|  |  |
| --- | --- |
| 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»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 6**

|  |  |
| --- | --- |
| **Тема**  Реализация монитора Хоара «Читатели-писатели»  **Студент** Жигалкин Д.Р  **Группа** ИУ7-55Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_**  **Преподаватель** Рязанова Н.Ю |  |

Москва.

2020 г.

**Задание на лабораторную работу:**

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».

Листинг 1.1 — Реализация монитора Хоара «Читатели-писатели»

1. #include <stdio.h>
2. #include <windows.h>
3. #define \_CRT\_SECURE\_NO\_WARNINGS
4. #define COUNT\_READERS 3
5. #define COUNT\_WRITERS 2
6. struct handle\_information
7. {
8. LONG active\_readers = 0;
9. LONG waiting\_readers = 0;
10. LONG waiting\_writers = 0;
11. bool active\_writer = false;
12. HANDLE mutex;
13. HANDLE can\_read;
14. HANDLE can\_write;
15. int \*buffer;
16. };
17. void start\_write(handle\_information\* my\_handle)
18. {
19. InterlockedIncrement(&my\_handle->waiting\_writers);
20. // обеспечение монопольного доступа писателя
21. if (my\_handle->active\_writer || my\_handle->active\_readers > 0)
22. {
23. WaitForSingleObject(my\_handle->can\_write, INFINITE);
24. }
25. WaitForSingleObject(my\_handle->mutex, INFINITE);
26. InterlockedDecrement(&my\_handle->waiting\_writers);
27. my\_handle->active\_writer = true;
28. // сброс "в ручную"
29. ResetEvent(my\_handle->can\_write);
30. ReleaseMutex(my\_handle->mutex);
31. }
32. void start\_read(handle\_information\* my\_handle)
33. {
34. InterlockedIncrement(&my\_handle->waiting\_readers);
35. if (my\_handle->active\_writer || my\_handle->waiting\_writers > 0)
36. {
37. WaitForSingleObject(my\_handle->can\_read, INFINITE);
38. }
39. InterlockedDecrement(&my\_handle->waiting\_readers);
40. InterlockedIncrement(&my\_handle->active\_readers);
41. // чтобы следующий читатель в очереди читателей смог начать чтение
42. SetEvent(my\_handle->can\_read);
43. }
44. void stop\_write(handle\_information\* my\_handle)
45. {
46. my\_handle->active\_writer = false;
47. if (WaitForSingleObject(my\_handle->can\_read, 0) == WAIT\_OBJECT\_0)
48. {
49. SetEvent(my\_handle->can\_read);
50. }
51. else
52. {
53. SetEvent(my\_handle->can\_write);
54. }
55. }
56. void stop\_read(handle\_information\* my\_handle)
57. {
58. // уменьшение количества активных писаталей
59. InterlockedDecrement(&my\_handle->active\_readers);
60. if (my\_handle->active\_readers == 0)
61. {
62. // активизация писателя из очереди писателей
63. SetEvent(my\_handle->can\_write);
64. }
65. }
66. DWORD reader(handle\_information\* my\_handle)
67. {
68. while (true)
69. {
70. start\_read(my\_handle);
71. printf("Reader #%ld read <- %d\n", GetCurrentThreadId(), \*my\_handle->buffer);
72. stop\_read(my\_handle);
73. Sleep(1000 \* (rand() % 3));
74. }
75. return 0;
76. }
77. DWORD writer(handle\_information\* my\_handle)
78. {
79. while (true)
80. {
81. start\_write(my\_handle);
82. printf("Writer #%ld write -> %ld\n", GetCurrentThreadId(), ++(\*my\_handle->buffer));
83. stop\_write(my\_handle);
84. Sleep(900 \* (rand() % 3));
85. }
86. return 0;
87. }
88. handle\_information\* Initialization(int \*buffer)
89. {
90. // создает новый своободный мьютекс
91. HANDLE mutex = CreateMutex(NULL, FALSE, NULL);
92. if (mutex == NULL)
93. {
94. perror("Can't create mutex");
95. exit(1);
96. }
97. // создание Event, который переключается "в ручную", объект в сигнальном состоянии
98. HANDLE can\_write = CreateEvent(NULL, TRUE, TRUE, NULL);
99. if (can\_write == NULL)
100. {
101. perror("Can't create event can write");
102. exit(1);
103. }
104. // создание Event, который переключается автоматически, объект в сигнальном состоянии
105. HANDLE can\_read = CreateEvent(NULL, FALSE, TRUE, NULL);
106. if (can\_read == NULL)
107. {
108. perror("Can't create event can read");
109. exit(1);
110. }
111. handle\_information\* my\_handle = new handle\_information();
112. my\_handle->mutex = mutex;
113. my\_handle->can\_read = can\_read;
114. my\_handle->can\_write = can\_write;
115. my\_handle->buffer = buffer;
116. return my\_handle;
117. }
118. int main()
119. {
120. HANDLE writers[COUNT\_WRITERS];
121. HANDLE readers[COUNT\_READERS];
122. int buffer = 0;
123. handle\_information\* my\_handle = Initialization(&buffer);
124. for (int i = 0; i < COUNT\_WRITERS; i++)
125. {
126. writers[i] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)&writer, my\_handle, 0, NULL);
127. if (writers[i] == NULL)
128. {
129. perror("Can't create writer");
130. return 1;
131. }
132. }
133. for (int i = 0; i < COUNT\_READERS; i++)
134. {
135. readers[i] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)&reader, my\_handle, 0, NULL);
136. if (readers[i] == NULL)
137. {
138. perror("Can't create reader");
139. return 1;
140. }
141. }
142. // ожидание освобождения
143. WaitForMultipleObjects(COUNT\_WRITERS, writers, TRUE, INFINITE);
144. WaitForMultipleObjects(COUNT\_READERS, readers, TRUE, INFINITE);
145. // освобождение ресурсов
146. CloseHandle(my\_handle->mutex);
147. CloseHandle(my\_handle->can\_read);
148. CloseHandle(my\_handle->can\_write);
149. return 0;
150. }

**Пример работы реализации:**

