Interprocess Communication:

Producer and Consumer Processes

For OS Internals and Design

Taught by: Dr. SMC

Written by: Capital Ex

Computer Science, Baccalaureate of Science

12/4/ 2019

Introduction

This project was developed, built, and test on Ubuntu Desktop 18.04 LTS. Pipes, message-passing, and shared memory will be implemented using the C programming language using GCC 7. Sockets will be implemented using the Rust programming language with version 1.30. Thus, all four interprocess communication methods (pipes, message-passing, sockets, and shared memory) will be shown.

Implementation

Pipes

System Calls

Table 1 contains the system calls used by this implementation.

Table 1 — System calls for piping IPC system

`pipe`	Communication System Call
`fork` `wait`	Process Control System Call

`close`
`write`
`read`
	Device Manipulation System Call

`fprintf`
`fclose`
`fopen`
	Device Manipulation System Call Wrapper

Setup

The setup for the Unix pipes requires a single file called `pipes.c`. The program can be built using GCC 7 using the following command: `gcc -Wall pipes.c -o pipes`. The code for `pipes.c` can be found in Figure 1.

```
 1  #include <sys/types.h>
 2  #include <stdio.h>
 3  #include <unistd.h>
 4  #include <stdlib.h>
 5  #include <time.h>
 6
 7  int main(void) {
 8      srand(time(NULL));
 9
10      int fd[2];
11      if (pipe(fd) == -1)
12          return -1;
13
14      pid_t pid = fork();
15      if (pid == 0) {
16          int output;
17          FILE *file = fopen("producer.txt", "w");
18
19          close(fd[0]);
20          for (int i = 0; i < 100; i++) {
21              output = rand();
```

```
22              write(fd[1], &output, sizeof(int));
23              fprintf(file, "produced: %d\n", output);
24          }
25          fclose(file);
26          close(fd[1]);
27          wait(NULL);
28      } else {
29          FILE *file = fopen("consumer.txt", "w");
30          int input = 0;
31
32          close(fd[1]);
33          while(read(fd[0], &input, sizeof(int)) != 0) {
34              fprintf(file, "consumed: %d\n", input);
35          }
36          fclose(file);
37          close(fd[0]);
38      }
39  }
```

Figure 1 — Complete source code for pipes.c program

Design

The program begins by initializing the random generator seeded with the current time on line 8, as shown in Figure 2. At lines 10 – 12, the programming initializes the pipe that will be used to pass data. At line 14, the program forks into the producer and consumer process.

```
 8      srand(time(NULL));
 9
10      int fd[2];
11      if (pipe(fd) == -1)
12          return -1;
13
14      pid_t pid = fork();
```

Figure 2 — Initialization code for pipes.c

The parent producer process will execute the code shown in Figure 3. The parent producer process will create an integer value called `output` on line 16 and a file pointer called `file` on line 17. The file `producer.txt` will be used to store the values created by the parent producer process. The parent producer process closes the read end of the pipe on line 19. The for-loop on line 20 executes the code on lines 21 – 23 one-hundred times. Each time the loop is executed, a new pseudo-random number is generated and stored in `output` on line 21. That value is then written to the pipe on line 22. Then the random number is written to the output file using `fprintf` on line 23. Once the loop exits, the `producer.txt` file is closed using `fclose` on line 25. Finally, the write end of the pipe is closed which inserts a `EOF` value into the pipe on line 26. This `EOF` value will cause the consumer to exit its loop. Finally, the parent process waits for the child consumer process to terminate.

```
16          int output;
17          FILE *file = fopen("producer.txt", "w");
18
19          close(fd[0]);
20          for (int i = 0; i < 100; i++) {
21              output = rand();
22              write(fd[1], &output, sizeof(int));
```

```
23              fprintf(file, "produced: %d\n", output);
24          }
25          fclose(file);
26          close(fd[1]);
27          wait(NULL);
```

Figure 3 — Code for the Unix pipe producer process

The consumer will execute the code on lines 29 – 37 shown in Figure 4. The consumer process opens a file for storing the values read from the pipe on line 29. The consumer also creates an integer variable to store a value from the pipe on line 30. At line 32, the consumer process closes the write end of the pipe. Line 33 begins a loop that executes line 34 until the read zero. The function read returns zero when the `EOF` value has been read from the pipe. Line 36 closes the `consumer.txt` file, and line 37 closes the read-end of the pipe, which concludes its execution.

```
29          FILE *file = fopen("consumer.txt", "w");
30          int input = 0;
31
32          close(fd[1]);
33          while(read(fd[0], &input, sizeof(int)) != 0) {
34              fprintf(file, "consumed: %d\n", input);
35          }
36          fclose(file);
37          close(fd[0]);
```

Figure 4 — Code for the Unix pipe consumer process

Results

Output is contained in Appendix A and is formatted into a two-columned numbered list for convenience.

Message Passing

System Calls

Table 2 contains the system calls used by this implementation.

Table 2 — System calls for message passing IPC system

`mq_open`
`mq_send`
`mq_close`
`mq_receive`
`mq_unlink`
	Communication System Call

`fork` `wait`	Process Control System Call
`fprintf` `fclose`	Device Manipulation System Call Wrapper

Setup

The setup for message passing requires a single C file: `message-passing.c`. The file will contain the code shown in Figure 5. The file `message-passing.c` can be compiled using the following command: `gcc -Wall message-passing.c -lrt -o message-passing`.

```c
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <mqueue.h>
#include <unistd.h>
#include <fcntl.h>
#include <time.h>
#define MSG_EMPTY 0
#define MSG_NUMBER 1

void producer(mqd_t queue);
void consumer(mqd_t queue);
typedef struct { char type; int value; } message_t;
typedef union { message_t message; char buffer[sizeof(message_t)]; } packet_t;

int main(void) {
    struct mq_attr attr = {
        .mq_flags = 0,
        .mq_maxmsg = 10,
        .mq_msgsize = sizeof(message_t),
        .mq_curmsgs = 0,
    };
    mqd_t queue = mq_open("/mq_project", O_CREAT | O_RDWR, 0666, &attr);
    srand(time(NULL));
    pid_t pid = fork();
    if (pid > 0) { producer(queue); }
    else { consumer(queue); }
}

void producer(mqd_t queue) {
    packet_t packet;
    FILE *f = fopen("producer.txt", "w");

    for (int i = 0; i < 100; i++) {
        packet.message.type = MSG_NUMBER;
        packet.message.value = rand();
        mq_send(queue, packet.buffer, sizeof(message_t), 0);
        fprintf(f, "consumed: %d\n", packet.message.value);
    }
    packet.message.type = MSG_EMPTY;
    mq_send(queue, packet.buffer, sizeof(message_t), 0);

    mq_close(queue);
    fclose(f);
    wait(NULL);
}

void consumer(mqd_t queue) {
    packet_t packet;
    FILE *f = fopen("consumer.txt", "w");

    mq_receive(queue, packet.buffer, sizeof(message_t), 0);
    while (packet.message.type != MSG_EMPTY) {
        fprintf(f, "produced: %d\n", packet.message.value);
        mq_receive(queue, packet.buffer, sizeof(message_t), 0);
    }

    mq_close(queue);
    fclose(f);
```

```
62      mq_unlink("/mq_project");
63  }
```

