南京信息工程大学 实验（实习）报告

实验（实习）名称 生产者消费者问题的模拟实现 实验（实习）日期 得分 指导教师

系 计算机学院专业 物联网工程年级 2021 班次 2 姓名 学号

**一、实验目的**

通过编写和调试一个解决生产者–消费者问题的简单模拟程序，进一步深入理解课堂教学中讲授的进程同步问题，以及用于解决同步问题的信号量机制的基本思想，即通过研究进程同步和信号量机制实现生产者消费者问题的并发控制，以便阶段性地巩固学习成果。

**二、实验内容与步骤**

编制程序模拟解决生产者-消费者同步问题。具体设计要求:

有界缓冲区内设有20个存储单元,放入/取出的数据项设定为1-20这20个整型数。

(1)每个生产者和消费者对有界缓冲区进行操作后,即时显示有界缓冲区的全部内容,当前指针位置和生产者/消费线程的标识符。

(2)生产者和消费者各有两个以上。

(3)多个生产者或多个消费者之间须有共享对缓冲区进行操作的函数代码。

1. **实验步骤**

*# 导入线程库*

import threading

*# 定义有界缓冲区的大小*

BUFFER\_SIZE = 20

*# 定义有界缓冲区的列表*

buffer = [0] \* BUFFER\_SIZE

*# 定义有界缓冲区的头指针和尾指针*

head = 0

tail = 0

*# 定义有界缓冲区的互斥锁*

buffer\_lock = threading.Lock()

*# 定义有界缓冲区的空闲单元和满单元的信号量*

empty = threading.Semaphore(BUFFER\_SIZE)

full = threading.Semaphore(0)

*# 定义生产者的函数*

def producer(id):

    global buffer, head, tail

*# 生成1-20之间的随机数作为数据项*

    item = random.randint(1, 20)

*# 等待有界缓冲区有空闲单元*

    empty.acquire()

*# 获取有界缓冲区的互斥锁*

    buffer\_lock.acquire()

*# 将数据项放入有界缓冲区的尾部*

    buffer[tail] = item

*# 更新尾指针*

    tail = (tail + 1) % BUFFER\_SIZE

*# 显示有界缓冲区的全部内容，当前指针位置和生产者线程的标识符*

    print(f"Producer {id} produced {item}")

    print(f"Buffer: {buffer}")

    print(f"Head: {head}, Tail: {tail}")

*# 释放有界缓冲区的互斥锁*

    buffer\_lock.release()

*# 通知有界缓冲区有满单元*

    full.release()

*# 定义消费者的函数*

def consumer(id):

    global buffer, head, tail

*# 等待有界缓冲区有满单元*

    full.acquire()

*# 获取有界缓冲区的互斥锁*

    buffer\_lock.acquire()

*# 从有界缓冲区的头部取出数据项*

    item = buffer[head]

*# 将有界缓冲区的头部置为0*

    buffer[head] = 0

*# 更新头指针*

    head = (head + 1) % BUFFER\_SIZE

*# 显示有界缓冲区的全部内容，当前指针位置和消费者线程的标识符*

    print(f"Consumer {id} consumed {item}")

    print(f"Buffer: {buffer}")

    print(f"Head: {head}, Tail: {tail}")

*# 释放有界缓冲区的互斥锁*

    buffer\_lock.release()

*# 通知有界缓冲区有空闲单元*

    empty.release()

*# 定义主函数*

def main():

*# 定义生产者和消费者的数量*

    NUM\_PRODUCERS = 3

    NUM\_CONSUMERS = 4

*# 创建生产者和消费者的线程列表*

    producers = []

    consumers = []

*# 创建并启动生产者和消费者的线程*

    for i in range(NUM\_PRODUCERS):

        p = threading.Thread(target=producer, args=(i,))

        producers.append(p)

        p.start()

    for i in range(NUM\_CONSUMERS):

        c = threading.Thread(target=consumer, args=(i,))

        consumers.append(c)

        c.start()

*# 等待所有的线程结束*

    for p in producers:

        p.join()

    for c in consumers:

        c.join()

*# 调用主函数*

if \_\_name\_\_ == "\_\_main\_\_":

    main()

