CSS-430 : Operating Systems : HW05-HandsOn

This homework is a Hands-On session, aimed at having you experience the joys and frustrations of multi-thread programming.

**Assignment**

* Write a program that produces-into, and consumes-from, a large memory buffer (in effect, unbounded). Call that buffer gBuf. Make the buffer big enough to hold 1000 ints.
* Construct a global variable, called gNum, that holds the next number to be produced.
* Write a function insert(int x) that writes x into the next free slot in gBuf, and updates the buffer’s input index. Call that index gIn.
* Write a function int extract() that returns the number at the current output slot (gOut) in gBuf; it should update that output index
* Each producer thread reads the current value of gNum. It writes that value into gBuf using the insert function. It then increments the value and writes it back to gNum.
* The consumer thread reads the next value from gBuf using extract and prints it to the screen.
* Initialize gNum to contain the value 0
* Make the program stop when producer reaches some constant gMax

As you have no doubt worked out, global variables are named to start with “g”

**Requirements**

* The output to the screen should consist of the numbers 1 2 3 4 etc, in-order, with no repeats and no missing numbers. See the “say” statements below
* Stop when gNum hits gMax.
* Producers must not run past the end of gBuf
* Do not use busy-waiting, or spinlock

**Starter Code**

The code below is a starting point. You can download it from Canvas, at Files|Homework|HW05|Chaos

It contains no synchronization whatsoever.

#include <stdio.h>        // printf

#include <pthread.h>      // pthread\*

#define BUFSIZE 1000

#define gMax    20

int NUMPROD = 1;    // number of producers

int NUMCONS = 1;    // number of consumers

int gBuf[BUFSIZE];  // global buffer

int gNum = 0;       // global counter

int gIn  = 0;       // input  cursor in gBuf

int gOut = 0;       // output cursor in gBuf

void say(int me, char\* msg, int x) {

    printf("%d ", me);

    printf(msg);

    printf("%d \n", x);

}

void insert(int x) {

  gBuf[gIn] = x;

  ++gIn;

}

int extract() {

  int x = gBuf[gOut];

  ++gOut;

  return x;

}

int incgNum() {

  int num = ++gNum;

  return num;

}

int getgNum() {

  int num = gNum;

  return num;

}

void\* producer(void\* arg) {

  int me = pthread\_self();

  int num = getgNum();

  while (num < gMax) {

    num = incgNum();

    say(me, "Produced: ", num);

    insert(gNum);

  }

  return NULL;

}

void\* consumer(void\* arg) {

  int me = pthread\_self();

  while (true) {

    int num = extract();

    say(me, "Consumed: ", num);

    if (num == gMax) {

      break;

    }

  }

  return NULL;

}

void checkInput(int argc, char\* argv[]) {

  if (argc == 1) {

    NUMPROD = 1;

    NUMCONS = 1;

    return;

  }

  if (argc != 3) {

    printf("Specify <producers>  <consumer> \n");

    printf("Eg:  2  3 \n");

    exit(0);

  }

  NUMPROD = atoi(argv[1]);

  if (NUMPROD < 1 || NUMPROD > 10) {

    printf("Number of producers must lie in the range 1..10 \n");

    exit(0);

  }

  NUMCONS = atoi(argv[2]);

  if (NUMCONS < 1 || NUMCONS > 10) {

    printf("Number of consumers must lie in the range 1..10 \n");

    exit(0);

  }

}

int main(int argc, char\* argv[]) {

  checkInput(argc, argv);

  pthread\_t prod[NUMPROD];

  pthread\_t cons[NUMCONS];

  for (int i = 0; i < NUMPROD; ++i) pthread\_create(&prod[i], 0, producer, NULL);

  for (int i = 0; i < NUMCONS; ++i) pthread\_create(&cons[i], 0, consumer, NULL);

  for (int i = 0; i < NUMPROD; ++i) pthread\_join(prod[i], NULL);

  for (int i = 0; i < NUMCONS; ++i) pthread\_join(cons[i], NULL);

  printf("All done! Hit any key to finish \n");

  getchar();

  return 0;

}

Provide the number-of-producers, and number-of-consumers as arguments to the program. So, if your program compiles to chaos.exe, for example, then:

chaos 2 3 will launch the program with 2 producers and 3 consumers

If you launch the program without any arguments, it will default to 1 producer and 1 consumer

**Hints**

* Pthreads exists on Linux, Windows and macOS. So use whichever one you like best for debugging. But the programs will be run on Linux for grading.
* Start with the code above and see what happens
* It won’t meet the Requirements above . . . yet
* Add synchronization. Which variables are shared across threads. Are any readonly? Or can they all be updated?
* If the behavior is hopelessly confusing, try calling the producer and consumer as functions, rather than threads, to make debugging easier
* Does the program finish early? – before the consumer thread has printed all of 1 thru gMax? If so, how do you fix that?
* Try to step thru the code with an interactive debugger. Can your IDE *freeze* threads to make debugging less chaotic?
* If your program simply refuses to reach gMax before stopping, you probably have a thread stuck waiting for a value that will never arrive. Check for that.
* Write comments in your code about how you choose to synchronize. For example, you might add 2 mutexes – one to guard upates to gBuf, another to guard updates to gNum. If so, add a comment to that effect. It’s *very* hard to infer the synchronization rules by just looking at the code.
* Similarly, if you write a function which assumes its caller already holds some mutex, add a comment to that effect
* Expect to spend at least 2 hours on this homework

**What to Submit**

* Submit your solution in a file called **Chaos.c**

**Grading:**

The real point of this homework is to make you experience the challenge of multi-thread programming .

Grading this homework presents a challenge (for the grader). The solution is only about 100 lines of code. But it can go wrong in many ways. Please follow these steps (keep gMax = 20 throughout)

1. 7 points

1 producer, 1 consumer. Must report produced 1 thru gMax, and consumed 1 thru gMax, in order. Must not hang. Must report “All done! Hit any key to finish” and stop.

1. 1 point  
   1 producer, 2 consumers. Must report produced 1 thru 20, and consumed 1 thru gMax, in order. Must show 2 consumers in action. Must not hang. Must report “All done! Hit any key to finish” and stop.
2. 1 point  
   2 producers, 1 consumer. Must report produced 1 thru 20, and consumed 1 thru gMax, in order. Must show 2 produceers in action. Must not hang. Must report “All done! Hit any key to finish” and stop.
3. 1 point

3 producers, 2 consumers. Must report produced 1 thru 20, and consumed 1 thru gMax, in order. Must show 3 producers and 2 consumers in action. Must not hang. Must report “All done! Hit any key to finish” and stop.

If your solution uses busy-wait, or spinlock, you lose 1 point. If your solution reports numbers out-of-order, you lose 1 point. If your code produces jumbled output that is difficult to read, you lose 1 point.