Figure 5 — Complete code for message-passing.c

Design

Lines 10 and 11 in Figure 6 define values representing the two message types. Line 15 defines a struct, `message_t`, to hold the message type and the integer value. Line 16 defines a union, `packet_t`, used to pass message structs from producer process to consumer process.

```
10  #define MSG_EMPTY 0
11  #define MSG_NUMBER 1
 …
15  typedef struct { char type; int value; } message_t;
16  typedef union { message_t message; char buffer[sizeof(message_t)]; } packet_t;
```

Figure 6 — Type definition for `message-passing.c`. Defines the message types; `message_t` struct, used to hold data about the message; and the `packet_t` union, used to send a `message_t` struct through `mq_send`

Lines 19 – 25 in Figure 7 initialize the message queue. The `attr` variable is used to define the attributes of the message queue. The queue is defined as having a size of 10 messages using `mq_maxmsg`, and each message is defined as having the size of one `message_t` data type using `mq_msgsize`. Next, the message queue is opened using `mq_open` on line 25. Then, the random number generator is seeded with the current time on 26. Finally, on line 27, the program uses the function `fork` to create the parent producer process and child consumer process.

```
19      struct mq_attr attr = {
20          .mq_flags = 0,
21          .mq_maxmsg = 10,
22          .mq_msgsize = sizeof(message_t),
23          .mq_curmsgs = 0,
24      };
25      mqd_t queue = mq_open("/mq_project", O_CREAT | O_RDWR, 0666, &attr);
26      srand(time(NULL));
27      pid_t pid = fork();
```

Figure 7 — Declaration of messaging queue attributes, open the message queue with read-write permissions for all, and forking of the parent process

The parent producer process executes the code contained in the function shown in Figure 8. It defines a packet struct to pass a message to the consumer process. Then on line 34, it uses `fopen` to create the file `producer.txt`. Then the for loop on line 36 executes the lines 37 – 40 one hundred times. Each time the loop is executed, the message type of the packet is set to `MSG_NUMBER`, and the message value is set to a randomly generated number by `rand`. Then the pack is sent using the `mq_send` system call, and the value generated is written to `producer.txt` using `fprintf`. Once the loop has completed, the producer sends a final packet of `MSG_EMPTY`. It then closes its message queue using `mq_close` and closes the file `producer.txt` with `fclose`. Finally, it uses the `wait` system call to pause execution until the consumer process finishes.

```
32  void producer(mqd_t queue) {
33      packet_t packet;
34      FILE *f = fopen("producer.txt", "w");
35
36      for (int i = 0; i < 100; i++) {
37          packet.message.type = MSG_NUMBER;
```

```
38          packet.message.value = rand();
39          mq_send(queue, packet.buffer, sizeof(message_t), 0);
40          fprintf(f, "produced: %d\n", packet.message.value);
41      }
42      packet.message.type = MSG_EMPTY;
43      mq_send(queue, packet.buffer, sizeof(message_t), 0);
44
45      mq_close(queue);
46      fclose(f);
47      wait(NULL);
48  }
```

Figure 8 — Code for the producer process using message passing queues

The child consumer process, once created, will execute the function shown in Figure 9. The child consumer process defines a variable called packet, and then it opens a text file called `consumer.txt` on lines 51 and 52. Next, on line 54, the producer process reads a message from the queue using the mq_receive communication system call. The loop on line 55 will run lines 56 and 57 until the message received from the queue is of type `MSG_EMPTY`. Each time the loop is ran, it first stores the received value to `consumer.txt`; then, it reads from the queue again. After the loop exits, the consumer process closes its connection to the queue using `mq_close` and closes its text file using `fclose`. Finally, the message queue is destroyed using `mq_unlink`.

```
50  void consumer(mqd_t queue) {
51      packet_t packet;
52      FILE *f = fopen("consumer.txt", "w");
53
54      mq_receive(queue, packet.buffer, sizeof(message_t), 0);
55      while (packet.message.type != MSG_EMPTY) {
56          fprintf(f, "consumed: %d\n", packet.message.value);
57          mq_receive(queue, packet.buffer, sizeof(message_t), 0);
58      }
59
60      mq_close(queue);
61      fclose(f);
62      mq_unlink("/mq_project");
63  }
```

Figure 9 — Code for the consumer process using message passing queues

Results

Output is contained in Appendix B and is formatted into a two-columned numbered list for convenience.

Sockets

System Calls

Table 3 contains the system calls used by this implementation.

Table 3 — System call wrappers used by Rust. Each wrapper function maps to a OS system call. For the case of TcpStream, each function has the same name as its system call. The functions with an asterisk are called automatically by the rust runtime

TcpStream::connect TcpStream::bind TcpStream::accept *TcpStream::shutdown	Communication System Call Wrappers

File::create write! println! *libc::close[1] read_to_string	Device Manipulation System Call Wrappers

Setup

First, Rust must be installed for Ubuntu. After installation, the project directors can be created. The commands shown in Figure 10 will create project folders for the producer process and for the consumer process. It is recommended that these commands are executed in an empty folder.

```
$ cargo init --bin producer
$ cargo init --bin consumer
```

Figure 10 — Commands to create the Rust project folders for the producer process and consumer process

Since pseudo random number generation isn't include in the rust standard library, the producer process requires extra change. The file `producer/Cargo.toml` must be edited to include the line `rand = "0.5.5"` below `[dependencies]` as shown in Figure 11.

```
1  name = "producer"
2  version = "0.1.0"
3  authors = ["unex"]
4
5  [dependencies]
6  rand = "0.5.5"
```

Figure 11 — Cargo.toml for the producer process

Next, the contents of `producer/src/main.rs` must be replaced with the code in Figure 12, and the contents of `consumer/src/main.rs` must be replaced with the code in Figure 13. Finally, the consumer process can be started by using `cargo run` in `consumer` director, and then producer process can be started using the same command in the `producer` directory. The consumer process must be started before the producer process.

