|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Отчет**

***к лабораторной работе №6 по курсу «Операционные системы»***

***по теме «Реализация монитора Хоара «Читатели-писатели» под ОС Windows»***

|  |  |
| --- | --- |
| **Студент: Прохорова Л. А.**  **Группа: ИУ7-53Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель: Рязанова Н. Ю.** |  |

Москва.

2020 г.

В лабораторной работе необходимо разработать многопоточное приложение, используя API ОС Windows такие как, потоки, события (event) и мьютексы (mutex). Потоки разделяют единственную глобальную переменную. Приложение реализует монитор Хоара «Читатели-писатели».

Код программы:

#include <stdio.h>

#include <stdbool.h>

#include <windows.h>

#define READERS\_CNT 5

#define WRITERS\_CNT 3

const DWORD sleep\_time = 350;

const int iterations = 8;

volatile LONG active\_readers = 0;

bool writer\_is\_active = **false**;

volatile LONG writers\_in\_wait = 0;

volatile LONG readers\_in\_wait = 0;

volatile int current\_num = 0;

HANDLE mutex;

HANDLE can\_read;

HANDLE can\_write;

HANDLE writers[WRITERS\_CNT];

HANDLE readers[READERS\_CNT];

void start\_write()

{

InterlockedIncrement(&writers\_in\_wait);

if ( WaitForSingleObject(can\_read, 0) == WAIT\_OBJECT\_0|| writer\_is\_active)

{

WaitForSingleObject(can\_write, INFINITE);

}

InterlockedDecrement(&writers\_in\_wait);

writer\_is\_active = **true**;

ResetEvent(can\_write);

}

void stop\_write()

{

writer\_is\_active = **false**;

if (!writers\_in\_wait)

{

SetEvent(can\_read);

}

else

{

SetEvent(can\_write);

}

}

void start\_read()

{

InterlockedIncrement(&readers\_in\_wait);

if (WaitForSingleObject(can\_write, 0) == WAIT\_OBJECT\_0 || writer\_is\_active)

{

WaitForSingleObject(can\_read, INFINITE);

}

WaitForSingleObject(mutex, INFINITE);

InterlockedDecrement(&readers\_in\_wait);

InterlockedIncrement(&active\_readers);

SetEvent(can\_read);

ReleaseMutex(mutex);

}

void stop\_read()

{

InterlockedDecrement(&active\_readers);

if (readers\_in\_wait == 0)

{

SetEvent(can\_write);

}

}

DWORD WINAPI writer(LPVOID lpParams)

{

for (int current\_it = 0; current\_it < iterations; current\_it++)

{

start\_write();

current\_num += 1;

printf("Writer #%ld write: %d**\n**", (int) lpParams, current\_num);

stop\_write();

Sleep(sleep\_time);

}

return EXIT\_SUCCESS;

}

DWORD WINAPI reader(LPVOID lpParams)

{

while (current\_num < iterations \* WRITERS\_CNT)

{

start\_read();

printf("Reader #%ld read: %d**\n**", (int) lpParams, current\_num);

stop\_read();

Sleep(sleep\_time);

}

return EXIT\_SUCCESS;

}

int init\_handles()

{

if ((mutex = CreateMutex(NULL, FALSE, NULL)) == NULL)

{

perror("Error. Can not create mutex!");

return EXIT\_FAILURE;

}

if ((can\_read = CreateEvent(NULL, FALSE, TRUE, NULL)) == NULL)

{

perror("Error. Can not create event can\_read!");

return EXIT\_FAILURE;

}

if ((can\_write = CreateEvent(NULL, TRUE, TRUE, NULL)) == NULL)

{

perror("Error. Can not create event can\_write!");

return EXIT\_FAILURE;

}

return EXIT\_SUCCESS;

}

int create\_threads(HANDLE \*threads, int threads\_count, DWORD (\*on\_thread)(LPVOID))

{

for (int cur\_thread\_num = 0; cur\_thread\_num < threads\_count; cur\_thread\_num++)

{

if ((threads[cur\_thread\_num] = CreateThread(NULL, 0, on\_thread, (LPVOID) cur\_thread\_num, 0, NULL)) == NULL)

{

perror("Error. Can not create event create thread!");

return EXIT\_FAILURE;

}

}

return EXIT\_SUCCESS;

}

int main()

{

setbuf(stdout, NULL);

int rc = EXIT\_SUCCESS;

if ((rc = init\_handles()) != EXIT\_SUCCESS || (rc = create\_threads(writers, WRITERS\_CNT, writer)) != EXIT\_SUCCESS

|| (rc = create\_threads(readers, READERS\_CNT, reader)) != EXIT\_SUCCESS){ return rc;}

WaitForMultipleObjects(WRITERS\_CNT, writers, TRUE, INFINITE);

WaitForMultipleObjects(READERS\_CNT, readers, TRUE, INFINITE);

CloseHandle(mutex);

CloseHandle(can\_read);

CloseHandle(can\_write);

return rc;

}

Демонстрация работы программы

