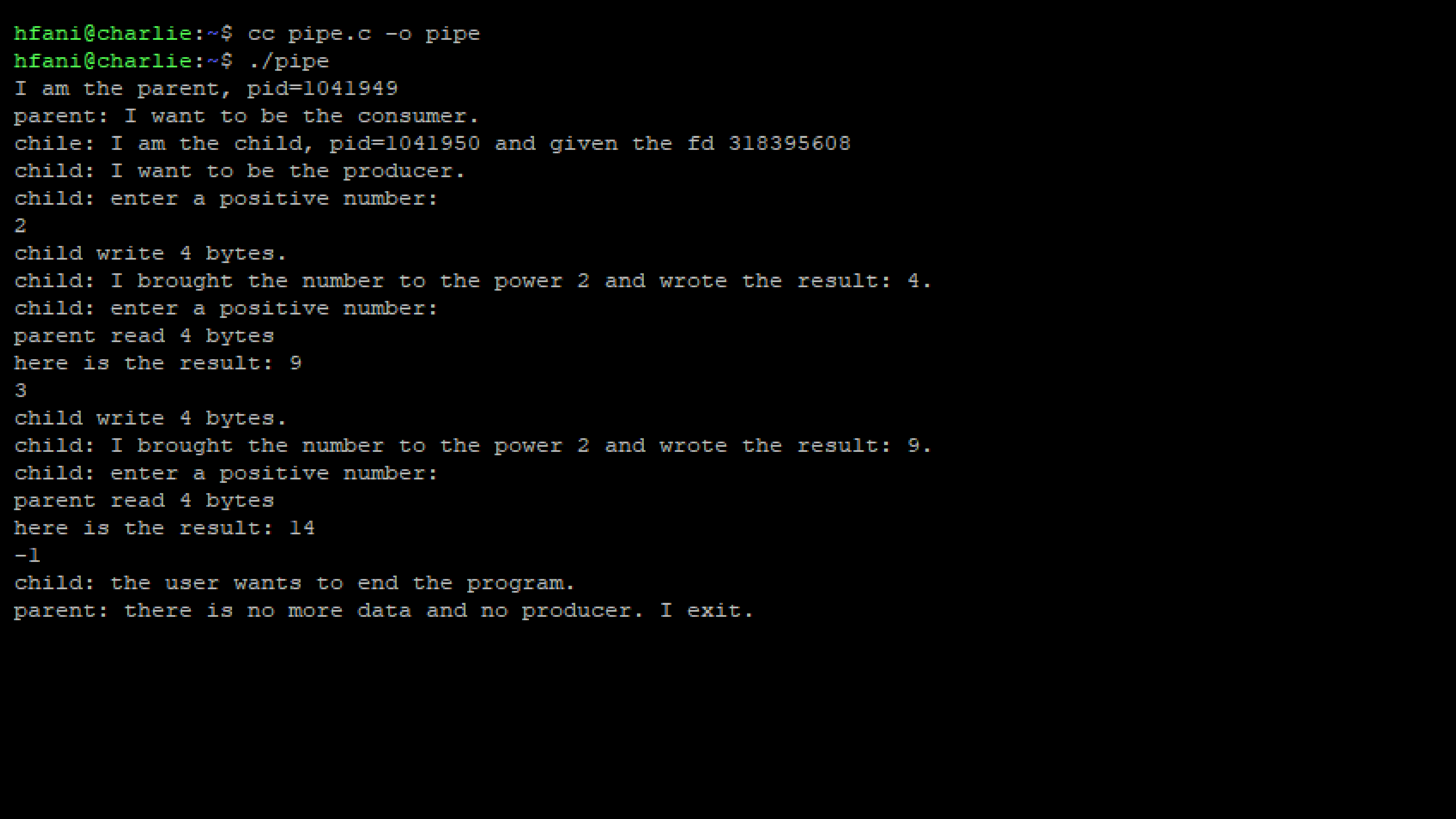
Synchronization

If parent wants to consume but there is no data, it pauses

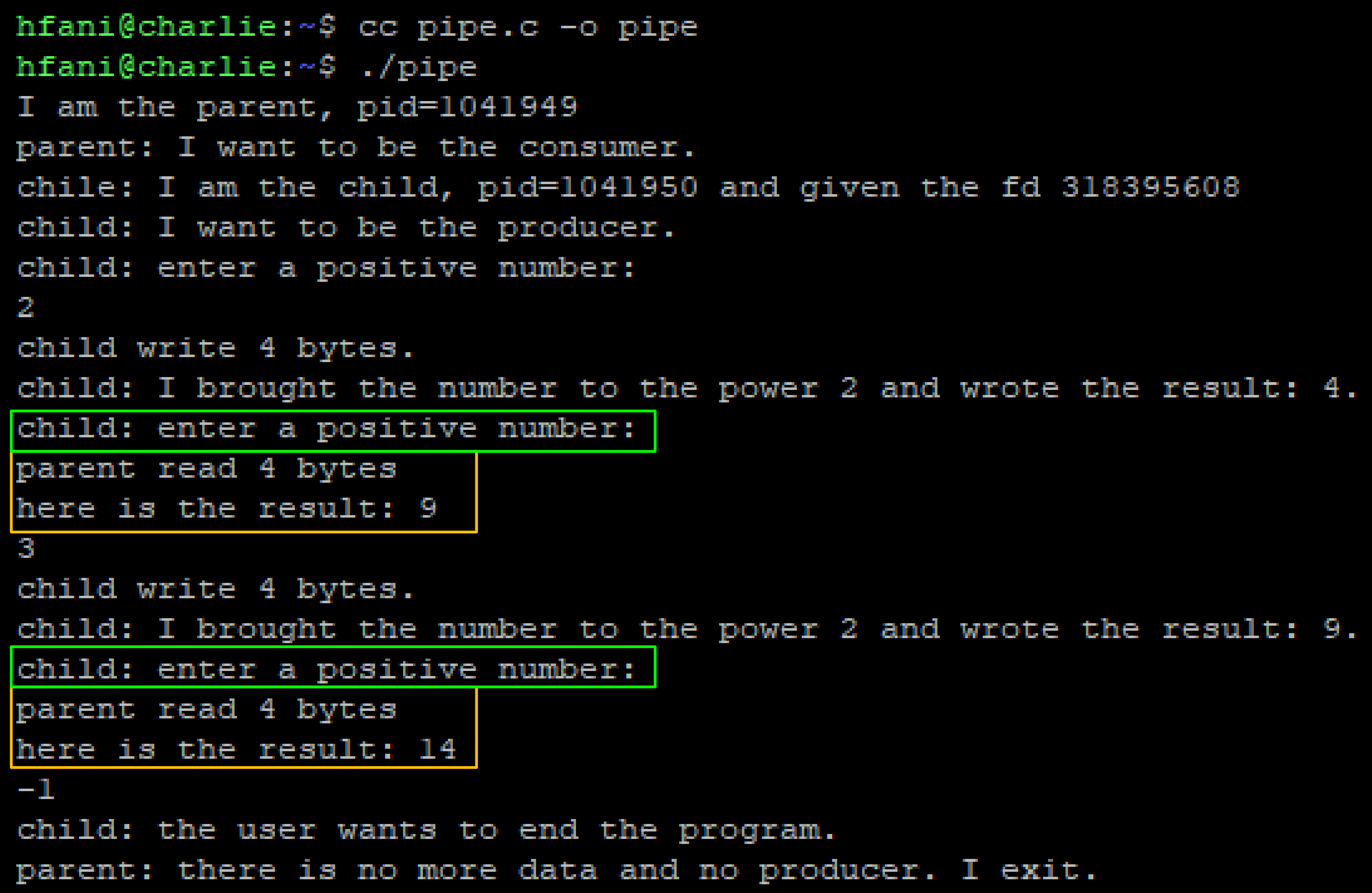


```
hfani@charlie:~$ vi pipe.c
int main(void)
{
        int fd[2];

        if (pipe(fd) < 0){
                printf("pipe error.\n");
                exit(1);
        }

        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!\n");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("chile: I am the child, pid=%d and given the fd %d\n", getpid(), fd);
                        printf("child: I want to be the producer.\n");
                        close(fd[0]);
                        int Y[1] = {-1};
                        int X;
                        while(1){
                                printf("child: enter a positive number:\n");
                                scanf("%d", &X);
                                if(X == -1){
                                        printf("child: the user wants to end the program.\n");
                                        exit(0);
                                }
                                Y[0] = X * X;
                                int byte_write = write(fd[1], Y, sizeof(Y));
                                printf("child write %d bytes.\n", byte_write);
                                printf("child: I brought the number to the power 2 and wrote the result: %d.\n", Y[0]);
                        }
                }
        }
        printf("parent: I want to be the consumer.\n");
        close(fd[1]);
        while(1){
                int Y[1];
                int byte_read = read(fd[0], Y, sizeof(Y));
                if (byte_read == 0){
                        printf("parent: there is no more data and no producer. I exit.\n");
                        exit(0);
                }
                printf("parent read %d bytes\n", byte_read);
                int result = Y[0] + 5;
                printf("here is the result: %d\n", result);
        }
}
```

There is no `wait()` system call for parent!