Design

The producer process first creates a file in the parent directory called `producer.txt` on line 8. The method call `unwrap` causes the program to halt and display an error message if an issue occurs. This allows for error handling code to be exclude from the example. Next the producer process acquires an `rng` instance from the `rand` library on line 9. Then, the for-loop on line 10 executes the lines 11 – 14 one hundred times. Each time the loop is ran, it creates a new `TcpStream` connected to `127.0.0.1` on port `8888` on line 11. Then, it generates a random 64-bit signed integer value on line 12. Afterward, the value is formatted into a string and written to the `TcpStream` on line 13. Finally, that value is saved to the `producer.txt` file on line 14. Once the loop has finished, lines 16 and 17 connect to the consumer process one final time and send the value `finished`.

[1] The namespace `libc` is an internal wrapper for the Unix standard library. Additionally, `libc::close` is called by `FileDisc::drop` which closes a `std::fs::File`.

```
 1  extern crate rand;
 2  use std::net::TcpStream;
 3  use std::fs::File;
 4  use std::io::prelude::*;
 5  use rand::prelude::*;
 6
 7  fn main() {
 8      let mut results = File::create("../producer.txt").unwrap();
 9      let mut rng = rand::thread_rng();
10      for _ in 0..100 {
11          let mut stream = TcpStream::connect("127.0.0.1:8888").unwrap();
12          let x: i64 = rng.gen();
13          write!(stream, "{}", x).unwrap();
14          write!(results, "produced: {}\n", x).unwrap();
15      }
16      let mut stream = TcpStream::connect("127.0.0.1:8888").unwrap();
17      write!(stream, "finished");
18  }
```

Figure 12 — Source code for the producer client process

First, the consumer process uses the macro `println` to display a massage to `stdout` on line 6. Next, the consumer process creates a `TcpListener` on `127.0.0.1` using port `8888` on line 7. `TcpListener::bind` is a wrapper around a network communication system call. Then, the file `consumer.txt` is opened in the parent directory on line 8. On line 9, a new string is allocated to store the incoming data. On line 10, the consumer process enters a loop where it listens for incoming connections. Each connection is received using `TcpListener::accept` on Line 11. On line 12, any possible contents in the string are cleared. Then, on line 13, the consumer process reads from the socket using `read_to_string`. The function `read_to_string` stores the received value into a string variable. On line 14 – 18, the consumer process examines the data sent. The loop will exit if the data sent is `finished`, otherwise it uses the rust macro `write` to store that value to a file. Once the process exits, the `TcpListener` and file are both automatically closed.

```
 1  use std::net::TcpListener;
 2  use std::fs::File;
 3  use std::io::prelude::*;
 4
 5  fn main() {
 6      println!("Consumer process listening on 127.0.0.1:8888");
 7      let listener = TcpListener::bind("127.0.0.1:8888").unwrap();
 8      let mut results = File::create("../consumer.txt").unwrap();
 9      let mut string = String::new();
10      loop {
11          let (mut socket, _) = listener.accept().unwrap();
12          string.clear();
13          socket.read_to_string(&mut string).unwrap();
14          if string == "finished" {
15              break;
16          } else {
17              write!(results, "consumed: {}\n", string).unwrap();
18          }
19      }
20  }
```

Figure 13 — Source code for the consumer server process

Results

Output is contained in Appendix C and is formatted into a two-columned numbered list for convenience.

Shared Memory

System Calls

Table 3 contains the system calls used by this implementation.

Table 4 — System calls used by shared memory IPC implementation

smh_open mmap smh_unlink	Shared Memory	Communication System Calls
sem_open sem_wait sem_post sem_close sem_unlink	Semaphores	
fopen fclose ftruncate fprintf	Device Manipulation System Call Wrappers	

Setup

The shared memory method of interprocess communication requires five files: `buffer.h`, `buffer.c`, `makefile`, `producer.c`, and `consumer.c`. The file `buffer.h` will include the definition of the ring buffer data type, the declaration of `buffer_write`, and the declaration of `buffer_read`. The code for this header file is shown in Figure 14. The file `buffer.c` will contain the implementation of `buffer_write` and `buffer_read`. Code for `buffer.c` can be found in Figure 15. The project can be built using the `makefile` shown in Figure 16. The command `make build` will create the executable files for both the consumer and producer process.

```
 1  #include <stdbool.h>
 2  #define BUFFER_MAX_SIZE 10
 3
 4  typedef struct {
 5      int data[BUFFER_MAX_SIZE];
 6      int in;
 7      int out;
 8      int size;
 9  } buffer_t;
10
11  void buffer_write(buffer_t *, int);
12  int buffer_read(buffer_t *);
```

Figure 14 — Declarations of the buffer data structure and function to operate on it

```
 1  #include "buffer.h"
```

```
 2
 3  void buffer_write(buffer_t *buffer, int value) {
 4      buffer->data[buffer->in] = value;
 5      buffer->in = (buffer->in + 1) % BUFFER_MAX_SIZE;
 6      buffer->size = buffer->size + 1;
 7  }
 8
 9  int buffer_read(buffer_t *buffer) {
10      int value = buffer->data[buffer->out];
11      buffer->out = (buffer->out + 1) % BUFFER_MAX_SIZE;
12      buffer->size = buffer->size - 1;
13      return value;
14  }
```

Figure 15 — Implementation of `buffer_write` and `buffer_read`

```
 1  buffer.o: buffer.c
 2          gcc -Wall -c buffer.c
 3
 4  consumer.o: consumer.c
 5          gcc -Wall -c consumer.c
 6
 7  producer.o: producer.c
 8          gcc -Wall -c producer.c
 9
10  build: buffer.o consumer.o producer.o
11          gcc -o consumer consumer.o buffer.o -lrt -pthread
12          gcc -o producer producer.o buffer.o -lrt -pthread
```

