Parallel and Concurrent Programming CS3S666

FIFO / Named Pipes

Last Time…

- We used anonymous pipes to communicate between two processes.
- This was achieved by redirecting pipes to stdin and stdout using the dup() command.

This Time…

- We will investigate using named pipes or Fifo’s.

Anonymous Pipes

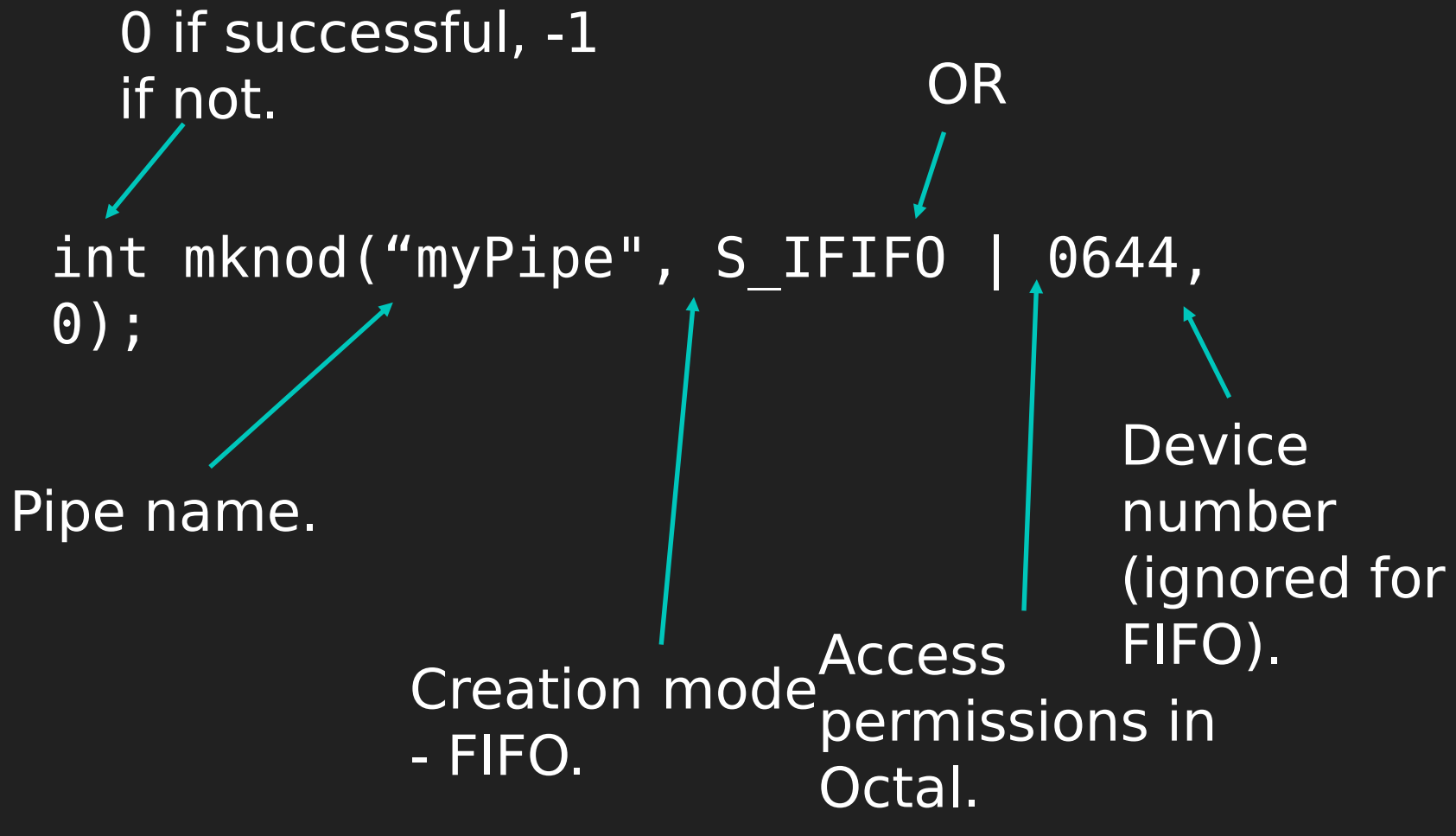
- We have used anonymous pipes between related processes.
  - (i.e. created using fork)

- What about if we want to pass data between unrelated processes?

Named Pipe

- A Named Pipe is also called a FIFO.
- It is like a pipe except it has a name.
  - It can very loosely be thought of as a file.

