# Report

Student ID：s3798993

Student Name: HengHao Li

# Github project name/url:

s3798993-OSP-2021-s1-Assign1-Mon1630/https://github.com/HenghaoLi/s3798993-OSP-2021-s1-Assign1-Mon1630.git

## The Producer-Consumer Problem

This experiment is about mutual exclusion and synchronization between producer and consumer. The problem refers to is the P and V operation. The experiment sets up a shared buffer, which is used mutually exclusive by the producer and the consumer, and when one thread uses the buffer, the other makes it wait until the previous thread releases the buffer.

Example first create two threads producer and consumer. producer Producer to the queue to continuously add elements to the end, consumer Consumer constantly consume elements in the queue, when the producer production reached 10 elements, the queue is full will be blocking has been waiting for space to add elements, at this time the consumer is still in the continuous consumption of elements in the queue . When the elements in the queue are empty, the thread blocks, and the consumer then pauses until the producer continues to add elements to the queue.

Suppose there are two threads, a producer and a consumer. The consumer runs first, acquires the lock, and then executes sem\_wait() on the full semaphore. Since there is no data yet, the consumer blocks and gives up the CPU, but, importantly, the consumer still holds the lock at this point.

Features of Producer-Consumer mode

Producers are guaranteed not to keep putting data into the buffer when it is full, and consumers are guaranteed not to consume data when the buffer is empty

When the buffer is full, the producer will go to sleep and will be woken up when the next consumer starts to consume data in the buffer; when the buffer is empty, the consumer will also go to sleep and will not be woken up until the producer adds data to the buffer.

Then the producer runs. If the producer can run, it can produce data and wake up the consumer thread. Unfortunately, it first calls sem\_wait() on the two-valued mutually exclusive semaphore. The lock is already held, so the producer is stuck as well.

Here there is a circular wait. The consumer holds the mutual exclusion and waits on the full semaphore. The producer can send the full signal, but is waiting on the mutual exclusion. Thus, the producer and consumer wait for each other - a typical deadlock.

feasible solution

To solve this problem, simply reduce the scope of the lock. As you can see, we adapted the operations of acquiring and releasing a mutual exclusion to be right next to the critical area. The result is a simple and effective bounded buffer, a common pattern for multithreaded programs.

## The Cigarette Smoker’s Problem

Problem description.

Suppose a system has three smoker processes and one Agent process. Each smoker keeps rolling a cigarette and smoking it, but to roll and smoke a cigarette, the smokers need three materials: tobacco, paper, and matches. The Agent process provides the three materials indefinitely, and each time the Agent puts two materials on the table, the smoker with the remaining material rolls a cigarette and smokes it, giving the provider a signal that it is done, at which point the provider puts the other two materials on the table, and so on (letting the three smokers take turns smoking). The smoker takes turns smoking).

Problem Analysis:

1) Relationship analysis. The supplier is in a synchronous relationship with each of the three smokers. Since the supplier cannot satisfy two or more smokers at the same time, the three smokers are mutually exclusive to the action of smoking (or learned by the three smokers taking turns to smoke).

2) Organize the ideas. Clearly there are 4 processes here. The supplier acts as a producer providing material to the three smokers.

3)Semaphore setup. The semaphores offerl, offer2, offer3 represent the resources for the combination of tobacco and paper, the resources for the combination of tobacco and matches, and the resources for the combination of paper and matches, respectively. The signal finish is used to synchronize the smoker and the supplier process. The supplier needs to wait for a smoker to finish smoking before offering the next resource, and the smoker has to tell the Agent through the finish signal that I have finished smoking.

Solution

Pseudocode

Set 0，1，2 Representing tobacco, paper and matches respectively semaphore S0 = 1;

semaphore S1 = 0;

semaphore S2 = 0;

semaphore S3 = 0;

int i = 0, j = 0;

cobegin

process Agent { // Agent

i = RAND() % 3;

j = RAND() % 3;

while(i == j) {

i = RAND() % 3;

j = RAND() % 3;

}

P(S0);

Put\_items[i]\_on\_table;

Put\_items[j]\_on\_table;

if((i=0 && j=1) || (i=1 && j=0))

V(S3);

else if((i=1 && j=2) || (i=2 && j=1))

V(S1);

else

V(S2)

}

process consumer\_k(k=1,2,3) {

while(true) {

P(S[k]);

take\_one\_item\_from\_table;

take\_one\_item\_from\_table;

V(S0);

make\_cigarette\_and\_smoking;

}

}