Figure 16 — Makefile to properly compile the producer and consumer process

Design

The producer process will execute the code found in `producer.c` shown in Figure 17. First on line 13, the random number generator is seeded with the current time. The producer process on line 14 acquires the shared memory object through the `shm_open` system call. The size of the shared memory object is set using `ftruncate` on line 15. The shared object is truncated to the size of the ring buffer struct, `buffer_t`. Next, on line 16, the communication system call `mmap` is used to create the virtual memory mapping. The shared memory object is used to initialize it. Then on line 17 – 19 the semaphore values needed are created using communication system call `sem_open`. It creates semaphore objects with read-write permissions for user, group, and root (i.e: `0666`). The `mutex` semaphore is initialize to the value 1, the `empty` semaphore is initialized to the value `BUFFER_MAX_SIZE`, and the `full` semaphore is initialized to 0. The value `BUFFER_MAX_SIZE` is defined in `buffer.h` and is shown in Figure 14. The loop starting on line 22 is where the communication will begin. On each iteration of the loop, the producer generates a random number. Then, it calls the communication system call `sem_wait` on the `empty` semaphore. The `empty` semaphore suspends the producer process when the buffer becomes full. Next, the producer calls `sem_wait` on the `mutex` semaphore. The `mutex` semaphore insures that only one process will access the buffer at a time. Once the producer can acquire both the `empty` semaphore and `mutex` semaphore, it writes the randomly generated value to the buffer. Finally,

the producer releases the mutex semaphore and signals on the full semaphore using sem_post. The full semaphore prevents the buffer from underflowing by suspending the consumer process.

```
 1  #include "buffer.h"
 2  #include <stdlib.h>
 3  #include <stdio.h>
 4  #include <sys/mman.h>
 5  #include <sys/stat.h>
 6  #include <sys/types.h>
 7  #include <fcntl.h>
 8  #include <semaphore.h>
 9  #include <unistd.h>
10  #include <time.h>
11  #define MM_SIZE sizeof(buffer_t)
12  int main(void) {
13      srand(time(NULL));
14      int sfd = shm_open("/shared.mm", O_CREAT | O_RDWR, 0666);
15      ftruncate(sfd, MM_SIZE);
16      buffer_t *buf = (buffer_t *) mmap(NULL, MM_SIZE, PROT_WRITE, MAP_SHARED, sfd, 0);
17      sem_t *mutex = sem_open("/mutex", O_CREAT | O_RDWR, 0666, 1);
18      sem_t *empty = sem_open("/empty", O_CREAT | O_RDWR, 0666, BUFFER_MAX_SIZE);
19      sem_t *full = sem_open("/full", O_CREAT | O_RDWR, 0666, 0);
20      FILE *file = fopen("producer.txt", "w");
21
22      for (int i = 0; i < 100; i++) {
23          int x = rand();
24          sem_wait(empty);
25          sem_wait(mutex);
26
27          buffer_write(buf, x);
28
29          sem_post(mutex);
30          sem_post(full);
31
32          fprintf(file, "produced: %d\n", x);
33      }
34
35      fclose(file);
36      sem_close(mutex);
37      sem_close(empty);
38      sem_close(full);
39      return 0;
40  }
```

Figure 17 — Source code for producer.c

The consumer process will execute the code found in consumer. c shown in Figure 18. The lines 13 – 19 are the same from the producer process, minus the call to srand. This allows either process to be started first while outputting the same results. On line 20, the consumer process opens the text file consumer. txt which is used to store values from the buffer. The consumer process will read from the buffer 100 timers in the loop at line 22. On line 23, the consumer process creates a variable x to hold the value from the buffer. On line 24, the consumer process calls sem_wait on the full semaphore. This will suspend the consumer process if there are no values in the buffer. Next, the consumer waits on the mutex on line 25. Once it acquires both semaphores, the consumer process reads a value from the buffer on line 27.

Finally, the consumer process signals on the `mutex` semaphore and `empty` semaphore. The `empty` semaphore prevents the producer process from overflowing the buffer. Once the loop completes, the consumer process closes the `consumer.txt` file using `fclose`. Then the consumer process closes its connection to the `mutex`, `empty`, and `full` semaphores using the `sem_close` communication system call. Finally, the consumer process is responsible with cleaning up the semaphores and shared memory objects. The `sem_unlink` communication system call removes the semaphores from the computer, and `shm_unlink` communication system call removes the shared memory object from the computer.

```c
#include "buffer.h"
#include <stdlib.h>
#include <stdio.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <semaphore.h>
#include <unistd.h>
#define MM_SIZE sizeof(buffer_t)

int main(void) {
    int sfd =  shm_open("/shared.mm", O_CREAT | O_RDWR, 0666);
    ftruncate(sfd, MM_SIZE);

    buffer_t * buf = (buffer_t *) mmap(NULL, MM_SIZE, PROT_WRITE, MAP_SHARED, sfd, 0);
    sem_t *mutex = sem_open("/mutex", O_CREAT | O_RDWR, 0666, 1);
    sem_t *empty = sem_open("/empty", O_CREAT | O_RDWR, 0666, BUFFER_MAX_SIZE);
    sem_t *full = sem_open("/full", O_CREAT | O_RDWR, 0666, 0);
    FILE *file = fopen("consumer.txt", "w");

    for (int i = 0; i < 100; i++) {
        int x;
        sem_wait(full);
        sem_wait(mutex);

        x = buffer_read(buf);

        sem_post(mutex);
        sem_post(empty);

        fprintf(file, "consumed: %d\n", x);
    }

    fclose(file);
    sem_close(mutex);
    sem_close(empty);
    sem_close(full);

    sem_unlink("/mutex");
    sem_unlink("/empty");
    sem_unlink("/full");
    shm_unlink("/shared.mm");
    return 0;
}
```

Figure 18 — Source code for consumer.c

Results

Output is contained in Appendix D and is formatted into a two-columned numbered list for convenience.

Discussion

The Windows Linux Subsystem seemed to provide great compatibility for Linux. However, it doesn't implement the interfaces need for message passing, and semaphores. Therefore, I had to transfer my project to Linux to complete it.

