操作系统作业一（进程线程）

120L020121 刘旭 2023.04.25

作业要求

设计一个包含至少3个线程的应用，需要使用全局共享数据、线程间通信、线程同步技术。代码不少于50行，并撰写报告（包含源代码、效果截图、功能设计说明、操作系统进程、线程和IPC知识理解和运用）。

源代码

```
import threading
import time
import random

# 全局共享数据，存储生产者生产的物品和消费者消费的物品
items = []

# 互斥锁，用于保护共享数据
lock = threading.Lock()

# 条件变量，用于线程间通信和同步
not_empty = threading.Condition(lock)
not_full = threading.Condition(lock)

# 消费者已消费的物品数量
consumed_items = 0


# 生产者线程类
class Producer(threading.Thread):
    def run(self):
        global items
        while True:
            # 检查是否已经消费了足够的物品
            if consumed_items >= x:
                print(f"[stop]      生产者 {self.name} 停止运行")
                break
            # 模拟生产物品的过程
            time.sleep(random.randint(1, 3))

            # 获取互斥锁
            lock.acquire()
```

```
            # 如果共享数据已满，则等待消费者消费
            while len(items) >= 10:
                not_full.wait()

            # 生产物品并添加到共享数据中
            item = random.randint(1, 100)
            items.append(item)
            print(f"[+]         生产者 {self.name} 生产了物品 {item}")

            print(f"[{consumed_items}]{items}")
            # 通知消费者线程有新物品可供消费
            not_empty.notify()

            # 释放互斥锁
            lock.release()


# 消费者线程类
class Consumer(threading.Thread):
    def run(self):
        global items, consumed_items
        while True:
            # 检查是否已经消费了足够的物品
            if consumed_items >= x:
                print(f"[stop]      消费者 {self.name} 停止运行")
                break

            # 模拟消费物品的过程
            time.sleep(random.randint(1, 3))

            # 获取互斥锁
            lock.acquire()

            # 如果共享数据为空，则等待生产者生产
            while not items:
                not_empty.wait()

            # 从共享数据中取出一个物品并消费
            item = items.pop(0)
            consumed_items += 1
            print(f"[-]         消费者 {self.name} 消费了物品 {item}")

            print(f"[{consumed_items}]{items}")
            # 通知生产者线程有空位可供生产
            not_full.notify()

            # 释放互斥锁
            lock.release()


# 设置消费者需要消费的物品数量
x = 10

print(f"当有{x}个物品被消费时，程序会结束运行。")
print(f"[consumed_items][items]\n")
```

```python
# 创建3个生产者线程和3个消费者线程并启动它们
for i in range(3):
    Producer().start()
    Consumer().start()

# 等待所有线程运行结束
for thread in threading.enumerate():
    if thread != threading.current_thread():
        thread.join()
```

效果截图

```
PS D:\VSCODE\OperatingSystem> & C:/Python310/python.exe
d:/VSCODE/OperatingSystem/Homework1/multi_threads.py
当有10个物品被消费时，程序会结束运行。
[consumed_items][items]

[+]         生产者 Thread-1 生产了物品 50
[0][50]
[-]         消费者 Thread-6 消费了物品 50
[1][]
[+]         生产者 Thread-5 生产了物品 9
[1][9]
[-]         消费者 Thread-2 消费了物品 9
[2][]
[+]         生产者 Thread-3 生产了物品 20
[2][20]
[+]         生产者 Thread-1 生产了物品 38
[2][20, 38]
[-]         消费者 Thread-6 消费了物品 20
[3][38]
[+]         生产者 Thread-5 生产了物品 28
[3][38, 28]
[-]         消费者 Thread-4 消费了物品 38
[4][28]
[-]         消费者 Thread-4 消费了物品 28
[5][]
[+]         生产者 Thread-5 生产了物品 24
[5][24]
[-]         消费者 Thread-2 消费了物品 24
[6][]
[+]         生产者 Thread-1 生产了物品 10
[6][10]
[-]         消费者 Thread-6 消费了物品 10
[7][]
[+]         生产者 Thread-3 生产了物品 93
[7][93]
[-]         消费者 Thread-4 消费了物品 93
[8][]
[+]         生产者 Thread-5 生产了物品 71
[8][71]
[-]         消费者 Thread-2 消费了物品 71
[9][]
```

```
[+]         生产者 Thread-1 生产了物品 84
[9][84]
[-]         消费者 Thread-6 消费了物品 84
[10][]
[stop]      消费者 Thread-6 停止运行
[+]         生产者 Thread-3 生产了物品 74
[10][74]
[stop]      生产者 Thread-3 停止运行
[-]         消费者 Thread-4 消费了物品 74
[11][]
[stop]      消费者 Thread-4 停止运行
[+]         生产者 Thread-5 生产了物品 27
[11][27]
[stop]      生产者 Thread-5 停止运行
[-]         消费者 Thread-2 消费了物品 27
[12][]
[stop]      消费者 Thread-2 停止运行
[+]         生产者 Thread-1 生产了物品 86
[12][86]
[stop]      生产者 Thread-1 停止运行
```

功能设计说明

该程序实现了一个简单的生产者-消费者模型，其中多个生产者线程和多个消费者线程共享同一个列表作为缓冲区。 程序中的生产者线程通过获取互斥锁和条件变量，向缓冲区中添加物品并通知消费者线程有新物品可供消费；消费者线程则通过获取互斥锁和条件变量，从缓冲区中取出物品并通知生产者线程有空位可供生产。 该程序使用了`Python`的`threading`模块实现多线程操作，并使用了互斥锁和条件变量来实现线程同步和通信，以保证共享数据的正确访问和操作。 在程序开始运行时，用户可以设置消费者需要消费的物品数量`x`，当消费者已经消费了`x`个物品时，程序会结束运行。 程序中同时创建了3个生产者线程和3个消费者线程，并在它们启动后等待所有线程运行结束。程序的输出会显示当前已经消费的物品数量和缓冲区中的物品列表。

该程序使用了以下技术实现全局共享数据、线程间通信、线程同步：

1. **全局共享数据**：程序中的`items`列表是全局共享数据，用于存储生产者生产的物品和消费者消费的物品。所有线程都可以访问和修改该列表。
2. **线程间通信**：程序中使用了条件变量`not_empty`和`not_full`来进行线程间通信。生产者线程通过`not_empty.notify()`通知消费者线程有新物品可供消费，而消费者线程通过`not_full.notify()`通知生产者线程有空位可供生产。同时，消费者线程在`not_empty.wait()`处等待生产者线程通知有新物品可供消费，而生产者线程在`not_full.wait()`处等待消费者线程通知有空位可供生产。
3. **线程同步**：程序中使用了互斥锁`lock`来保护共享数据`items`的访问和操作。在生产者线程和消费者线程需要访问和修改`items`列表时，先获取互斥锁`lock`，操作完成后再释放锁，以保证共享数据的正确访问和操作。同时，在生产者线程和消费者线程需要等待通知时，通过`not_empty.wait()`和`not_full.wait()`释放锁并等待通知，以避免线程阻塞和浪费CPU资源。