

**МИНОБРНАУКИ РОССИИ**

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

|  |
| --- |
| **РТУ МИРЭА** |
|  |
| **Институт кибербезопасности и цифровых технологий (ИКБ)** |
|  |
| КБ-2 «Прикладные информационные технологии» |

**ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ №3**

**В РАМКАХ ДИСЦИПЛИНЫ «ОПЕРАЦИОННЫЕ СИСТЕМЫ»**

Выполнил:

Студент 2-ого курса

Учебной группы БИСО-02-22

Зубарев В.С.

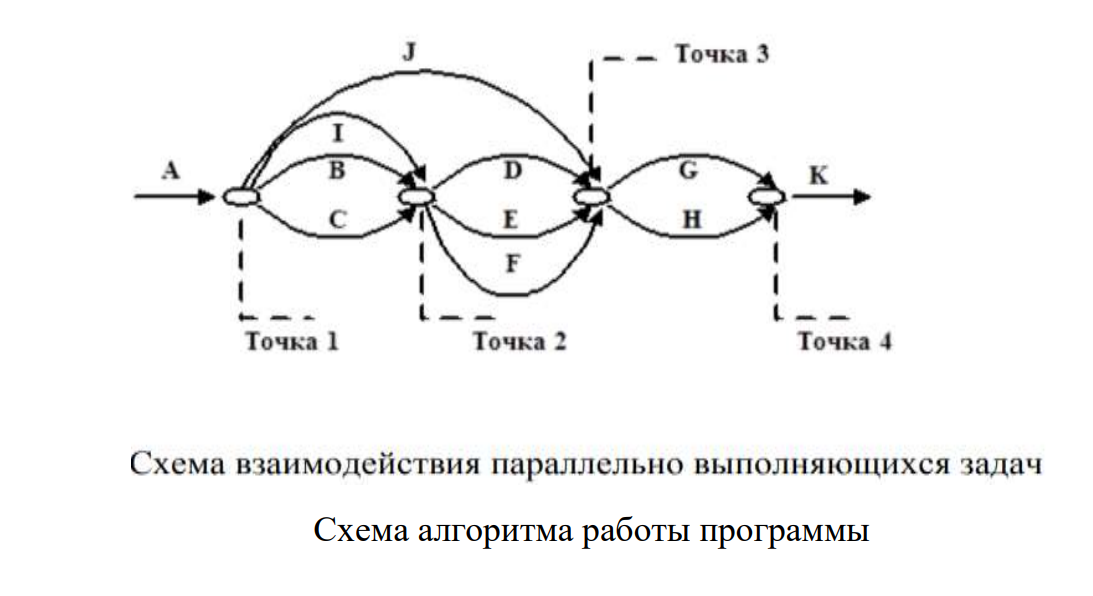
Оглавление

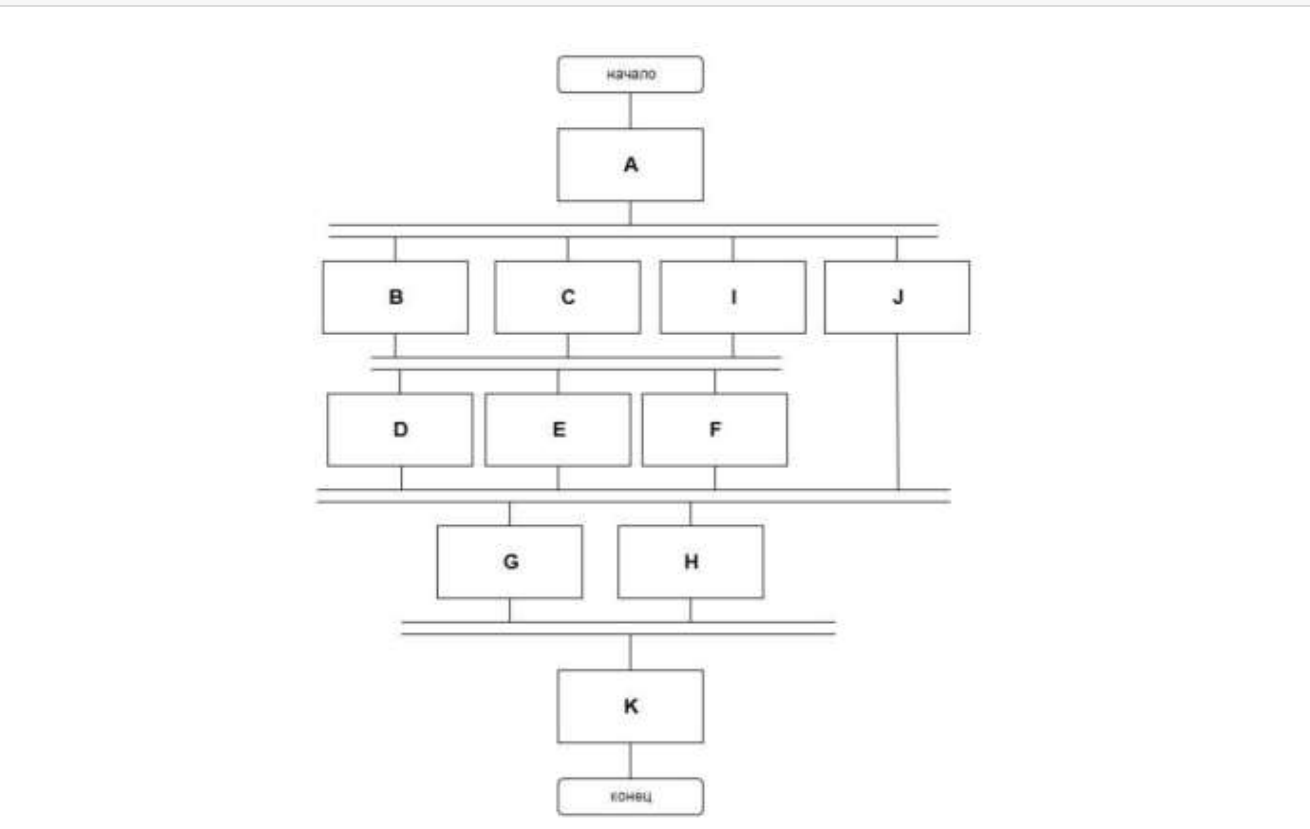
[Задание 3](#_Toc162698930)

[Ответы на вопросы 8](#_Toc162698931)