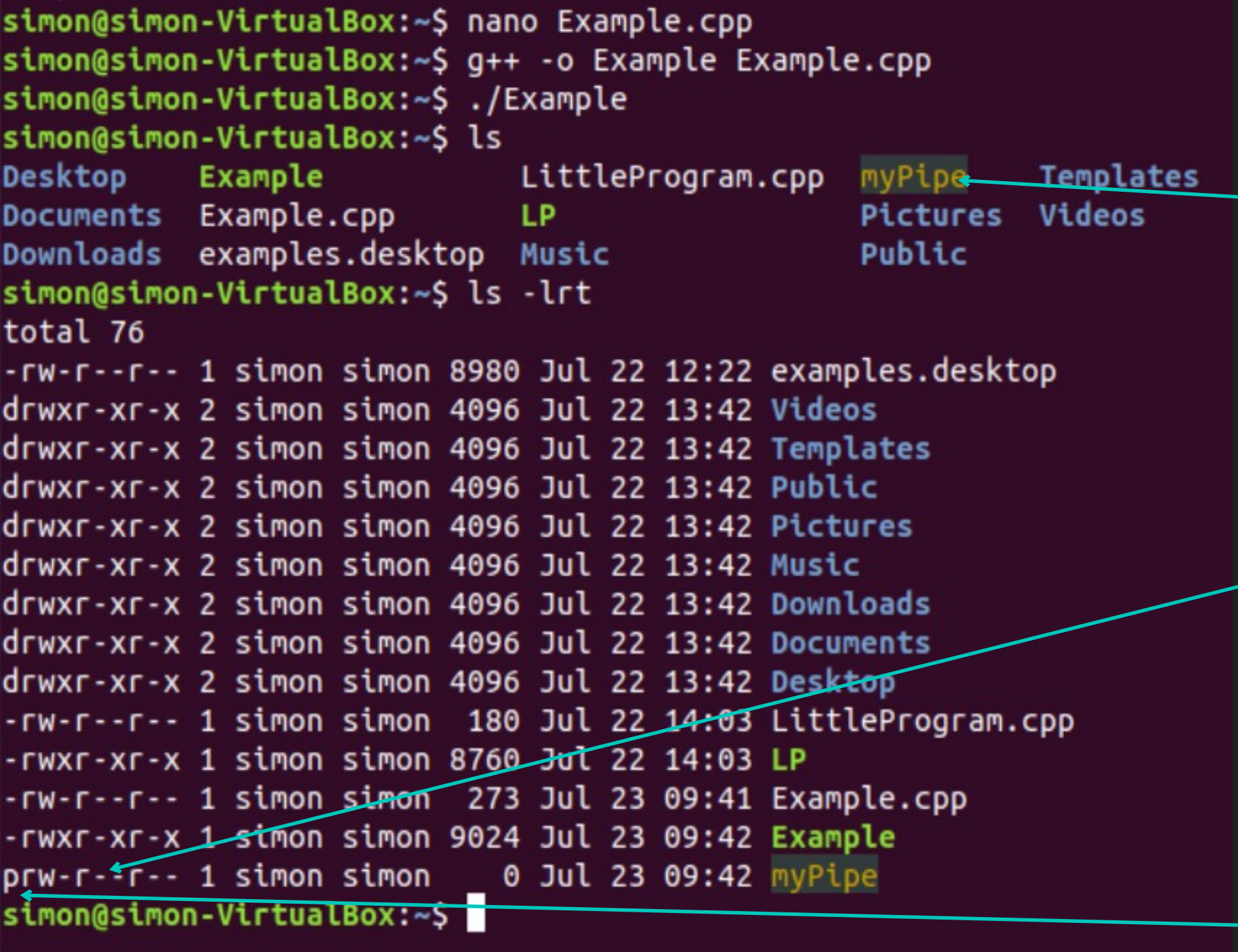
Creating a FIFO



Creating a Pipe

```
1  #include<cstdlib>
2  #include<iostream>
3  #include<unistd.h>
4  #include <sys/wait.h>
5  #include <sys/types.h>
6  #include <sys/stat.h>
7  #include <fcntl.h>
8
9  using namespace std;
10
11 int main(void)
12 {
13
14   if (mknod("myPipe", S_IFIFO | 0644, 0) < 0)
15     cout << "Error" << endl;
16
17   return(0);
18 }
19
```

Notice extra headers.



Pipe appears as file in directory it was created.

File permissions:

rw- r--

r+ 6 4 4

‘P’ indicates it’s a pipe.

Named Pipes

- Why not use a file?
- Named pipes do not actually store any data in the File.
- The file acts as a reference to the Pipe.
- A named Pipe can be accessed by multiple processes.

Named Pipes

- The pipe exists in the kernel when at least one process is making use of it.
- Data is directly passed between processes, without sitting on the hard drive.
- Once the data has been read it is lost.

More on Named Pipes

- Will fail if a pipe already exists.

Opening a Pipe to Write

```cpp
#include<cstdlib>
#include<iostream>
#include<unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

using namespace std;

int main(void)
{

  if (mknod("myPipe", S_IFIFO | 0644, 0) < 0)
    cout << "Pipe Already Created." << endl;

  cout << getpid() << endl;

  int fd = open("myPipe", O_WRONLY);

  cout << "File Opened" << endl;

  return(0);
}
```

```
simon@simon-VirtualBox:~$ nano Example.cpp
simon@simon-VirtualBox:~$ g++ -o Example Example.cpp
simon@simon-VirtualBox:~$ ./Example
Pipe Already Created
1531
```

Program hangs.

