HPC-29-1

January 30, 2024

1 Assignment 3

1. Implement Producer-Consumer problem (PCP). Analyze the significance of semaphore, mutex, bounded buffer, producer thread, consumer thread using the code available on Producer-Consumer Problem in Python - AskPython.
   (a) Write a brief about the problem and solution.
   (b) Code and Output
2. Demonstrate how PCP occurs for a application of your choice.

Ans.

The producer-consumer problem is a classic synchronization problem in OS, especially in concurrent programming and multi-threading. It involves two types of processes:

1. Producers: These processes generate data or items and store them in the shared buffer;
2. Consumers: These processes retrieve and consume items from the buffer.

The challenge is to make sure that the following happen: * Producers do not produce items if the buffer is full * Consumers do not consume items if the buffer is empty.

The main objective is to keep producers and consumers in sync so as to avoid problems like data corruption, race conditions, and deadlocks.

Solution:

Here is a straightforward solution that makes use of bounded buffers and semaphores:

1. Shared Buffer:

- A fixed-size buffer is shared between producers and consumers.

2. Semaphore for Empty Slots (empty):

- Initialized to the size of the buffer.
- Represents the number of empty slots in the buffer.
- Decrements by producers when they add an item.
- Decrements by consumers when they remove an item.

3. Semaphore for Full Slots (full):

- Initialized to 0.
- Represents the number of filled slots in the buffer.
- Incremented by producers when they add an item.
- Decrements by consumers when they remove an item.

4. Mutex Lock:

- Protects access to the shared buffer to race conditions.

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[3]:
```
import threading
import time
```

[4]:
```
# Shared Memory variables
CAPACITY = 10
buffer = [-1 for i in range(CAPACITY)]
in_index = 0
out_index = 0
```

[5]:
```
# Declaring Semaphores
mutex = threading.Semaphore()
empty = threading.Semaphore(CAPACITY)
full = threading.Semaphore(0)
```

[6]:
```
# Producer Thread Class
class Producer(threading.Thread):
  def run(self):

    global CAPACITY, buffer, in_index, out_index
    global mutex, empty, full

    items_produced = 0
    counter = 0

    while items_produced < 20:
      empty.acquire()
      mutex.acquire()

      counter += 1
      buffer[in_index] = counter
      in_index = (in_index + 1)%CAPACITY
      print("Producer produced : ", counter)

      mutex.release()
      full.release()

      time.sleep(0)

      items_produced += 1
```

[7]:
```
# Consumer Thread Class
class Consumer(threading.Thread):
  def run(self):

    global CAPACITY, buffer, in_index, out_index, counter
    global mutex, empty, full

    items_consumed = 0

    while items_consumed < 20:
      full.acquire()
      mutex.acquire()

      item = buffer[out_index]
      out_index = (out_index + 1)%CAPACITY
      print("Consumer consumed item : ", item)

      mutex.release()
      empty.release()

      time.sleep(0.5)

      items_consumed += 1
```

[8]:
```
producer = Producer()
consumer = Consumer()

consumer.start()
producer.start()

producer.join()
consumer.join()
```

```
Producer produced :  1
Consumer consumed item :  1
Producer produced :  2
Producer produced :  3
Producer produced :  4
Producer produced :  5
Producer produced :  6
Producer produced :  7
Producer produced :  8
Producer produced :  9
Producer produced :  10
Producer produced :  11
Consumer consumed item :  2
Producer produced :  12
Consumer consumed item :  3
```

```
Producer produced :  13
Consumer consumed item :  4
Producer produced :  14
Consumer consumed item :  5
Producer produced :  15
Consumer consumed item :  6
Producer produced :  16
Consumer consumed item :  7
Producer produced :  17
Consumer consumed item :  8
Producer produced :  18
Consumer consumed item :  9
Producer produced :  19
Consumer consumed item :  10
Producer produced :  20
Consumer consumed item :  11
Consumer consumed item :  12
Consumer consumed item :  13
Consumer consumed item :  14
Consumer consumed item :  15
Consumer consumed item :  16
Consumer consumed item :  17
Consumer consumed item :  18
Consumer consumed item :  19
Consumer consumed item :  20
```

