|  |  |
| --- | --- |
| **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» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 6 |

**Название:** Реализация монитора Хоара «Читатели-писатели» под ОС Windows

**Дисциплина:** Операционные системы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-52Б |  |  | Е.В. Брянская |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Н.Ю. Рязанова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

**Задание 1.**

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».

#define K\_TIME 5

#define NUM\_WRITERS 3

#define NUM\_READERS 6

HANDLE mtx;

HANDLE can\_write;

HANDLE can\_read;

HANDLE writers[NUM\_WRITERS];

HANDLE readers[NUM\_READERS];

BOOL active\_writer = FALSE;

volatile long active\_readers = 0;

volatile long waiting\_readers = 0, waiting\_writers = 0;

int shr\_var = -1;

void **start\_write**()

{

InterlockedIncrement(&waiting\_writers);

if (active\_readers > 0 || active\_writer)

if (WaitForSingleObject(can\_write, INFINITE) == WAIT\_FAILED)

{

perror("WaitForSingleObject error\n");

exit(4);

}

InterlockedDecrement(&waiting\_writers);

active\_writer = TRUE;

}

void **stop\_write**()

{

active\_writer = FALSE;

if (waiting\_readers > 0)

SetEvent(can\_read);

else if (waiting\_writers)

SetEvent(can\_write);

}

void **start\_read**()

{

InterlockedIncrement(&waiting\_readers);

if (active\_writer || waiting\_writers > 0)

{

if (WaitForSingleObject(can\_read, INFINITE) == WAIT\_FAILED)

{

perror("WaitForSingleObject error\n");

exit(4);

}

}

if (WaitForSingleObject(mtx, INFINITE) == WAIT\_FAILED)

{

perror("WaitForSingleObject error\n");

exit(4);

}

InterlockedDecrement(&waiting\_readers);

InterlockedIncrement(&active\_readers);

if (!ReleaseMutex(mtx))

{

perror("ReleaseMutex error\n");

exit(5);

}

if (waiting\_readers)

SetEvent(can\_read);

}

void **stop\_read**()

{

InterlockedDecrement(&active\_readers);

if (active\_readers == 0 && waiting\_writers)

SetEvent(can\_write);

}

DWORD WINAPI **action\_writer**()

{

long cur\_id = GetCurrentThreadId();

srand(time(NULL) + cur\_id);

sleep(rand() % K\_TIME);

while (1)

{

start\_write();

shr\_var++;

printf(">>> WRITER %ld: \twrote %d\n", cur\_id, shr\_var);

stop\_write();

sleep(rand() % K\_TIME + 1);

}

return 0;

}

DWORD WINAPI **action\_reader**()

{

long cur\_id = GetCurrentThreadId();

srand(time(NULL) + cur\_id);

sleep(rand() % K\_TIME);

while (1)

{

start\_read();

printf(">>> READER %ld: \tread %d\n", cur\_id, shr\_var);

stop\_read();

sleep(rand() % K\_TIME + 1);

}

return 0;

}

void **create\_mutex**()

{

mtx = CreateMutex(NULL, FALSE, NULL);

if (mtx == NULL)

{

perror("CreateMutex error\n");

exit(1);

}

}

void **create\_events**()

{

can\_write = CreateEvent(NULL, FALSE, FALSE, NULL);

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

if (can\_write == NULL || can\_read == NULL)

{

perror("CreateEvent error\n");

exit(2);

}

}

void **create\_threads**()

{

for (int i = 0; i < NUM\_WRITERS; i++)

{

writers[i] = CreateThread(NULL, 0, action\_writer, NULL, 0, NULL);

if (writers[i] == NULL)

{

perror("CreateThread error\n");

exit(3);

}

}

rand();

for (int i = 0; i < NUM\_READERS; i++)

{

readers[i] = CreateThread(NULL, 0, action\_reader, NULL, 0, NULL);

if (readers[i] == NULL)

{

perror("CreateThread error\n");

exit(3);

}

}

}

int **main**()

{

create\_mutex();

create\_events();

create\_threads();

if (WaitForMultipleObjects(NUM\_WRITERS, writers, TRUE, INFINITE) == WAIT\_FAILED ||

WaitForMultipleObjects(NUM\_READERS, readers, TRUE, INFINITE) == WAIT\_FAILED)

{

perror("WaitForMultipleObjects error\n");

exit(5);

}

CloseHandle(mtx);

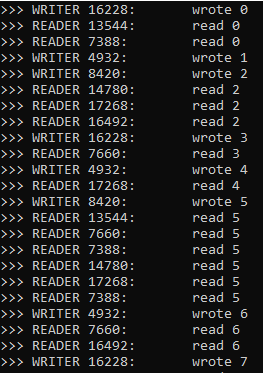
CloseHandle(can\_read);

CloseHandle(can\_write);

return 0;

}

Результат выполнения программы:

****