No file descriptor for the pipe.

```
simon@simon-VirtualBox:/proc/1531/fd$ ls -lrt
total 0
lrwx------ 1 simon simon 64 Jul 23 10:08 2 -> /dev/pts/0
lrwx------ 1 simon simon 64 Jul 23 10:08 1 -> /dev/pts/0
lrwx------ 1 simon simon 64 Jul 23 10:08 0 -> /dev/pts/0
simon@simon-VirtualBox:/proc/1531/fd$
```

Opening a Pipe to Write

- The open function blocks until a reader is created

Opening a Pipe to read

```
1  #include<cstdlib>
2  #include<iostream>
3  #include<unistd.h>
4  #include <sys/wait.h>
5  #include <sys/types.h>
6  #include <sys/stat.h>
7  #include <fcntl.h>
8
9  using namespace std;
10
11 int main(int argc, const char * argv[])
12 {
13   if (mknod("myPipe", S_IFIFO | 0644, 0) < 0)
14     cout << "Pipe Already Created." << endl;
15
16   cout << getpid() << endl;
17
18   int fd = open("myPipe", O_RDONLY);
19
20   cout << "File Opened to Read" << endl;
21   int x;
22   cin >> x;
23   return(0);
24 }
25
```

```
simon@simon-VirtualBox:~$ nano ExampleReader.cpp
simon@simon-VirtualBox:~$ g++ -o ExampleRead ExampleReader.cpp
simon@simon-VirtualBox:~$ ./ExampleRead
Pipe Already Created
1723
File Opened to read
```

Reading Program runs

```
simon@simon-VirtualBox:~$ ./Example
Pipe Already Created
1531
File Opened
simon@simon-VirtualBox:~$
```

The writing program from earlier stops hanging

Process has fd for pipe

```
simon@simon-VirtualBox:/proc/1723/fd$ ls -lrt
total 0
lr-x------ 1 simon simon 64 Jul 23 10:18 3 -> /home/simon/myPipe
lrwx------ 1 simon simon 64 Jul 23 10:18 2 -> /dev/pts/2
lrwx------ 1 simon simon 64 Jul 23 10:18 1 -> /dev/pts/2
lrwx------ 1 simon simon 64 Jul 23 10:18 0 -> /dev/pts/2
simon@simon-VirtualBox:/proc/1723/fd$
```

Sending data down a pipe

```
1  using namespace std;
2
3  int main(void)
4  {
5
6    if (mknod("myPipe", S_IFIFO | 0644, 0) < 0)
7      cout << "Pipe Already Created." << endl;
8
9    cout << getpid() << endl;
10
11   int fd = open("myPipe", O_WRONLY);
12
13   cout << "File Opened to write" << endl;
14
15   string str;
16
17   while (str[0] != 'q')
18   {
19     cout << "Enter something: " << endl;
20     cin >> str;
21     if (write(fd, str.c_str(), str.length()) <= 0)
22     {
23       str[0] = 'q';
24     }
25   }
26   close(fd);
27   cout << "Write End Closed" << endl;
28   return(0);
29 }
30
```

Will send EOF to Reader.

```
simon@simon-VirtualBox:~$ nano Example.cpp
simon@simon-VirtualBox:~$ g++ -o Example Example.cpp
simon@simon-VirtualBox:~$ ./Example
Pipe Already Created
1780
File Opened to write
Enter Something
12
Enter Something
1
Enter Something
hello
Enter Something
q
Write End Closed
```

```
simon@simon-VirtualBox:~$ nano ExampleReader.cpp
simon@simon-VirtualBox:~$ g++ -o ExampleRead ExampleReader.cpp
simon@simon-VirtualBox:~$ ./ExampleRead
Pipe Already Created
1894
File Opened to read
12 2
1 1
hello 5
q 1
Read end closed
simon@simon-VirtualBox:~$
```

Slight Diversion

- Microservices
- Use pipes (or queues) to move data around

Scenarios to Try

- Multiple Writers.
  - Multiple writers can send to one FIFO, and all messages will get to the reader in the order they are sent.
- Multiple Readers.
  - Message will be sent to one of the readers randomly, can’t broadcast same message to multiple readers.

Anonymous Pipe vs FIFO

- Anonymous pipe more light weight than FIFO.
- FIFO can be accessed by unrelated processes.
- FIFO does have non-blocking versions of write (means you can write when no one is listening).

Summary

- We have seen how to create and use a named pipe
- This ends a section of the work on processes, forks and pipes
- Quiz Time…

How would you create a child process?

POLL OPEN

ü
1. fork()

2. dup()

3. exec()

How would you execute another program in a separate process?

POLL OPEN

1. fork()

2. dup()

ü 3. exec()

What happens if you try to open a write to a FIFO that does not have a reader?

POLL OPEN

1. The process continues as normal

ü 2. The open function call blocks the process

3. program crashes

Next Time

- More on Interprocess Communication.
- Shared Memory.