While using the semaphores and shared memory, I learned that dealing with crashes could be a serious annoyance. Since the semaphore is stored as a file, it can stay locked after a crash. So, I had to delete the file to restart. I also learned that shared objects can be found in `/dev/shm/`.

Original, I had used the old System V IPC tools. The nice thing about this is that there is a command line tool to view all the shared object that you have (memory, semaphore, and mutexes). Unfortunately, the code for using them was far more complex. Additionally, I wanted to use what the Linux manual recommend.

Pipes were the simplest way to communicate between process. However, the book notes that they can be the slowest. Sadly, the time to test their speeds wasn't available.

Rust was chosen since it proved clean wrappers around the system calls (and didn't require `try-catch` blocks). Bu,t this require me to investigate the Rust source code to fully understand how the developers provided wrappers for the socket system calls, and the name of the Rust wrapper for a file close system call.

The results for all the processes were identical in pattern. Random numbers that were in the same order. Since, there was only two processes and all methods of reading were sequential, therefore all results should be in the same format: producer generates (`x, y, z`), consumer reads (`x, y, z`) in order. Thus, the results are relative uninteresting, but included none the less.

Learned a lot about constructing make files. The make file in Figure 16 originally didn't include the file name after the colon (i.e.: "`buffer.o:`" instead of "`buffer.o: buffer.c`"). So, every time I ran the script, it rebuilt all the files.

Appendix A

1. produced: 256634566
2. produced: 1271586491
3. produced: 1191118472
4. produced: 580493727
5. produced: 645085999
6. produced: 826644614
7. produced: 1127132834
8. produced: 2065737372
9. produced: 1634633211
10. produced: 799866789
11. produced: 1910821535
12. produced: 1289136987
13. produced: 11609378
14. produced: 34492715
15. produced: 1424224784
16. produced: 247007222
17. produced: 2140595995
18. produced: 1877340090
19. produced: 1605278790
20. produced: 663357881
21. produced: 556545363
22. produced: 1614342845
23. produced: 1407310438
24. produced: 1116379841
25. produced: 852793247
26. produced: 355393371
27. produced: 1360161839
28. produced: 1402468437
29. produced: 1135892381
30. produced: 1462436152
31. produced: 1894076550
32. produced: 1392526947
33. produced: 586538996
34. produced: 937711375
35. produced: 1973020675
36. produced: 1231624995
37. produced: 1764355989
38. produced: 952669861
39. produced: 1149878720
40. produced: 1251505552
41. produced: 1752536651
42. produced: 913216607
43. produced: 393158891
44. produced: 1764146029
45. produced: 947709323
46. produced: 1817383675
47. produced: 2011153252
48. produced: 940821670
49. produced: 1547240117
50. produced: 1468948394
51. produced: 1604179551
52. produced: 2103785481
53. produced: 935807591
54. produced: 864006341
55. produced: 1072681674
56. produced: 1788600838
57. produced: 1219399713
58. produced: 285359865
59. produced: 1043585627
60. produced: 207808446
61. produced: 1747796018
62. produced: 790178529
63. produced: 1600335393
64. produced: 186851366
65. produced: 1727889904
66. produced: 1425872420
67. produced: 1418476361
68. produced: 1344762246
69. produced: 231058634
70. produced: 420871433
71. produced: 448784150
72. produced: 1983595285
73. produced: 1334088041
74. produced: 841943042
75. produced: 1600257666
76. produced: 134313716
77. produced: 511843069
78. produced: 1463927270
79. produced: 1075135386
80. produced: 2059083187
81. produced: 785392016
82. produced: 531831289
83. produced: 2015385020
84. produced: 1721199607
85. produced: 1395837630
86. produced: 940583046
87. produced: 1362316797
88. produced: 467753695
89. produced: 1225942911
90. produced: 258418776
91. produced: 675562141
92. produced: 826255281
93. produced: 1048597306
94. produced: 128413887
95. produced: 1013106647
96. produced: 629003562
97. produced: 1554286307
98. produced: 284099361
99. produced: 1973765808
100. produced: 1785344941

Appendix A Cont.

1. consumed: 256634566
2. consumed: 1271586491
3. consumed: 1191118472
4. consumed: 580493727
5. consumed: 645085999
6. consumed: 826644614
7. consumed: 1127132834
8. consumed: 2065737372
9. consumed: 1634633211
10. consumed: 799866789
11. consumed: 1910821535
12. consumed: 1289136987
13. consumed: 11609378
14. consumed: 34492715
15. consumed: 1424224784
16. consumed: 247007222
17. consumed: 2140595995
18. consumed: 1877340090
19. consumed: 1605278790
20. consumed: 663357881
21. consumed: 556545363
22. consumed: 1614342845
23. consumed: 1407310438
24. consumed: 1116379841
25. consumed: 852793247
26. consumed: 355393371
27. consumed: 1360161839
28. consumed: 1402468437
29. consumed: 1135892381
30. consumed: 1462436152
31. consumed: 1894076550
32. consumed: 1392526947
33. consumed: 586538996
34. consumed: 937711375
35. consumed: 1973020675
36. consumed: 1231624995
37. consumed: 1764355989
38. consumed: 952669861
39. consumed: 1149878720
40. consumed: 1251505552
41. consumed: 1752536651
42. consumed: 913216607
43. consumed: 393158891
44. consumed: 1764146029
45. consumed: 947709323
46. consumed: 1817383675
47. consumed: 2011153252
48. consumed: 940821670
49. consumed: 1547240117
50. consumed: 1468948394
51. consumed: 1604179551
52. consumed: 2103785481
53. consumed: 935807591
54. consumed: 864006341
55. consumed: 1072681674
56. consumed: 1788600838
57. consumed: 1219399713
58. consumed: 285359865
59. consumed: 1043585627
60. consumed: 207808446
61. consumed: 1747796018
62. consumed: 790178529
63. consumed: 1600335393
64. consumed: 186851366
65. consumed: 1727889904
66. consumed: 1425872420
67. consumed: 1418476361
68. consumed: 1344762246
69. consumed: 231058634
70. consumed: 420871433
71. consumed: 448784150
72. consumed: 1983595285
73. consumed: 1334088041
74. consumed: 841943042
75. consumed: 1600257666
76. consumed: 134313716
77. consumed: 511843069
78. consumed: 1463927270
79. consumed: 1075135386
80. consumed: 2059083187
81. consumed: 785392016
82. consumed: 531831289
83. consumed: 2015385020
84. consumed: 1721199607
85. consumed: 1395837630
86. consumed: 940583046
87. consumed: 1362316797
88. consumed: 467753695
89. consumed: 1225942911
90. consumed: 258418776
91. consumed: 675562141
92. consumed: 826255281
93. consumed: 1048597306
94. consumed: 128413887
95. consumed: 1013106647
96. consumed: 629003562
97. consumed: 1554286307
98. consumed: 284099361
99. consumed: 1973765808
100. consumed: 1785344941