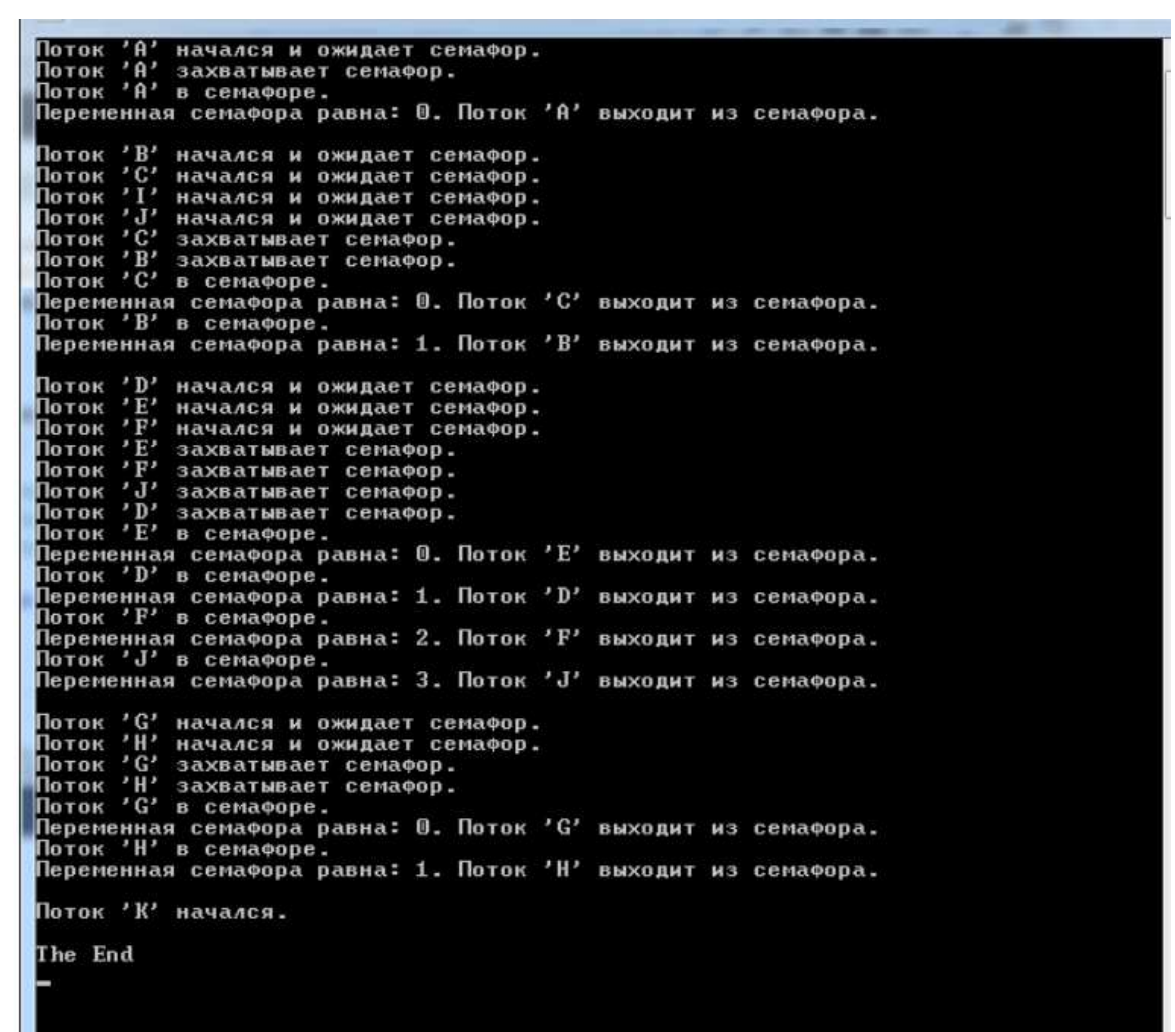
# Задание

Реализовать синхронизацию процессов в трех контрольных точках (смотри рисунок) с помощью объекта синхронизации ядра – «семафор» (по варианту задания). Все процессы в контрольных точках запускаются, процессом, пришедшим в контрольную точку первым. Разработать алгоритм синхронизации взаимодействующих процессов в контрольных точках и приложение на языке высокого уровня. Создать интерфейс программы или консольное приложение, для контроля синхронизации процессов в контрольных точках

