CS 425 Project 2: Solving Producers-Consumers Problem with Pthread (100 points)

(due in the midnight of 11/29/25)

1 Problem Description

This is a team project, and each team can have up to two teammates. In this project, you will solve a *producers-consumers problem* with the pthread library, which is formulated as follows:

- there are multiple producers and multiple consumers;
- there is a bounded buffer shared among producers and consumers;
- there is an integer variable only shared among producers;
- each producer thread repeatedly inserts the current value of the interger variable to the bounded buffer if the buffer is not full, and then increases the value of the integer variable by one;
- each consumer thread repeatedly picks up an integer value from the bounded buffer and print the value such that integer values are printed consecutively, i.e, 0, 1, 2, ...up to a certain integer (specified by a user). To have a better idea of the behavior of consumers, the consumer ID should be display along with the buffered integers, showing which consumer thread prints which buffered integer. So the output looks like the following:

```
0, 1
1, 1
2, 1
3, 2
4, 2
5, 3
...
```

where the first column is a buffered integer, and the second column is a user-friendly consumer ID.

You need to apply the following **principles and concepts** to analyze the producers-consumers problem and design.

- minimize the number of shared variables.
- critical section problem: shared variables should be placed into a critical section, and mutual exclusion is applied to threads that are sharing the variable.
- busy waiting and blocking wait: acquiring a spinlock can lead to busy waiting, while calling a blocking wait operation of a semaphore won't.
- overflow issue and underflow issue: a producer should avoid overflow, while a consumer should avoid underflow.

- only non-shared variables of the main thread are needed to be passed into a producer or a consumer thread.

This project also includes an experimental assessment of the performance of busy waiting and blocking wait in varied settings, including buffer size, the number of producer threads, the number of consumer threads, the largest number to print. All these settings are input in the command line and stored in some variables. So the execution of your code should follow the user input format below:

```
./a.out <buffer_size> <num_producers> <num_consumers> <upper_limit>
```

Note that the buffer size in textbook is a constant, while it is a variable in this project.

2 Problem Analysis (10 pts)

First, analyze the problem. Never dive into implemention or coding befor you have a high-level view about the problem you are trying to solve. For this project, we analyze the problem by the following two steps:

1. (5 pts) analyze the relationship between producer-consumer problem and critical section problem. Relationship includes similarity and difference, more importantly, the hierachy.
2. (5 pts) analyze the difference between the producers-consumers problem and the producer-consumer problem.

3 Design (15 pts)

Analysis is about big idea, while design involves more details that can be theoretical, technical, and logical. It is important not note that design details should be independent to the choice of programming languages and libraries. Your design is graded by the following rubrics:

1. (3 pts) design shared variables and non-shared variables. Also pay attention to the difference between constant and variables. In the textbook, *BUFFER_SIZE* is a constant. But in this project, it should be designed as a variable.
2. (3 pts) design critical section of producer: how many critical sections should the producer function have, why?
3. (3 pts) design critical section of consumer: how many critical section should the consumer function have, why?
4. (3 pts) design semaphores. For example, a mutex semaphore is needed for some crtical section.
5. (3 pts) design data structure that needs to be passed into producer and consumer.

4 Implementation of producers-consumers problem (55 pts)

Implement your design in C/C++ language with the pthread library. You will use the C *struct* data type for the implementation of data passing from the main thread to a producer or consumer thread. The syntax of C *struct* is the following:

```
/*a user-defined struct data to encapsulate multiple piece information*/
struct <structname>
{
 <datatype1> <member1>;
 <datatype2> <member2>;
 ...
};
```

The following is a segment of sample code on how to pass struct data into a *producer* thread.

```
int main()
{
    // declaration of tid_producer and attr_producer;

    // declare and allocate a struct object
    struct v *data = (struct v *) malloc(sizeof(struct v));

    // set data members
    data->member1 = value1;
    data->member2 = value2;
    ...
    /* pass a struct data into a producer thread */
    pthread_create(&tid_producer, &attr_producer, producer, data);
    ...
}

void *producer(void *param)
{
  struct v *data;

  data = (struct v*) param; // cast the generic pointer param
  ...
}
```

