1. Brief description of the task.

A hairdresser has one workstation and a reception area with several chairs. When the barber finishes cutting a client's hair, he lets the client go and then goes to the reception area to see if there are any waiting clients. If there are, he invites one of them in and cuts his hair. If there are no waiting customers, he goes back to his chair and sleeps in it. Each incoming customer looks at what the barber is doing. If the barber is sleeping, the customer wakes him up and sits in his chair. If the barber is working, the customer goes to the reception area. If there is a free chair in the reception room, the client sits down and waits for his turn. If there is no free chair, the client leaves.

The solution should ensure that the barbershop is functioning properly with the barber cutting the hair of anyone who comes in while there are customers and then sleeping until the next customer shows up.

2. Description of the methods that were used to solve the problem.

To solve this problem, concurrent programming methods can be used. Synchronisation primitives such as locks, signal quantities or condition variables are usually used to coordinate the actions of the barber and the customer.

3. Justify the need to use a specific synchronization object to solve your problem.

In this case, a specific synchronisation object, i.e. a semaphore, can be used. Signal quantities are well suited for dealing with limited resources such as the number of available chairs in the reception area. By using a semaphore to represent the number of available chairs, customers can acquire and release chairs as they enter or leave the reception area. This ensures that the number of waiting customers does not exceed the number of available chairs.

4. Schemes of algorithms for solving the problem at hand.

import threading

import time

import random

MAX\_CHAIRS = 5 # 接待区域的椅子数量

barber\_available = threading.Semaphore(0)

customer\_available = threading.Semaphore(0)

chairs\_available = threading.Semaphore(MAX\_CHAIRS)

def cut\_hair(customer):

print(f"Barber is cutting hair of Customer {customer}")

def get\_haircut(barber):

print(f"Customer {barber} is getting a haircut")

def leave(customer):

print(f"Customer {customer} is leaving the barbershop")

def barber():

while True:

customer\_available.acquire() # 如果没有顾客可用，理发师进入睡眠状态

chairs\_available.release() # 增加可用椅子的数量

cut\_hair(threading.current\_thread().name) # 进行理发操作

barber\_available.release() # 表示理发师可用

def customer(customer\_num):

if chairs\_available.acquire(False):

# 如果有可用椅子，顾客占用一把椅子

customer\_available.release() # 表示有顾客可用

barber\_available.acquire() # 等待理发师可用

get\_haircut(customer\_num) # 理发

chairs\_available.release() # 释放椅子

else:

leave(customer\_num) # 没有可用椅子，顾客离开

# 创建并启动理发师线程

barber\_thread = threading.Thread(target=barber)

barber\_thread.start()

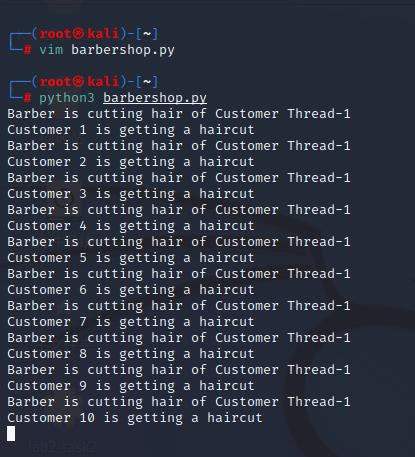
# 创建并启动顾客线程

for i in range(10):

time.sleep(random.randint(1, 3)) # 随机等待一段时间后，顾客到来

customer\_thread = threading.Thread(target=customer, args=(i+1,))

customer\_thread.start()



5. Control questions

1. Explain what is the problem with process/thread synchronization?

The problem with process/thread synchronization arises when multiple processes or threads access shared resources concurrently. It can lead to issues such as race conditions, deadlocks, and data inconsistencies. Synchronization mechanisms are used to coordinate the access and ensure the orderly execution of concurrent processes/threads.

1. What is the "race effect"?

The "race effect" refers to a situation where the outcome of a concurrent execution depends on the relative timing or interleaving of operations. In other words, the result of the execution can be different based on the race condition, where multiple processes/threads compete for shared resources without proper synchronization.

1. What's a clincher?

In the context of a debate or argument, a clincher is a compelling or decisive statement or point that helps to settle the discussion or persuade the other party.

1. What is mutual exclusivity?

Mutual exclusivity refers to the property or condition where only one process or thread can access a shared resource at a given time. It ensures that concurrent access to the resource is serialized, preventing conflicts and maintaining data integrity.

1. What is a critical section in terms of operating system theory?

In operating system theory, a critical section refers to a section of code that accesses shared resources and needs to be executed as an atomic operation. It should not be interrupted or concurrently executed by other processes/threads to avoid race conditions or data inconsistencies. Synchronization mechanisms like locks, semaphores, or monitors are used to implement critical section protection.

1. What is a kernel object? Which windows or Linux operating system kernel objects do you know?

A kernel object is a fundamental data structure managed by the operating system kernel. It represents a system resource or entity that can be shared and accessed by multiple processes or threads. In Windows or Linux operating systems, some kernel objects include files, processes, threads, mutexes, semaphores, events, and pipes.

1. What are some approaches to preventing, the consequences of deadlocks?

Some approaches to preventing or mitigating the consequences of deadlocks include:

Deadlock avoidance: Employ resource allocation strategies that dynamically analyze the resource requests to ensure the system remains in a safe state, avoiding the possibility of deadlocks.

Deadlock detection: Periodically check the system's resource allocation and resource allocation graph to identify the presence of deadlocks. If detected, appropriate actions can be taken, such as terminating processes or performing resource preemption.

Deadlock prevention: Design the system in a way that eliminates one or more of the necessary conditions for deadlock occurrence, such as resource exclusion, hold and wait, no preemption, and circular wait.

Deadlock recovery: When a deadlock is detected, the system can recover by terminating one or more processes involved in the deadlock or by rolling back the actions of processes to a safe state.

1. How can you diagnose the presence of a large process queue or deadlock in a system? What are the differences between these situations?

To diagnose the presence of a large process queue or deadlock in a system, you can monitor system metrics such as:

Process/Thread count: If the number of processes or threads waiting increases significantly, it may indicate a large process queue.

Resource utilization: High resource utilization can be a sign of processes waiting for resources, potentially indicating a deadlock.

Blocked or waiting processes/threads: Monitoring the state of processes or threads can reveal if they are blocked or waiting for resources.

Resource allocation graph: Analyzing the resource allocation graph can help identify circular wait situations that lead to deadlocks.

1. How is the critical section synchronization object different from semaphore synchronization object in windows operating system.

In the Windows operating system, the critical section synchronization object is a lightweight synchronization primitive that allows mutually exclusive access to a shared resource. It is designed for use within a single process and provides better performance compared to other synchronization objects like mutexes or semaphores.

1. What are files projected into memory? What basic operating system mechanisms are implemented through this approach?

Files projected into memory refer to the technique of mapping a file's content directly into a process's virtual memory space. This allows the process to access the file's data as if it were in memory, without the need for explicit read or write operations. Any changes made to the memory-mapped file are automatically reflected in the underlying file, and vice versa.