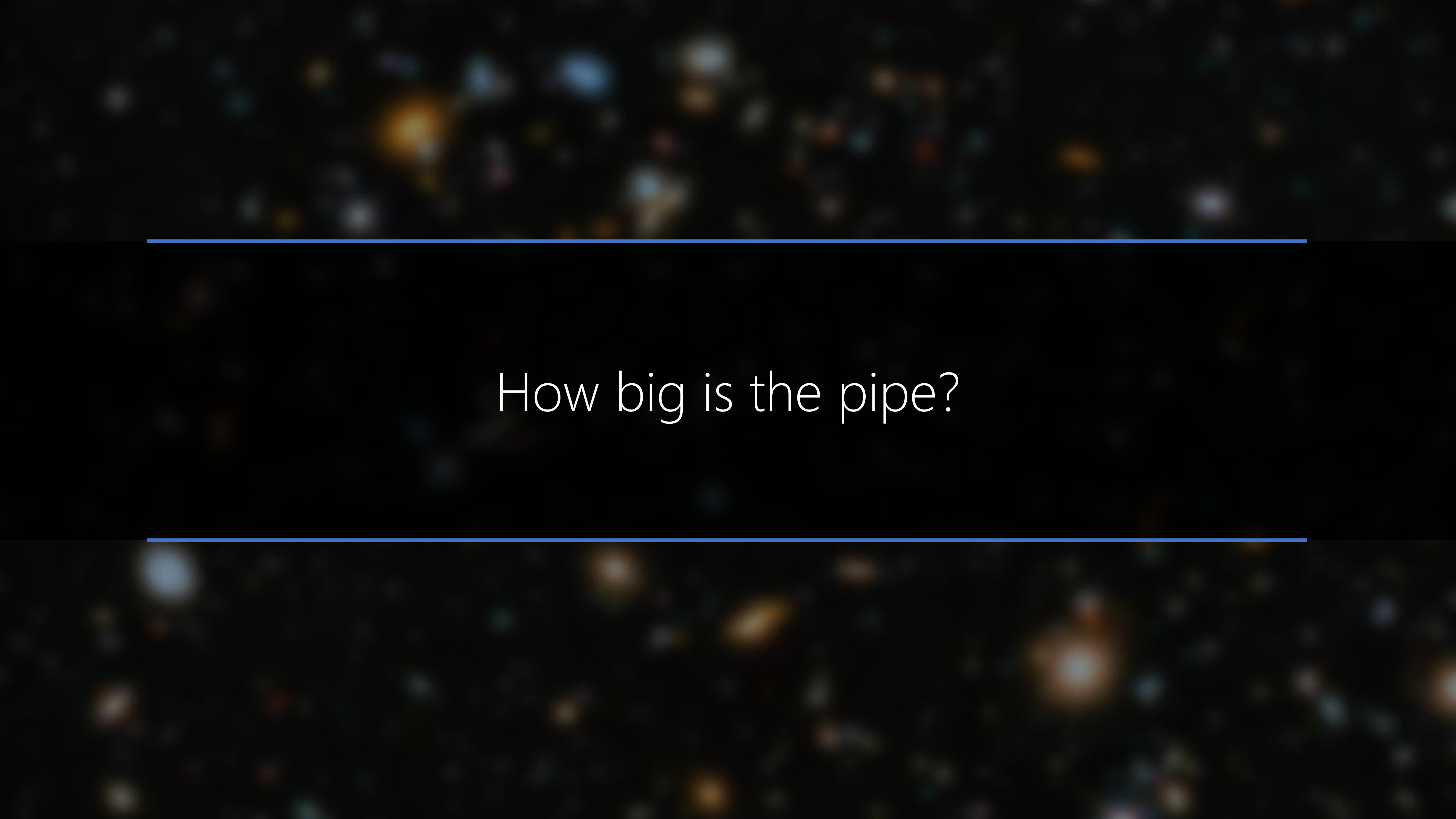


```
hfani@charlie:~$ cc pipe.c -o pipe
hfani@charlie:~$ ./pipe
I am the parent, pid=1041949
parent: I want to be the consumer.
chile: I am the child, pid=1041950 and given the fd 318395608
child: I want to be the producer.
child: enter a positive number:
2
child write 4 bytes.
child: I brought the number to the power 2 and wrote the result: 4.
child: enter a positive number:
parent read 4 bytes
here is the result: 9
3
child write 4 bytes.
child: I brought the number to the power 2 and wrote the result: 9.
child: enter a positive number:
parent read 4 bytes
here is the result: 14
-1
child: the user wants to end the program.
parent: there is no more data and no producer. I exit.
```



```
hfani@charlie:~$ cc pipe.c -o pipe
hfani@charlie:~$ ./pipe
I am the parent, pid=1041949
parent: I want to be the consumer.
chile: I am the child, pid=1041950 and given the fd 318395608
child: I want to be the producer.
child: enter a positive number:
2
child write 4 bytes.
child: I brought the number to the power 2 and wrote the result: 4.
child: enter a positive number:
parent read 4 bytes
here is the result: 9
3
child write 4 bytes.
child: I brought the number to the power 2 and wrote the result: 9.
child: enter a positive number:
parent read 4 bytes
here is the result: 14
-1
child: the user wants to end the program.
parent: there is no more data and no producer. I exit.
```

Appreciate the benefit of processor sharing: While the child is waiting for user input, the parent does the addition with 5



How big is the pipe?