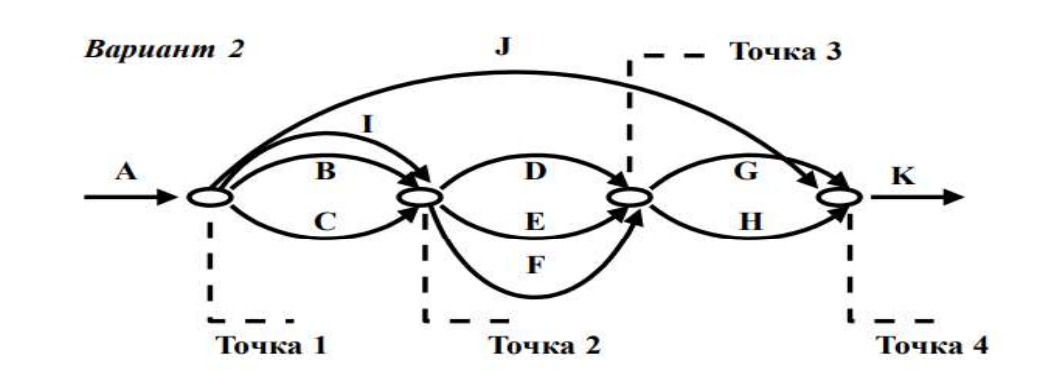


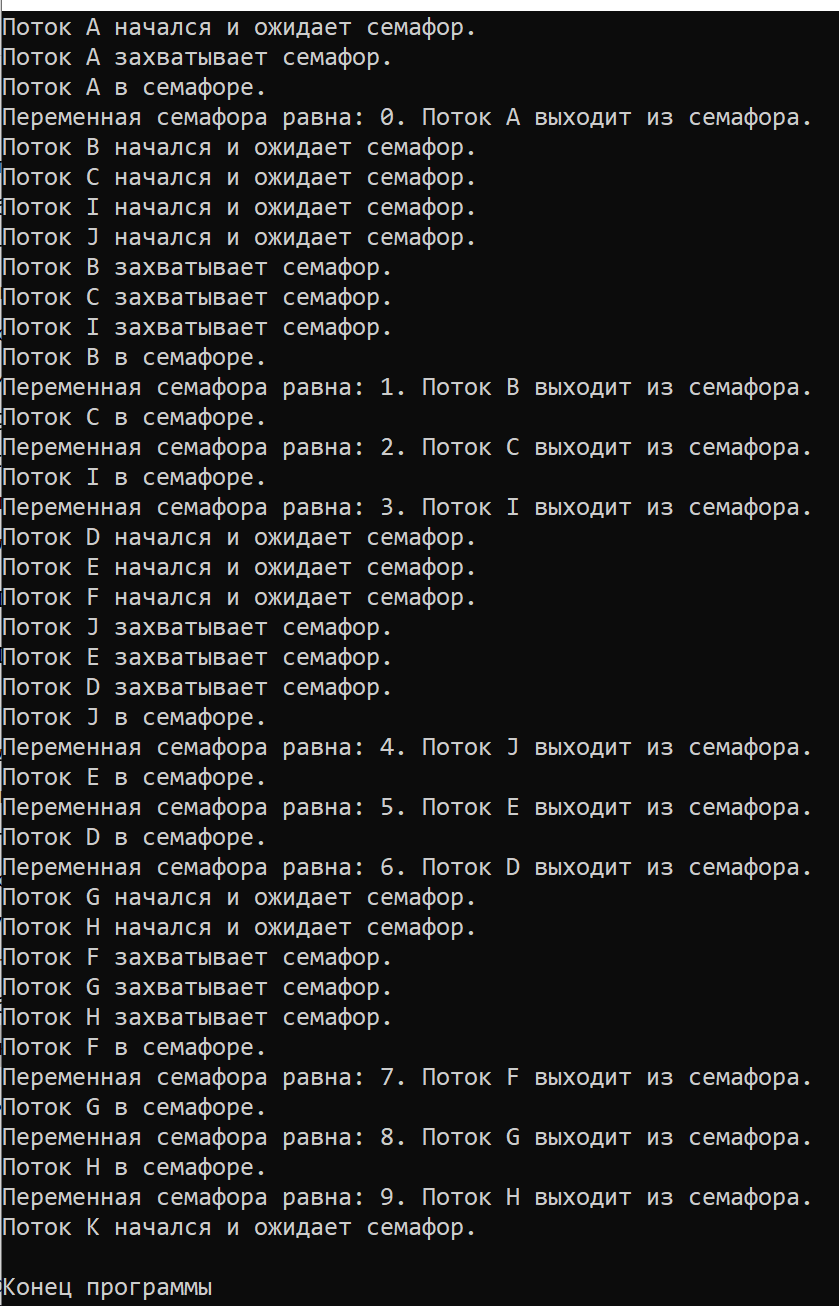


В соответствии с алгоритмом: Процесс А запускает процессы B, C, I, J, процесс, выполняющийся первым из процессов B, C, I, запускает процессы D, E, F, а процесс J конкурирует в запуске процессов G, H с первым, выполнившимся из процессов D, E, F и, в заключении, первый, выполнившийся из процессов G, H, запускает процесс K, который завершает программу. Синхронизация (блокировка) процессов происходит в точках 2, 3, 4. Результат работы консольного приложения представлен на рисунке. Информация о запуске и выполнении потоков в контрольных точках выводится на экран в текстовом режиме.