Appendix B

1. produced: 1804289383
2. produced: 846930886
3. produced: 1681692777
4. produced: 1714636915
5. produced: 1957747793
6. produced: 424238335
7. produced: 719885386
8. produced: 1649760492
9. produced: 596516649
10. produced: 1189641421
11. produced: 1025202362
12. produced: 1350490027
13. produced: 783368690
14. produced: 1102520059
15. produced: 2044897763
16. produced: 1967513926
17. produced: 1365180540
18. produced: 1540383426
19. produced: 304089172
20. produced: 1303455736
21. produced: 35005211
22. produced: 521595368
23. produced: 294702567
24. produced: 1726956429
25. produced: 336465782
26. produced: 861021530
27. produced: 278722862
28. produced: 233665123
29. produced: 2145174067
30. produced: 468703135
31. produced: 1101513929
32. produced: 1801979802
33. produced: 1315634022
34. produced: 635723058
35. produced: 1369133069
36. produced: 1125898167
37. produced: 1059961393
38. produced: 2089018456
39. produced: 628175011
40. produced: 1656478042
41. produced: 1131176229
42. produced: 1653377373
43. produced: 859484421
44. produced: 1914544919
45. produced: 608413784
46. produced: 756898537
47. produced: 1734575198
48. produced: 1973594324
49. produced: 149798315
50. produced: 2038664370
51. produced: 1129566413
52. produced: 184803526
53. produced: 412776091
54. produced: 1424268980
55. produced: 1911759956
56. produced: 749241873
57. produced: 137806862
58. produced: 42999170
59. produced: 982906996
60. produced: 135497281
61. produced: 511702305
62. produced: 2084420925
63. produced: 1937477084
64. produced: 1827336327
65. produced: 572660336
66. produced: 1159126505
67. produced: 805750846
68. produced: 1632621729
69. produced: 1100661313
70. produced: 1433925857
71. produced: 1141616124
72. produced: 84353895
73. produced: 939819582
74. produced: 2001100545
75. produced: 1998898814
76. produced: 1548233367
77. produced: 610515434
78. produced: 1585990364
79. produced: 1374344043
80. produced: 760313750
81. produced: 1477171087
82. produced: 356426808
83. produced: 945117276
84. produced: 1889947178
85. produced: 1780695788
86. produced: 709393584
87. produced: 491705403
88. produced: 1918502651
89. produced: 752392754
90. produced: 1474612399
91. produced: 2053999932
92. produced: 1264095060
93. produced: 1411549676
94. produced: 1843993368
95. produced: 943947739
96. produced: 1984210012
97. produced: 855636226
98. produced: 1749698586
99. produced: 1469348094
100. produced: 1956297539

Appendix B Cont.

1. consumed: 1804289383
2. consumed: 846930886
3. consumed: 1681692777
4. consumed: 1714636915
5. consumed: 1957747793
6. consumed: 424238335
7. consumed: 719885386
8. consumed: 1649760492
9. consumed: 596516649
10. consumed: 1189641421
11. consumed: 1025202362
12. consumed: 1350490027
13. consumed: 783368690
14. consumed: 1102520059
15. consumed: 2044897763
16. consumed: 1967513926
17. consumed: 1365180540
18. consumed: 1540383426
19. consumed: 304089172
20. consumed: 1303455736
21. consumed: 35005211
22. consumed: 521595368
23. consumed: 294702567
24. consumed: 1726956429
25. consumed: 336465782
26. consumed: 861021530
27. consumed: 278722862
28. consumed: 233665123
29. consumed: 2145174067
30. consumed: 468703135
31. consumed: 1101513929
32. consumed: 1801979802
33. consumed: 1315634022
34. consumed: 635723058
35. consumed: 1369133069
36. consumed: 1125898167
37. consumed: 1059961393
38. consumed: 2089018456
39. consumed: 628175011
40. consumed: 1656478042
41. consumed: 1131176229
42. consumed: 1653377373
43. consumed: 859484421
44. consumed: 1914544919
45. consumed: 608413784
46. consumed: 756898537
47. consumed: 1734575198
48. consumed: 1973594324
49. consumed: 149798315
50. consumed: 2038664370
51. consumed: 1129566413
52. consumed: 184803526
53. consumed: 412776091
54. consumed: 1424268980
55. consumed: 1911759956
56. consumed: 749241873
57. consumed: 137806862
58. consumed: 42999170
59. consumed: 982906996
60. consumed: 135497281
61. consumed: 511702305
62. consumed: 2084420925
63. consumed: 1937477084
64. consumed: 1827336327
65. consumed: 572660336
66. consumed: 1159126505
67. consumed: 805750846
68. consumed: 1632621729
69. consumed: 1100661313
70. consumed: 1433925857
71. consumed: 1141616124
72. consumed: 84353895
73. consumed: 939819582
74. consumed: 2001100545
75. consumed: 1998898814
76. consumed: 1548233367
77. consumed: 610515434
78. consumed: 1585990364
79. consumed: 1374344043
80. consumed: 760313750
81. consumed: 1477171087
82. consumed: 356426808
83. consumed: 945117276
84. consumed: 1889947178
85. consumed: 1780695788
86. consumed: 709393584
87. consumed: 491705403
88. consumed: 1918502651
89. consumed: 752392754
90. consumed: 1474612399
91. consumed: 2053999932
92. consumed: 1264095060
93. consumed: 1411549676
94. consumed: 1843993368
95. consumed: 943947739
96. consumed: 1984210012
97. consumed: 855636226
98. consumed: 1749698586
99. consumed: 1469348094
100. consumed: 1956297539