Shell’s Pipe (vertical bar `|’)

```
top | grep hfani | {another program}
```

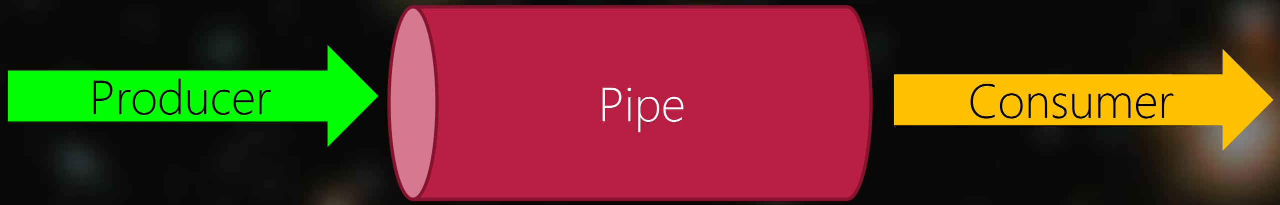
Named Pipe → FIFO

Like Pipe but …

```
mkfifo(const char *path, mode_t mode)
```

Pipe	FIFO
Unamed File, cannot be found in File System	Named File, should be **`open()`** like a regular to read or write
Between processes with the *same ancestor*	Between *any* processes
It is *deleted* after processes are terminated.	It *exists* even after processes termination. Should be explicitly deleted.

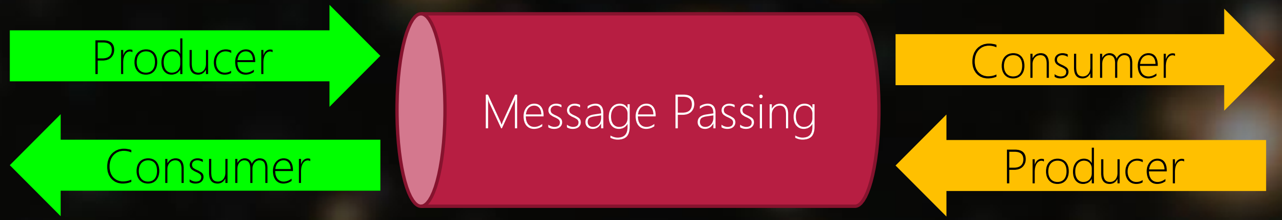
Half Duplex Conversation: One Talks, One Listens

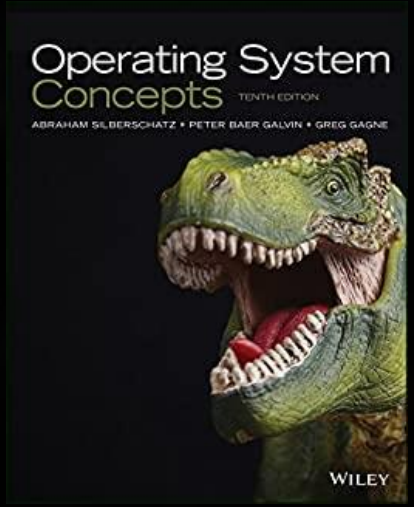


Full Duplex Conversation: Both Talk, Both Listen

if both talk at the same time? if both listen at the same time?

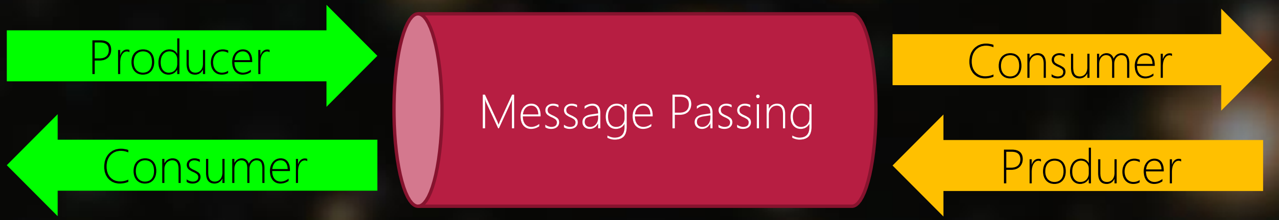
Advanced Synchronization: semaphore, mutex, ...





