МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет о лабораторной работе №2.24 по дисциплине основы программной инженерии

Выполнил:
Шальнев Владимир Сергеевич,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:
Доцент кафедры
прикладной математики и
компьютерной безопасности,
Воронкин Р.А.

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2022 г.

Выполнение:

Пример 1

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from threading import Condition, Thread
from queue import Queue
from time import sleep

cv = Condition()
q = Queue()


# Consumer function for order processing
def order_processor(name):
    while True:
        with cv:
            # Wait while queue is empty
            while q.empty():
                cv.wait()
            try:
                # Get data (order) from queue
                order = q.get_nowait()
                print(f"{name}: {order}")
                # If get "stop" message then stop thread
                if order == "stop":
                    break
            except:
                pass
            sleep(0.1)


if __name__ == "__main__":
    # Run order processors
    Thread(target=order_processor, args=("thread 1",)).start()
    Thread(target=order_processor, args=("thread 2",)).start()
    Thread(target=order_processor, args=("thread 3",)).start()
    # Put data into queue
    for i in range(10):
        q.put(f"order {i}")
    # Put stop-commands for consumers
    for _ in range(3):
        q.put("stop")
    # Notify all consumers
    with cv:
        cv.notify_all()
```

Результат работы примера 1

```
thread 1: order 0
thread 1: order 1
thread 1: order 2
thread 1: order 3
thread 1: order 4
thread 1: order 5
thread 1: order 6
thread 1: order 7
thread 1: order 8
thread 1: order 9
thread 1: stop
thread 2: stop
thread 3: stop

Process finished with exit code 0
```

Пример 2

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Thread, BoundedSemaphore
from time import sleep, time


ticket_office = BoundedSemaphore(value=3)


def ticket_buyer(number):
    start_service = time()
    with ticket_office:
        sleep(1)
        print(f"client {number}, service time: {time() - start_service}")


if __name__ == "__main__":
    buyer = [Thread(target=ticket_buyer, args=(i,)) for i in range(5)]
    for b in buyer:
        b.start()
```

Результат работы примера 2

```
client 0, service time: 1.0003414154052734
client 1, service time: 1.0009071826934814
client 2, service time: 1.0006632804870605
client 3, service time: 2.000310182571411
client 4, service time: 2.0010628700256348

Process finished with exit code 0
```

Пример 3

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Thread, Event
from time import sleep, time


event = Event()


def worker(name: str):
    event.wait()
    print(f"Worker: {name}")


if __name__ == "__main__":
    # Clear event
    event.clear()
    # Create and start workers
    workers = [Thread(target=worker, args=(f"wrk {i}",)) for i in range(5)]
    for w in workers:
        w.start()

    print("Main thread")
    event.set()
```

Результат работы примера 3

```
Main thread
Worker: wrk 2
Worker: wrk 0
Worker: wrk 4Worker: wrk 3

Worker: wrk 1

Process finished with exit code 0
```

Пример 4

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Timer


if __name__ == "__main__":
    timer = Timer(interval=3, function=lambda: print("Message from Timer!"))
    timer.start()
```

Результат работы примера 4

```
Message from Timer!

Process finished with exit code 0
```

Пример 5

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Barrier, Thread
from time import sleep


br = Barrier(3)
store = []


def f1(x):
    print("Calc part1")
    store.append(x**2)
    sleep(0.5)
    br.wait()


def f2(x):
    print("Calc part2")
    store.append(x*2)
    sleep(1)
    br.wait()


if __name__ == "__main__":
    Thread(target=f1, args=(3,)).start()
    Thread(target=f2, args=(7,)).start()
    br.wait()
    print("Result: ", sum(store))
```

Результат работы примера 5

```
Calc part1
Calc part2
Result:  23

Process finished with exit code 0
```

Решение первой индивидуальной задачи

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np
from threading import Thread
from time import time
from queue import Queue
```

```
q = Queue()


def determinant(matrix, number):
    if len(matrix) == 1:
        return matrix[0][0]
    else:
        n = len(matrix)
        summa = 0
        for i in range(n):
            minor = []
            for x in range(n):
                temporary = []
                for y in range(n):
                    if x != i and y != 0:
                        temporary.append(matrix[x][y])
                if len(temporary) == n - 1:
                    minor.append(temporary)
            if i % 2 == 0:
                summa += matrix[i][0] * determinant(minor, -1)
            else:
                summa -= matrix[i][0] * determinant(minor, -1)
        if number != -1:
            q.put_nowait((number, summa))
        return float(summa)


def parallel(matrix):
    temp = []
    threads = []
    for i, elem in enumerate(matrix):
        temp.append(matrix[i][0])
    matrices = []
    n = len(matrix)
    for i in range(n):
        minor = []
        for x in range(n):
            temporary = []
            for y in range(n):
                if x != i and y != 0:
                    temporary.append(matrix[x][y])
            if len(temporary) == n - 1:
                minor.append(temporary)
        matrices.append(minor)
    for i, mat in enumerate(matrices):
        th = Thread(
            target=determinant,
            args=(mat, i,),
        )
        th.start()
        threads.append(th)

    for th in threads:
        th.join()

    answer = 0

    while not q.empty():
        i, det = q.get()
        if i % 2 == 0:
            answer += temp[i] * det
        else:
            answer -= temp[i] * det
```

```
    return answer


if __name__ == '__main__':
    data = np.matrix(np.random.randint(-9, 9, (9, 9)))
    data = data.tolist()
    start = time()
    print(parallel(data))
    time_par = time() - start
    print('Вычисление определителя заняло: {:.3f} секунд'
          .format(time_par)
          )
```