Appendix C

1. produced: 2793208351648120705
2. produced: -472829036233956617
3. produced: 3540374064039845040
4. produced: 7744624208487816746
5. produced: -4018731280968137768
6. produced: 7022827680573549712
7. produced: -290904733740753663
8. produced: -3951630927150252602
9. produced: -3762765417800154854
10. produced: 5310998184773233082
11. produced: -1730712357677906359
12. produced: -4607441656562819676
13. produced: 2517844301574023307
14. produced: 5191718906551239289
15. produced: 2996788475576540045
16. produced: 7689030276062423781
17. produced: -7101421479629207176
18. produced: -9131160107800897548
19. produced: 4680231119030263539
20. produced: 7440756006710175997
21. produced: -7988827380160529137
22. produced: 2861164944476554318
23. produced: 5796329330671212278
24. produced: -994166255031959357
25. produced: -648460531177893822
26. produced: 4228049761529838671
27. produced: 5717789117013829303
28. produced: -3916159591860692088
29. produced: 5511236938485799578
30. produced: -3281755637210742911
31. produced: 6749427755843152826
32. produced: 7514704443979499662
33. produced: 8967580043161341029
34. produced: -3087834817689043428
35. produced: -8684610802942080540
36. produced: 1489673056651211347
37. produced: 8177431869473636317
38. produced: -5317061166081381740
39. produced: 3668451675039355494
40. produced: -1292145770928744457
41. produced: -8009901606978375176
42. produced: 7080219295057732985
43. produced: 9103539406292912443
44. produced: -4628002797419771406
45. produced: 8534722947421205768
46. produced: -3921809515250480954
47. produced: 5318518377254102181
48. produced: 7233869046186765699
49. produced: -2364460483829337556
50. produced: -659192013160788840
51. produced: -6229775352551159713
52. produced: -2840877612480580385
53. produced: -5842861387856151888
54. produced: -7228691004094770392
55. produced: -6943929380442288825
56. produced: -8879261119436817819
57. produced: -4081203945477860265
58. produced: 5439272161899379272
59. produced: 4020289482227695398
60. produced: -1275609166876634099
61. produced: 797086644429029316
62. produced: -3441107462088697235
63. produced: -6076906420627197567
64. produced: 450211653107907627
65. produced: 4220177666868405026
66. produced: 2593485980005252187
67. produced: -5944740830628447934
68. produced: -8872574027250956647
69. produced: 2760068552388927726
70. produced: 604918533729935053
71. produced: -5902432503698013377
72. produced: -2901106260762350204
73. produced: 6308435728392813168
74. produced: -2672497288996403434
75. produced: 8146390179744521136
76. produced: -3371934668365130040
77. produced: -3365086855010099731
78. produced: 6649192389210753781
79. produced: -2438964783290529281
80. produced: -973507741805201806
81. produced: -5413499285524764466
82. produced: 2822924090364697178
83. produced: 8825492293106771938
84. produced: 5273193637990342526
85. produced: 2727770707469129404
86. produced: 5133619042697097151
87. produced: 7235731136265741871
88. produced: 8018693821570446653
89. produced: -2821155636609799925
90. produced: 8298819899548839443
91. produced: 5123296339744653983
92. produced: 704039390347233118
93. produced: -7669145651362269839
94. produced: -527370657132381385
95. produced: 1672207910969799227
96. produced: 2597175860058207879
97. produced: 7642211299283373264
98. produced: -3471307126939193075
99. produced: 9069552075111360207
100. produced: 325199137760688414

Appendix C Cont.

1. consumed: 793208351648120705
2. consumed: -472829036233956617
3. consumed: 3540374064039845040
4. consumed: 7744624208487816746
5. consumed: -4018731280968137768
6. consumed: 7022827680573549712
7. consumed: -290904733740753663
8. consumed: -3951630927150252602
9. consumed: -3762765417800154854
10. consumed: 5310998184773233082
11. consumed: -1730712357677906359
12. consumed: -4607441656562819676
13. consumed: 2517844301574023307
14. consumed: 5191718906551239289
15. consumed: 2996788475576540045
16. consumed: 7689030276062423781
17. consumed: -7101421479629207176
18. consumed: -9131160107800897548
19. consumed: 4680231119030263539
20. consumed: 7440756006710175997
21. consumed: -7988827380160529137
22. consumed: 2861164944476554318
23. consumed: 5796329330671212278
24. consumed: -994166255031959357
25. consumed: -648460531177893822
26. consumed: 4228049761529838671
27. consumed: 5717789117013829303
28. consumed: -3916159591860692088
29. consumed: 5511236938485799578
30. consumed: -3281755637210742911
31. consumed: 6749427755843152826
32. consumed: 7514704443979499662
33. consumed: 8967580043161341029
34. consumed: -3087834817689043428
35. consumed: -8684610802942080540
36. consumed: 1489673056651211347
37. consumed: 8177431869473636317
38. consumed: -5317061166081381740
39. consumed: 3668451675039355494
40. consumed: -1292145770928744457
41. consumed: -8009901606978375176
42. consumed: 7080219295057732985
43. consumed: 9103539406292912443
44. consumed: -4628002797419771406
45. consumed: 8534722947421205768
46. consumed: -3921809515250480954
47. consumed: 5318518377254102181
48. consumed: 7233869046186765699
49. consumed: -2364460483829337556
50. consumed: -659192013160788840
51. consumed: -6229775352551159713
52. consumed: -2840877612480580385
53. consumed: -5842861387856151888
54. consumed: -7228691004094770392
55. consumed: -6943929380442288825
56. consumed: -8879261119436817819
57. consumed: -4081203945477860265
58. consumed: 5439272161899379272
59. consumed: 4020289482227695398
60. consumed: -1275609166876634099
61. consumed: 797086644429029316
62. consumed: -3441107462088697235
63. consumed: -6076906420627197567
64. consumed: 450211653107907627
65. consumed: 4220177666868405026
66. consumed: 2593485980005252187
67. consumed: -5944740830628447934
68. consumed: -8872574027250956647
69. consumed: 2760068552388927726
70. consumed: 604918533729935053
71. consumed: -5902432503698013377
72. consumed: -2901106260762350204
73. consumed: 6308435728392813168
74. consumed: -2672497288996403434
75. consumed: 8146390179744521136
76. consumed: -3371934668365130040
77. consumed: -3365086855010099731
78. consumed: 6649192389210753781
79. consumed: -2438964783290529281
80. consumed: -973507741805201806
81. consumed: -5413499285524764466
82. consumed: 2822924090364697178
83. consumed: 8825492293106771938
84. consumed: 527319363799034252 6
85. consumed: 2727770707469129404
86. consumed: 5133619042697097151
87. consumed: 7235731136265741871
88. consumed: 8018693821570446653
89. consumed: -2821155636609799925
90. consumed: 8298819899548839443
91. consumed: 5123296339744653983
92. consumed: 704039390347233118
93. consumed: -7669145651362269839
94. consumed: -527370657132381385
95. consumed: 1672207910969799227
96. consumed: 2597175860058207879
97. consumed: 7642211299283373264
98. consumed: -3471307126939193075
99. consumed: 9069552075111360207
100. consumed: 325199137760688414