COMP3300: Operating Systems Fundamentals

Race Condition, Mutual Access
Semaphores, Mutex, …



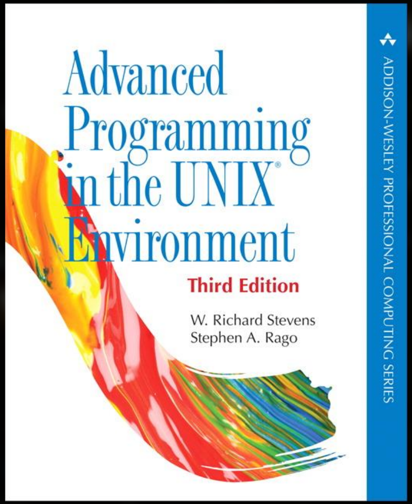
Producer

Consumer

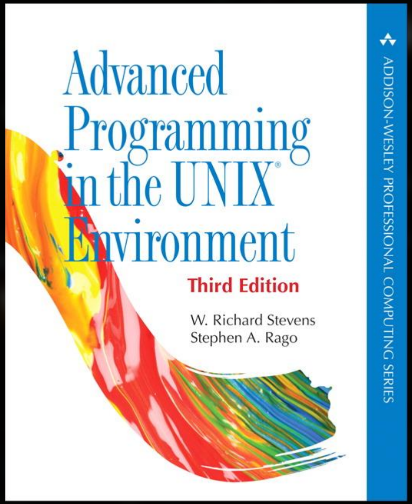
Message Passing

Consumer

Producer


Advanced Programming in the UNIX® Environment
Third Edition
W. Richard Stevens
Stephen A. Rago
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Chapter 16: Network IPC (Sockets)



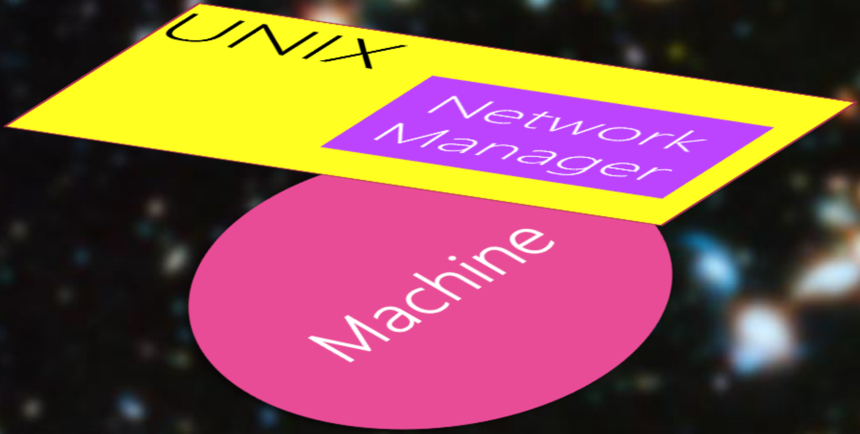
Advanced Programming in the UNIX® Environment