Результата работы программы

```
180085226.0
Вычисление определителя заняло: 1.283 секунд

Process finished with exit code 0
```

Решение второй индивидуальной задачи

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Thread, Lock
from math import sqrt
from queue import Queue

q = Queue()
lock = Lock()


def infinite_sum(x):
    lock.acquire()
    eps = 10 ** -7
    a = 1
    summa = 1
    i = 1
    prev = 0
    while abs(summa - prev) > eps:
        prev = summa
        numerator = 1
        denominator = 1
        for j in range(1, i + 1):
            if 2 * j - 3 > 0:
                numerator *= 2 * j - 3
            denominator *= 2 * j
        temp = (numerator / denominator) * x ** i
        if (-1) ** (i + 1) > 0:
            summa += temp
        else:
            summa -= temp
        i += 1
    q.put(summa)
    lock.release()


def checker(calc_result, n_result):
    print(f"Calculated sum: {calc_result}")
```

```
    print(f"Verification sum: {n_result}")


if __name__ == '__main__':
    checksum = sqrt(1 - 0.8)
    thread1 = Thread(target=infinite_sum, args=(-0.8, ))
    thread1.start()
    thread1 = Thread(target=checker, args=(q.get(), checksum))
    thread1.start()
```

Результат работы программы

```
Calculated sum: 0.447213900189959993
Verification sum: 0.44721359549995787

Process finished with exit code 0
```

Ответы на вопросы:

1. Каково назначение и каковы приемы работы с Lock-объектом.

Lock-объект может находится в двух состояниях: захваченное (заблокированное) и не захваченное (не заблокированное, свободное). После создания он находится в свободном состоянии. Для работы с Lock-объектом используются методы acquire() и release(). Если Lock свободен, то вызов метода acquire() переводит его в заблокированное состояние. Повторный вызов acquire() приведет к блокировке инициировавшего это действие потока до тех пор, пока Lock не будет разблокирован каким-то другим потоком с помощью метода release().

2. В чем отличие работы с RLock-объектом от работы с Lock-объектом.

RLock может освободить только тот поток, который его захватил.

3. Как выглядит порядок работы с условными переменными?

- На стороне потребителя: проверить доступен ли ресурс, если нет, то перейти в режим ожидания с помощью метода wait(), и ожидать оповещение от Producer’a о том, что ресурс готов и с ним можно работать. Метод wait() может быть вызван с таймаутом, по истечении которого поток выйдет из состояния блокировки и продолжит работу.
- На стороне производителя: произвести работы по подготовке ресурса, после того, как ресурс готов оповестить об этом ожидающие потоки с помощью методов notify() или notify_all().

4. Какие методы доступны у объектов условных переменных?

acquire(*args) – захват объекта-блокировки.

release() – освобождение объекта-блокировки.

wait(timeout=None) – блокировка выполнения потока до оповещения о снятии блокировки.

wait_for(predicate, timeout=None) – метод позволяет сократить количество кода, которое нужно написать для контроля готовности ресурса и ожидания оповещения.

notify(n=1) – снимает блокировку с остановленного методом wait() потока. Если необходимо разблокировать несколько потоков, то для этого следует передать их количество через аргумент n.

notify_all() – снимает блокировку со всех остановленных методом wait() потоков.

5. Каково назначение и порядок работы с примитивом синхронизации “семафор”?

Суть его идеи заключается в том, при каждом вызове метода acquire() происходит уменьшение счетчика семафора на единицу, а при вызове release() – увеличение. Значение счетчика не может быть меньше нуля, если на момент вызова acquire() его значение равно нулю, то происходит блокировка потока до тех пор, пока не будет вызван release(). Семафоры поддерживают протокол менеджера контекста.

Для работы с семафорами в Python есть класс Semaphore, при создании его объекта можно указать начальное значение счетчика через параметр value.

6. Каково назначение и порядок работы с примитивом синхронизации “событие”?

Объект класса Event управляет внутренним флагом, который сбрасывается с помощью метода clear() и устанавливается методом set(). Потоки, которые используют объект Event для синхронизации блокируются при вызове метода wait(), если флаг сброшен.

is_set() – возвращает True если флаг находится в взведенном состоянии.
set() – переводит флаг в взведенное состояние.

clear() – переводит флаг в сброшенное состояние.

wait(timeout=None) – блокирует вызвавший данный метод поток если флаг соответствующего Event-объекта находится в сброшенном состоянии.

7. Каково назначение и порядок работы с примитивом синхронизации “таймер”?

Timer реализован как поток, является наследником от Thread, поэтому для его запуска необходимо вызвать start(), если необходимо остановить работу таймера, то вызовите cancel().

8. Каково назначение и порядок работы с примитивом синхронизации “барьер”?

Будет дожидаться завершения работы группы потоков.

Barrier(parties, action=None, timeout=None)

parties – количество потоков, которые будут работать в рамках барьера.

action – определяет функцию, которая будет вызвана, когда потоки будут освобождены (достигнут барьера).

timeout – таймаут, который будет использовать как значение по умолчанию для методов wait().

9. Сделайте общий вывод о применении тех или иных примитивов синхронизации в зависимости от решаемой задачи.

Каждый из этих примитивов по-своему хорош в определенной области. К примеру, очень удобно использовать семафор при условии, что несколько

потоков работают с одними и теме же данными. Универсального примитива, подходящего под любою задачу – нет.