CS 341: Operating Systems, Spring 2018
Programming Assignment 1: A Producer-Consumer Problem

1 Introduction

Your assignment is to write two programs to implement a small producer-consumer problem. You will write a producer program that reads in a jumbled file and feeds the contents of that file to a separate consumer program through a buffer in shared memory. Your producer and your consumers will run as separate programs, so you will use mutexes to protect and manage shared data structures.

2 Producer: Input of Jumbled Data File

Your producer program will read a jumbled input file. We'll produce the jumbled input file, all your producer has to do is read the file and put it into a buffer in shared memory (see the section on shared memory below). The input files will be jumbled with every pair of bytes swapped. For example, an unjumbled file containing "abcdef" will be jumbled to "badcfe". Of course, we're going to feed you larger files than that.

The producer program should take the name of a jumbled input file as a command-line argument.

3 Consumer: Restoring the Data File

The consumer process will read the jumbled data provided by the producer process through the shared buffer. The consumer will reverse the simple jumbling and write the data to an output file.

The consumer program should take the name of the unjumbled output file as a command-line argument.

Requirement 1: there should be a minimum of BUSYWAITING in your code.

Requirement 2: there should be NO DEADLOCKS and NO RACE CONDITIONS in your code.

4 Program Start-up

It should be possible to start the producer and consumer in any order. If the producer is started first, it should create and initialize an empty buffer in shared memory, put as much data as it can into the shared buffer and wait for the consumer to start extracting data. If the consumer is started first, it should create and initialize an empty buffer in shared memory and then wait for the producer to start filling the shared buffer with data. The program that is started second, should locate the buffer in shared memory, attach itself to the shared buffer and then start filing or extracting data as appropriate.

5 Implementation

This program is an implementation of the Producer-Consumer problem where the producer may create or acquire large amounts of input (i.e. any amount of input data), but have only a small, fixed space to communicate the data to the consumers. Your program must be able to handle any amount of input data. To demonstrate proper functioning of the producer-consumer algorithm, your shared buffer may contain no more that 1024 bytes of data at a time. It it is not acceptable for the producer to simply read in all the data into a large buffer and quit, and then have the consumers simply read the data left behind by the now terminated producer.

6 Program Termination

After all in input has been processed, the producer and the consumer processes should shut down and then print termination messages.

7 Extra Credit

For extra credit, you can make a single program that splits into two processes. This program first creates the shared memory segment, attaches to the shared memory segment, initializes the shared buffer and then does a fork() to create a second process. One process becomes the producer, the other process becomes the consumer. Both processes continue the unjumbling as before.

8 What to turn in

- A writeup documenting your design **paying particular attention to the program synchronization requirements of your application**.
- All source code including both implementation (.c) and interface(.h) files.
- A makefile for producing an executable program file.

Your grade will be based on:

- Correctness (how well your code is working, including avoidance of deadlocks.
- Efficiency (avoidance of recomputation of the same results, avoidance of busy-waits).
- Good design (how well written your design document and code are, including modularity and comments).