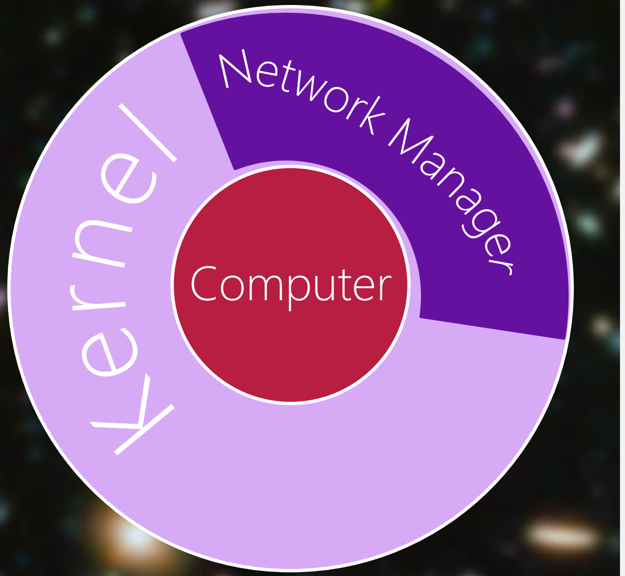
Third Edition

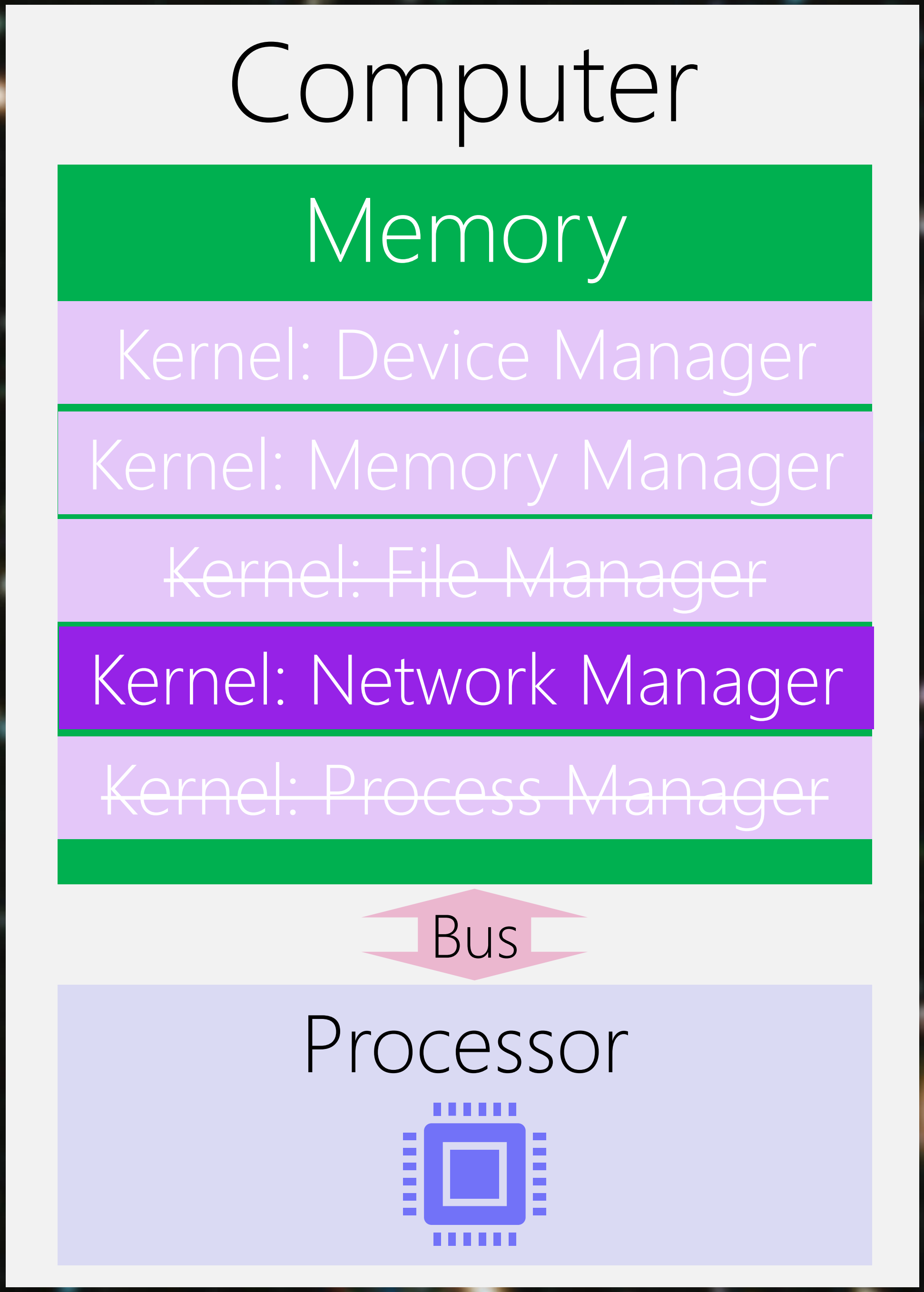
W. Richard Stevens
Stephen A. Rago

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Chapter 16: Network IPC (Sockets)