Similar to Lab 3, you can use a *for* loop to create a number of producer threads, e.g.,

```
  for (i = 0; i < num_producers; i++) {
    // create data
    ...
    pthread_create(&tid_producer[i], &attr_producer[i], producer, data);
  }
```

The value of *num_producers* is input by a user from the keyboard. Parallelly, there should be variable named *num_consumers* whose value is input by a user from the keyboard too. Similar to Lab 3, you can use *pthread_join()* to join all threads (including producer and consumer threads).

4.1 major implementation (45 pts)

Your implementation will be graded based on the following rubrics:

- (15 pts) Implementation matches the design;
- (5 pts) correct initialization of variables;
- (5 pts) correct passing data to producers and consumer threads;
- (5 pts) correct use pthread library functions to create and join threads;
- (5 pts) correct implementation of synchronization contraints on producers and consumers, respectively;
- (5 pts) correct user input format, i.e., following the above specified format for user input;
- (5 pts) correct output format and output values, i.e., no gap, no duplication in the output sequence of intergers from 0 to a user-specific number; (In the context of process synchronization, correct output does not always imply the correctness of other categories.)

4.2 Critical section using spinlock (10 pts)

Now, you are going to revise your implementation regarding to critical section by replacing the binary semaphore *mutex* with a spinlock. The use of spinlock in *pthread* is pretty straightforward. The following code shows:

- how to declare a spinlock;
- how to initiate the value, where the parameter PTHREAD_PROCESS_PRIVATE indicates the lock is sharable of threads in the same process;
- how the spinlock is used in the entry section and exit section, respectively.

The use of pthread semaphores can be found in sample code.

```
#include <semaphore.h>
pthread_spinlock_t lock; // declare a spinlock

/* init a spinlock */
pthread_spin_init(&lock, PTHREAD_PROCESS_PRIVATE);

/* entry section */
pthread_spin_lock(&lock);

/*** critical section ***/

/* exit section */
pthread_spin_unlock(&lock);
```

This task should be relatively straightforward. You basically do the following things:

- replace the declaration of *mutex* by the declaration of a *spinlock*;
- replace the initization of *mutex* by the initialization of a *spinlock*;
- replace all *wait()* and *signal()* (i.e., *post()*) operations on *mutex* by *lock()* and *unlock()* operations on *spinlock*;

5 Experimental assessment of binary semaphore and spinlock (20 pts)

Before the experimental assessment, you need to disable the *printf()* statement in *consumer()* function since we are going to set relatively large value for *upper_limit* and don't want too many I/O operations. You can decide appropriate values for *upper_limit* adapting to the CPU speed of your computer. Values of *upper_limit* leading the elapse time ranging from 1 second to 30 seconds can be considered appropriate. Here the elapse time can be the elapse time of the first consumer that reaches the *upper_limit*, or the elapse time of the first producer that reaches the *upper_limit*.

Collect and present experimental results on elapse time under different setting of the following parameters:

1. Buffer size: 10, 20, 30, 40, 50, 100
2. Number of threads: (1,1), (1,5), (5, 1), etc.
3. Length of Critical Section:

To change the length of critical section, you can add the following code into the critical section:

```
for (j = 0; j < 1e6; j++);
```

Experimental results should be presented in tables and figures. Give analysis of the experimental results to conclude your comparison on mutex based and spinlock based solutions for the producer-consumer problem.

The evaluation will be graded based on completeness of data presentation and analysis, including:

- (5 pts) Table presentation is included;
- (5 pts) Figure presentation is included;
- (10 pts) in-depth analysis

On submission:
Submit a report and your source code to blackboard. The report should contain:

1. problem analysis
2. design
3. presentation and analysis of elapse time