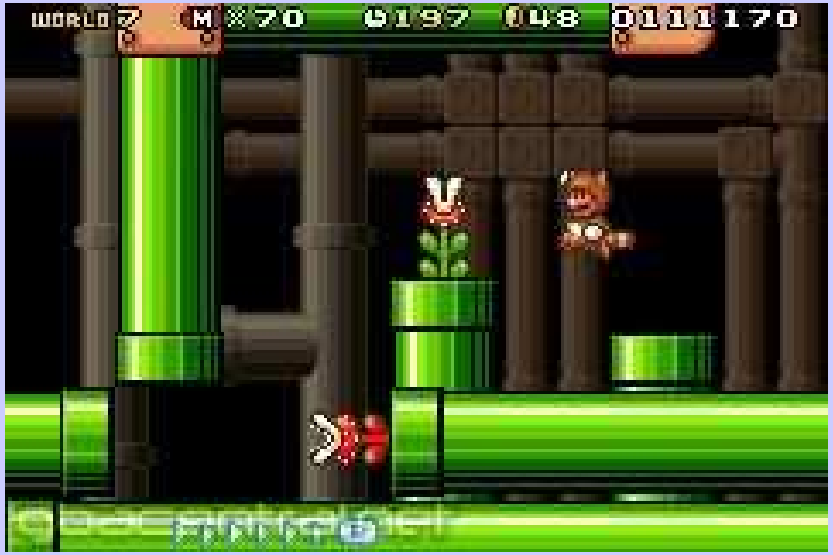
Pipes!

- Files
- read()
- write()
- Creating pipes
- Half-Duplex Pipes
- Full-Duplex Pipes



What is a File in Unix / Linux

Standard Error

Text Files

Hardware Devices

Pipes

Kitchen Sink

Standard Input

Links

Directories

Sockets

Hard Drives

Standard Output

To Clarify:

EVERYTHING IS A FILE IN UNIX!

I cin read()

```
ssize_t read(int file, void *buffer, size_t length)
```

On success, read() will return the number of bytes that were read. On failure, it will return -1.

The file that you wish to read from. This could be anything from a text file, to standard output (ie, the screen), to a pipe, to ...

This is a pointer to the location that you would like to store the data. This can be any datatype.

This is the size of the buffer, or the total number of bytes that you wish to read.

read() Example:

```
int main(void)
{
  char buffer[64];
  int i = 0;

  for(; i < 64; ++i)
    buffer[i] = 0;

  read(fileno(stdin), buffer, 63);

  printf("%s", buffer);
  return EXIT_SUCCESS;
}
```

Create a string buffer to store the data.

Set everything in the buffer to 0 (read() does not add the null terminator to strings)

Read data from standard input. (Note: The fileno() command converts from the stdio FILE structure to a Unix file number.)

write()

```
ssize_t write(int file, const void *buffer, size_t length)
```

On success, returns the number of bytes written to file. On failure, write() returns -1.

This is a pointer to the data you want to send into the file. Once again, this can be any datatype.

This is the file that you want to write to. It can be a text file, standard output, or any number of other file types.

This is the size of the buffer that you want to send. If this is a string, then this will be equal to the string length.

write() Example:

```
char *message = “Hello world!\n”;

int main(void)
{
  write(fileno(stdout),
        message,
        strlen(message));

  return EXIT_SUCCESS;
}
```

This is the message that you want to send.

This statement will write the message to standard output. Notice that the length specified is not long enough to include the null terminator for the string. Since we are not using printf(), the null terminator is not necessary.

What's all this hype about pipe()?

Pipes can be used to create a simple form of communication between processes. In a sense, creating a pipe is like opening the same file twice: once for reading, and once for writing. One process can then write information into one end of the pipe, allowing another process to retrieve the information from the other end.

```
int pipe write(int thePipe[2])
```

Returns -1 if the pipe cannot be created.

If `pipe()` is successful, then thePipe[2] will be an array of two file integers upon return. These will be explained in the next few slides...