МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«Омский государственный технический университет»

Факультет информационных технологий и компьютерных систем

Кафедра «Прикладная математика и фундаментальная информатика»

**Домашнее задание**

|  |  |  |
| --- | --- | --- |
| по дисциплине | Операционные системы |  |

|  |  |
| --- | --- |
| Студента | Зензина Егора Николаевича |
|  | фамилия, имя, отчество полностью |
| Курс | 2 Группа ФИТ-222 |
| Направление | 02.03.02 Фундаментальная информатика |
|  | и информационные технологии |
|  | код, наименование |
| Руководитель | старший преподаватель |
|  | должность, ученая степень, звание |
|  | Карабцов Р.Д. |
|  | фамилия, инициалы |
| Выполнил | 18.05.2024 |
|  | дата, подпись студента |
| баллы |  |
|  | дата, подпись руководителя |

Омск-2024

**Задание**

Разработать многопоточную программу, отображающие на экране взаимодействие трех нитей «читателей» из общей области данных и трех «писателей», записывающих в этот буфер данные. Буфер предназначен для хранения 12 символов. Первая нить-писатель выводит латинскими буквами название города "Novosibirsk", вторая нить-писатель выводит латинскими буквами название города "Semipalatink", а третья — название города "Ekaretinburg". Такой вывод эти три нити осуществляют в два приема, первый из которых записывает половину своего текста без завершающего этот промежуточный текст нуля. Между такими половинами вывода нити производят задержку на случайную величину миллисекунд, но не более 1 сек. После вывода своего текста в буфер каждая нить-писатель переходит в ожидание порядка 2-3 сек до следующей попытки записи в буфер. Нити-читатели - через случайный интервал порядка 300 мсек - читают данные из буфера, если это позволяют средства синхронизации доступа между нитями, и вывод прочитанный текст на экран, каждая в свой столбец. Каждый вывод нити-читателя осуществляется в новую строку своего столбца, поэтому суммарные действия вывода в таких нитях предусмотреть только для 20 - 24 строк. Синхронизацию осуществить с помощью семафоров.

1. Чем абстрактные семафоры отличаются от известных вам программных реализаций.

2. Укажите в каких ситуациях предпочтительней использование мьютексов вместо семафоров и в каких ситуациях целесообразно обратное предпочтение.

**Решение**

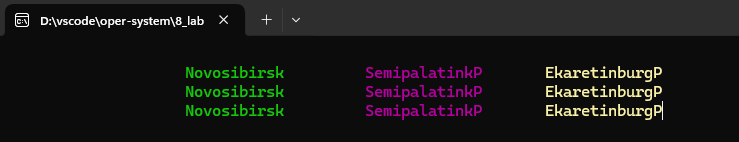
****

Рисунок 1. Вывод программы

1. Абстрактные семафоры отличаются от известных программных реализаций, таких как двоичные и счетные семафоры, тем, что предоставляют более высокий уровень абстракции и гибкости. В отличие от двоичных семафоров, которые имеют только два состояния (заблокирован и разблокирован), и счетных семафоров, которые управляют счетчиком ресурсов, абстрактные семафоры могут быть настроены для различных режимов работы и обеспечивают более сложные условия синхронизации.

2.Выбор между мьютексами и семафорами:

Мьютексы предпочтительнее использовать в случаях, когда необходимо защитить доступ к общему ресурсу или критической секции кода, разрешая доступ только одному потоку. Мьютексы обеспечивают, что в любой момент времени только один поток может захватить мьютекс, предотвращая конфликты доступа к ресурсам.

**Листинг программы**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <windows.h>

#include <process.h>

#include <conio.h>

HANDLE consoleout;

HANDLE Semaphore, Semaphore2;

CRITICAL\_SECTION CriticalSection;

int Delay1, Delay2;

char Buffer[12];

char names[3][14] = {

    {"Novosibirsk"},

    {"Semipalatink"},

    {"Ekaretinburg"}};

int n = 0;

void SetWindow(int Width, int Height)

{

  COORD coord;

  coord.X = Width;

  coord.Y = Height;

  SMALL\_RECT Rect;

  Rect.Top = 0;

  Rect.Left = 0;

  Rect.Bottom = Height - 1;

  Rect.Right = Width - 1;

  SetConsoleScreenBufferSize(consoleout, coord);

  SetConsoleWindowInfo(consoleout, TRUE, &Rect);

}

DWORD WINAPI Write(void \*arg)

{

  int r = (int)arg;

  r--;

  while (1)

  {

    WaitForSingleObject(Semaphore, INFINITE);

    for (n = 0; n < 6; n++)

    {

      Buffer[n] = names[r][n];

    }

    Sleep(100 \* (int)Delay1

    for (n = 6; n < 12; n++)

    {

      Buffer[n] = names[r][n];

    }

    ReleaseSemaphore(Semaphore2, 1, NULL;

    Sleep(2000 + rand() % 1001);

  }

}

DWORD WINAPI Read(void \*arg)

{

  COORD pos;

  char buff[12];

  int lenBuff;

  for (int i = 0; i < 40; i++)

  {

    WaitForSingleObject(Semaphore2, INFINITE);

    pos.X = 20 \* (int)arg;

    pos.Y = i + 1;

    EnterCriticalSection(&CriticalSection);

    SetConsoleCursorPosition(consoleout, pos);

    switch((int)arg){

        case 1: SetConsoleTextAttribute(consoleout, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

        break;

        case 2: SetConsoleTextAttribute(consoleout, FOREGROUND\_RED   | FOREGROUND\_BLUE  | FOREGROUND\_INTENSITY);

        break;

        case 3: SetConsoleTextAttribute(consoleout, FOREGROUND\_RED   | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

        break;

    }

    lenBuff = sprintf(buff, "%s", Buffer);

    lenBuff = ((int)arg > 1) ? lenBuff - 1 : lenBuff;

    WriteFile(consoleout, buff, lenBuff, NULL, NULL);

    LeaveCriticalSection(&CriticalSection);

    ReleaseSemaphore(Semaphore, 1, NULL);

    Sleep(100 \* (int)Delay2);

  }

  return 0;

}

void main(int args, char \*argv[])

{

  system("cls");

  consoleout = GetStdHandle(STD\_ERROR\_HANDLE);

  SetWindow(120, 30);

  HANDLE writeThread[3];

  HANDLE readThread[3];

  Delay1 = 20;

  Delay2 = 3;

  InitializeCriticalSection(&CriticalSection;

  if (args >= 3)

  {

    sscanf(argv[1], "%d", &Delay1);

    sscanf(argv[2], "%d", &Delay2);

  }

  Semaphore = CreateSemaphore(NULL, 1, 1, NULL);

  Semaphore2 = CreateSemaphore(NULL, 0, 1, NULL);

  for (int i = 0; i < 3; i++)

  {

    writeThread[i] = CreateThread(NULL, 4096, Write, (void \*)(i + 1), 0, NULL);

    readThread[i] = CreateThread(NULL, 4096, Read, (void \*)(i + 1), 0, NULL);

  }

  getchar();

  DeleteCriticalSection(&CriticalSection);

  for (int i = 0; i < 3; i++)

  {

    CloseHandle(writeThread[i]);

    CloseHandle(readThread[i]);

  }

}