Computer

Memory

Kernel: Device Manager

Kernel: Memory Manager

~~Kernel: File Manager~~

Kernel: Network Manager

~~Kernel: Process Manager~~

Bus

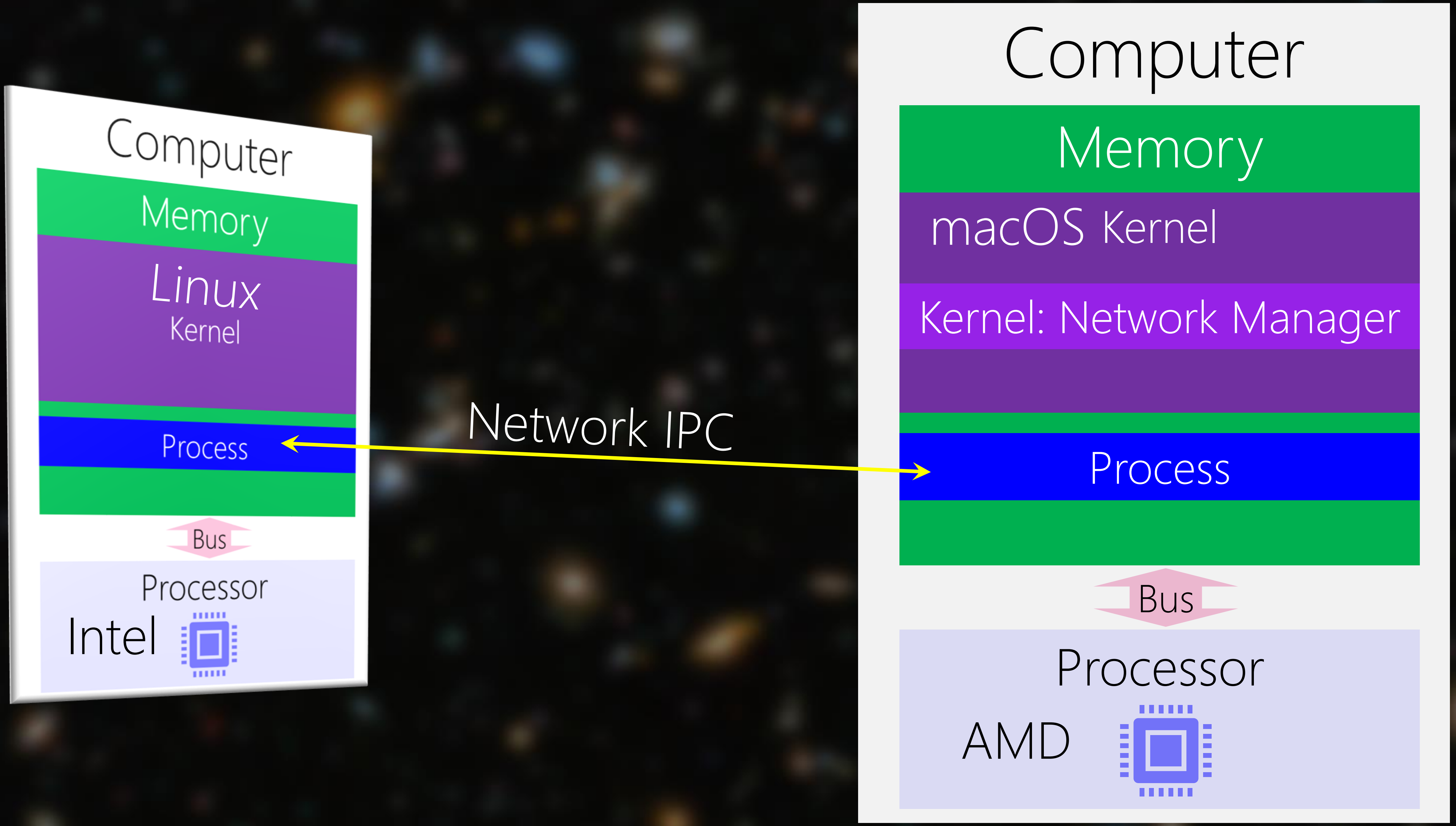
Processor

Multi~~processing~~ Computers

aka Computer Network

Multiple Single Processor ~~Multiprocessor~~

Step outside the computer system. Observe the World!