Implemantation use if else and for loop

[9]:
```
import time
```

[10]:
```
CAPACITY = 10
buffer = [-1 for i in range(CAPACITY)]
in_index = 0
out_index = 0
```

[11]:
```
mutex = threading.Semaphore()
empty = threading.Semaphore(CAPACITY)
full = threading.Semaphore(0)
```

[12]:
```
class Producer(threading.Thread):
    def run(self):
        global CAPACITY, buffer, in_index
        global mutex, empty, full

        for counter in range(1, 21):
            empty.acquire()
            mutex.acquire()
```

```
            buffer[in_index] = counter
            in_index = (in_index + 1) % CAPACITY
            print("Producer produced:", counter)

            mutex.release()
            full.release()

            time.sleep(0)
```

[13]:
```
class Consumer(threading.Thread):
    def run(self):
        global CAPACITY, buffer, out_index
        global mutex, empty, full

        for _ in range(20):
            full.acquire()
            mutex.acquire()

            item = buffer[out_index]
            out_index = (out_index + 1) % CAPACITY
            print("Consumer consumed item:", item)

            mutex.release()
            empty.release()

            time.sleep(0.5)
```

[14]:
```
producer = Producer()
consumer = Consumer()

consumer.start()
producer.start()

producer.join()
consumer.join()
```

```
Producer produced: 1
Consumer consumed item: 1
Producer produced: 2
Producer produced: 3
Producer produced: 4
Producer produced: 5
Producer produced: 6
Producer produced: 7
Producer produced: 8
Producer produced: 9
Producer produced: 10
Producer produced: 11
```

```
Consumer consumed item: 2
Producer produced: 12
Consumer consumed item: 3
Producer produced: 13
Consumer consumed item: 4
Producer produced: 14
Consumer consumed item: 5
Producer produced: 15
Consumer consumed item: 6
Producer produced: 16
Consumer consumed item: 7
Producer produced: 17
Consumer consumed item: 8
Producer produced: 18
Consumer consumed item: 9
Producer produced: 19
Consumer consumed item: 10
Producer produced: 20
Consumer consumed item: 11
Consumer consumed item: 12
Consumer consumed item: 13
Consumer consumed item: 14
Consumer consumed item: 15
Consumer consumed item: 16
Consumer consumed item: 17
Consumer consumed item: 18
Consumer consumed item: 19
Consumer consumed item: 20
```

example useage

[15]:
```
import queue
import random
```

Example

Event Handling in GUI:

- Producers: User input events (clicks, keystrokes).
- Consumers: Event handlers or listeners.
- Buffer: Event queue.

[16]:
```
MAX_QUEUE_SIZE = 5
event_queue = queue.Queue(MAX_QUEUE_SIZE)
mutex = threading.Lock()
empty = threading.Semaphore(MAX_QUEUE_SIZE)
full = threading.Semaphore(0)
```

[17]:
```
class UserClickProducer(threading.Thread):
    def run(self):
        global MAX_QUEUE_SIZE, event_queue
        global mutex, empty, full

        for _ in range(10):
            print("User clicked")

            empty.acquire()
            mutex.acquire()

            event_queue.put("Click")

            mutex.release()
            full.release()

            time.sleep(random.uniform(0.1, 0.5))
```

[18]:
```
class EventHandlerConsumer(threading.Thread):
    def run(self):
        global MAX_QUEUE_SIZE, event_queue
        global mutex, empty, full

        for _ in range(10):
            full.acquire()
            mutex.acquire()

            event = event_queue.get()
            print(f"Handling event: {event}")

            mutex.release()
            empty.release()

            time.sleep(random.uniform(0.1, 0.5))
```

[19]:
```
user_click_producer = UserClickProducer()
event_handler_consumer = EventHandlerConsumer()

user_click_producer.start()
event_handler_consumer.start()

user_click_producer.join()
event_handler_consumer.join()
```

```
User clicked
Handling event: Click
User clicked
Handling event: Click
```

```
User clicked
Handling event: Click
User clicked
Handling event: Click
User clicked
User clicked
User clicked
User clicked
Handling event: Click
Handling event: Click
User clicked
Handling event: Click
User clicked
Handling event: Click
Handling event: Click
Handling event: Click
```