Appendix D

1. produced: 1646658347
2. produced: 2052291063
3. produced: 1779738665
4. produced: 1248496517
5. produced: 560191548
6. produced: 1634703514
7. produced: 753467058
8. produced: 572252244
9. produced: 1824662373
10. produced: 1593459037
11. produced: 986534051
12. produced: 1302254379
13. produced: 1577175842
14. produced: 1157883172
15. produced: 1255647601
16. produced: 237283432
17. produced: 1809764894
18. produced: 2047429296
19. produced: 1514148747
20. produced: 1653908683
21. produced: 1090325202
22. produced: 430519064
23. produced: 447179426
24. produced: 1561751364
25. produced: 2139153689
26. produced: 1660898032
27. produced: 1382089547
28. produced: 1393150280
29. produced: 1788689643
30. produced: 1927006535
31. produced: 145648055
32. produced: 1287864342
33. produced: 1831813950
34. produced: 1925386720
35. produced: 388877211
36. produced: 244521850
37. produced: 1412606587
38. produced: 1142344270
39. produced: 816774094
40. produced: 1089785312
41. produced: 588319659
42. produced: 1803308146
43. produced: 244556043
44. produced: 18011853
45. produced: 813707670
46. produced: 1500203644
47. produced: 255295286
48. produced: 475988916
49. produced: 1400149292
50. produced: 1769444033
51. produced: 2129897599
52. produced: 342990846
53. produced: 52479449
54. produced: 429593378
55. produced: 1904742211
56. produced: 44149491
57. produced: 2090491410
58. produced: 1139348110
59. produced: 1437299771
60. produced: 1731697406
61. produced: 918870997
62. produced: 1582947826
63. produced: 872078100
64. produced: 603201299
65. produced: 1360850899
66. produced: 1260955312
67. produced: 847723150
68. produced: 625973838
69. produced: 255815934
70. produced: 1664497244
71. produced: 1715759150
72. produced: 844135593
73. produced: 1320321742
74. produced: 1960315193
75. produced: 862147446
76. produced: 2134029413
77. produced: 1313035189
78. produced: 1117442732
79. produced: 462534681
80. produced: 565700834
81. produced: 739403118
82. produced: 444948633
83. produced: 908691680
84. produced: 791882567
85. produced: 874542011
86. produced: 665950243
87. produced: 836032058
88. produced: 817549773
89. produced: 1805298353
90. produced: 125848182
91. produced: 401763531
92. produced: 576685702
93. produced: 1708796008
94. produced: 1273841632
95. produced: 1179887002
96. produced: 922163259
97. produced: 387313296
98. produced: 2027610152
99. produced: 1548137097
100. produced: 643129230

Appendix D Cont.

1. consumed: 1646658347
2. consumed: 2052291063
3. consumed: 1779738665
4. consumed: 1248496517
5. consumed: 560191548
6. consumed: 1634703514
7. consumed: 753467058
8. consumed: 572252244
9. consumed: 1824662373
10. consumed: 1593459037
11. consumed: 986534051
12. consumed: 1302254379
13. consumed: 1577175842
14. consumed: 1157883172
15. consumed: 1255647601
16. consumed: 237283432
17. consumed: 1809764894
18. consumed: 2047429296
19. consumed: 1514148747
20. consumed: 1653908683
21. consumed: 1090325202
22. consumed: 430519064
23. consumed: 447179426
24. consumed: 1561751364
25. consumed: 2139153689
26. consumed: 1660898032
27. consumed: 1382089547
28. consumed: 1393150280
29. consumed: 1788689643
30. consumed: 1927006535
31. consumed: 145648055
32. consumed: 1287864342
33. consumed: 1831813950
34. consumed: 1925386720
35. consumed: 388877211
36. consumed: 244521850
37. consumed: 1412606587
38. consumed: 1142344270
39. consumed: 816774094
40. consumed: 1089785312
41. consumed: 588319659
42. consumed: 1803308146
43. consumed: 244556043
44. consumed: 18011853
45. consumed: 813707670
46. consumed: 1500203644
47. consumed: 255295286
48. consumed: 475988916
49. consumed: 1400149292
50. consumed: 1769444033
51. consumed: 2129897599
52. consumed: 342990846
53. consumed: 52479449
54. consumed: 429593378
55. consumed: 1904742211
56. consumed: 44149491
57. consumed: 2090491410
58. consumed: 1139348110
59. consumed: 1437299771
60. consumed: 1731697406
61. consumed: 918870997
62. consumed: 1582947826
63. consumed: 872078100
64. consumed: 603201299
65. consumed: 1360850899
66. consumed: 1260955312
67. consumed: 847723150
68. consumed: 625973838
69. consumed: 255815934
70. consumed: 1664497244
71. consumed: 1715759150
72. consumed: 844135593
73. consumed: 1320321742
74. consumed: 1960315193
75. consumed: 862147446
76. consumed: 2134029413
77. consumed: 1313035189
78. consumed: 1117442732
79. consumed: 462534681
80. consumed: 565700834
81. consumed: 739403118
82. consumed: 444948633
83. consumed: 908691680
84. consumed: 791882567
85. consumed: 874542011
86. consumed: 665950243
87. consumed: 836032058
88. consumed: 817549773
89. consumed: 1805298353
90. consumed: 125848182
91. consumed: 401763531
92. consumed: 576685702
93. consumed: 1708796008
94. consumed: 1273841632
95. consumed: 1179887002
96. consumed: 922163259
97. consumed: 387313296
98. consumed: 2027610152
99. consumed: 1548137097
100. consumed: 643129230