Computer

Memory

Linux

Kernel

Process

Bus

Processor

Intel

Network IPC

Computer

Memory

macOS Kernel

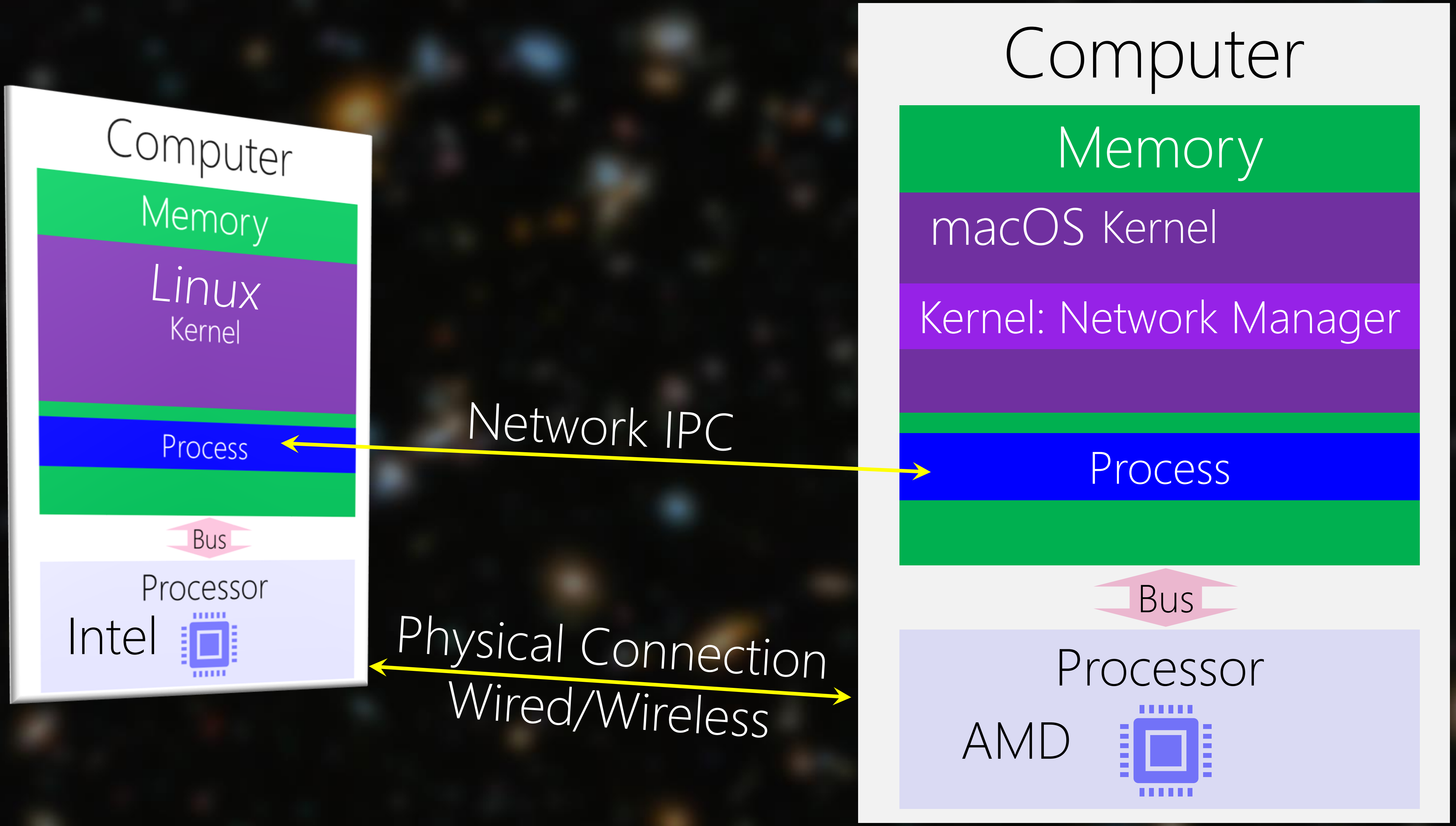
Kernel: Network Manager

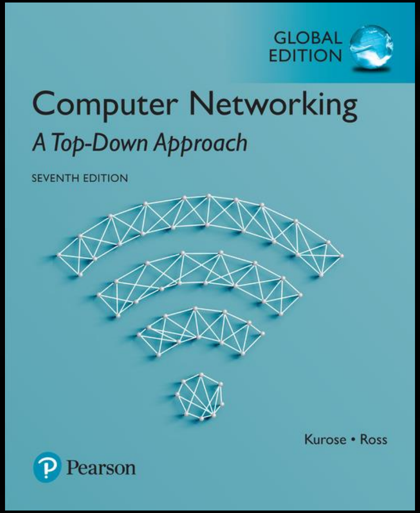
Process

Bus

Processor

AMD

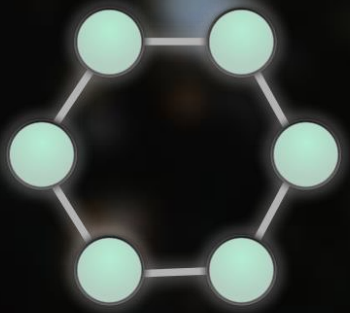

Computer
Memory
Linux
Kernel
Process
Bus
Processor
Intel
Network IPC
Physical Connection
Wired/Wireless
Computer
Memory
macOS Kernel
Kernel: Network Manager
Process
Bus
Processor
AMD



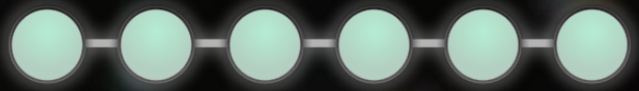
COMP3670: Computer Networks

Network Topology

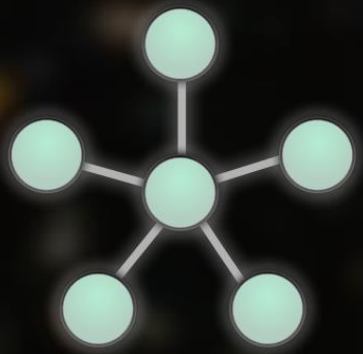
The way computer systems are connected.
By software control, we can convert one to another.



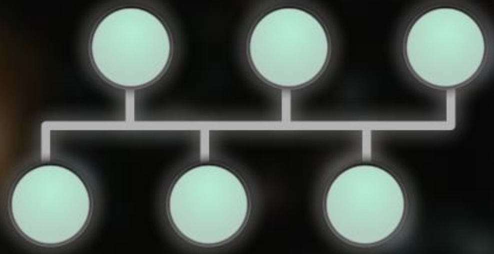
Ring



Line



Star



Bus

Network Protocol

Conversation Protocol between Computers
Language, Order (Who Talks, Who Listens), Addressing (Finding Each Other), ...

1975: 2-network communications between Stanford and University College London
1977: 3-network between sites in the US, the UK, and Norway