Вариант 2





Листинг программы

import threading  
import time  
  
  
  
def worker(num):  
 global padding, semaphoreI, terminate\_flag  
 print(f"Поток {num} начался и ожидает семафор.")  
 semaphore1.acquire()  
 if not terminate\_flag:  
 padding += 100  
 print(f"Поток {num} захватывает семафор.")  
 time.sleep(1 + padding / 1000)  
 print(f"Поток {num} в семафоре.")  
 print(f"Переменная семафора равна: {semaphoreI}. Поток {num} выходит из семафора.")  
 semaphoreI += 1  
  
def create\_and\_start\_thread(num):  
 t = threading.Thread(target=worker, args=(num,))  
 threads.append(t)  
 t.start()  
  
threads = []  
semaphore1 = threading.Semaphore(0)  
padding = 0  
semaphoreI = 0  
terminate\_flag = False  
  
  
create\_and\_start\_thread("A")  
time.sleep(0.5)  
semaphore1.release()  
time.sleep(3)  
  
for char in ["B", "C", "I", "J"]:  
 create\_and\_start\_thread(char)  
  
time.sleep(0.5)  
semaphore1.release(3)  
time.sleep(3)  
  
for char in ["D", "E", "F"]:  
 create\_and\_start\_thread(char)  
  
time.sleep(0.5)  
semaphore1.release(3)  
time.sleep(3)  
  
for char in ["G", "H"]:  
 create\_and\_start\_thread(char)  
  
time.sleep(0.5)  
semaphore1.release(3)  
time.sleep(3)  
  
terminate\_flag = True  
k = threading.Thread(target=worker, args=("K",))  
k.start()  
time.sleep(1)  
  
for thread in threads:  
 thread.join()  
  
print("\nКонец программы")

# Ответы на вопросы

1. Независимые и взаимодействующие параллельные процессы:

Независимые:

Не имеют общих ресурсов.

Не влияют друг на друга.

Пример: вычисление π и сортировка массива.

Взаимодействующие:

Используют общие ресурсы.

Могут влиять друг на друга.

Пример: многопользовательская игра, web-сервер.

2. Метод критической секции:

Обеспечивает эксклюзивный доступ к критическому участку кода.

Предотвращает race conditions.

Использует мьютексы или семафоры.

3. Оптимальное число потоков:

Зависит от:

Аппаратных ресурсов (процессоров, ядер).

Программных ресурсов (алгоритмов, ОС).

Задачи.

Слишком много потоков может привести к:

Снижению производительности.

Увеличению контекстных переключений.

4. Программы, процессы и потоки выполнения:

Программа: Набор инструкций для выполнения.

Процесс: Экземпляр программы, выполняющийся в памяти.

Поток: Легковесный процесс, выполняющийся в рамках одного процесса.

5. Жизненный цикл процесса:

Создание: Запуск программы.

Выполнение: Работа программы.

Завершение: Окончание работы программы.

6. Виды межпроцессного взаимодействия:

Сообщения: Передача данных между процессами.

Совместная память: Общий доступ к памяти между процессами.

Синхронизация: Координация действий между процессами.

7. Семафоры:

Счетчики, которые используются для ограничения доступа к ресурсам.

Решают проблему взаимного исключения.

Используются для синхронизации потоков и процессов.

Примеры использования семафоров:

Ограничение доступа к принтеру.

Регулирование доступа к базе данных.

Синхронизация работы нескольких потоков в многопоточном приложении.