|  |  |
| --- | --- |
| 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**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 06 |

**Название:**

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

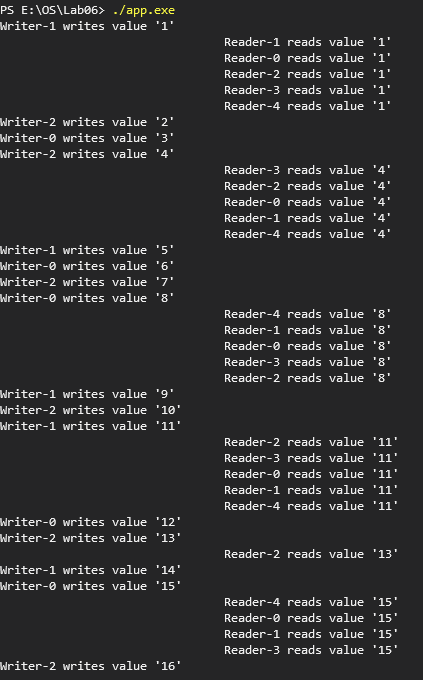
**Дисциплина:  *Операционные системы***

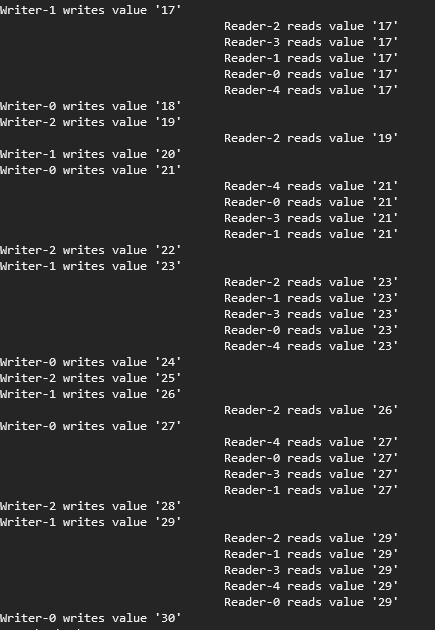
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ***ИУ7И-52Б*** |  |  | Чыонг Н. В. У. |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | **Рязанова Н. Ю.** |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

*Москва, 2021*

* **Задание:**

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».





1. #include <stdio.h>
2. #include <stdbool.h>
3. #include <windows.h>
5. #define WRITERS\_COUNT 3
6. #define READERS\_COUNT 5
8. #define ITERATIONS\_NUMBER 10
10. #define PAUSE 200 /\* ms\*/
12. **HANDLE** mutex;
13. **HANDLE** can\_read;
14. **HANDLE** can\_write;
16. **HANDLE** writers[WRITERS\_COUNT];
17. **HANDLE** readers[READERS\_COUNT];
19. **volatile** **LONG** active\_readers\_count = 0;
20. **bool** active\_writer = **false**;
22. **volatile** **LONG** waiting\_writers\_count = 0;
23. **volatile** **LONG** waiting\_readers\_count = 0;
25. **int** value = 0;
27. **void** start\_read(**void**)
28. {
29. WaitForSingleObject(mutex, INFINITE);
31. InterlockedIncrement(&waiting\_readers\_count);
33. **if** (active\_writer || WaitForSingleObject(can\_write, 0) == WAIT\_OBJECT\_0)
34. {
35. WaitForSingleObject(can\_read, INFINITE);
36. }
38. InterlockedDecrement(&waiting\_readers\_count);
39. InterlockedIncrement(&active\_readers\_count);
40. SetEvent(can\_read);
41. ReleaseMutex(mutex);
42. }
44. **void** stop\_read(**void**)
45. {
46. InterlockedDecrement(&active\_readers\_count);
47. **if** (active\_readers\_count == 0)
48. {
49. SetEvent(can\_write);
50. }
51. }
53. **void** start\_write(**void**)
54. {
55. InterlockedIncrement(&waiting\_writers\_count);
56. **if** (active\_writer || active\_readers\_count > 0)
57. {
58. WaitForSingleObject(can\_write, INFINITE);
59. }
61. InterlockedDecrement(&waiting\_writers\_count);
62. active\_writer = **true**;
63. ResetEvent(can\_write);
64. }
66. **void** stop\_write(**void**)
67. {
68. active\_writer = **false**;
70. **if** (waiting\_readers\_count > 0)
71. {
72. SetEvent(can\_read);
73. }
74. **else**
75. {
76. SetEvent(can\_write);
77. }
78. }
80. **DWORD** WINAPI writer(**LPVOID** lpParams)
81. {
82. **for** (**int** i = 0; i < ITERATIONS\_NUMBER; ++i)
83. {
84. start\_write();
86. value++;
87. printf("Writer-%d writes value '%d'\n", (**int**)lpParams, value);
89. stop\_write();
90. Sleep(PAUSE);
91. }
93. **return** EXIT\_SUCCESS;
94. }
96. **DWORD** WINAPI reader(**LPVOID** lpParams)
97. {
98. **while** (value < WRITERS\_COUNT \* ITERATIONS\_NUMBER)
99. {
100. start\_read();
102. printf("\t\t\t\tReader-%d reads value '%d'\n", (**int**)lpParams, value);
104. stop\_read();
105. Sleep(PAUSE);
106. }
108. **return** EXIT\_SUCCESS;
109. }
111. **int** createHandles(**void**)
112. {
113. **if** ((mutex = CreateMutex(NULL, FALSE, NULL)) == NULL)
114. {
115. perror("CreateMutex");
116. **return** EXIT\_FAILURE;
117. }
119. **if** ((can\_read = CreateEvent(NULL, FALSE, TRUE, NULL)) == NULL)
120. {
121. perror("CreateEvent");
122. **return** EXIT\_FAILURE;
123. }
125. **if** ((can\_write = CreateEvent(NULL, TRUE, TRUE, NULL)) == NULL)
126. {
127. perror("CreateEvent");
128. **return** EXIT\_FAILURE;
129. }
131. **return** EXIT\_SUCCESS;
132. }
134. **int** createThreads(**HANDLE**\* threads, **int** threads\_count, **DWORD**(\*fn\_on\_thread)(**LPVOID**))
135. {
136. **for** (**int** i = 0; i < threads\_count; ++i)
137. {
138. **if** ((threads[i] = CreateThread(NULL, 0, fn\_on\_thread, (**LPVOID**)i, 0, NULL)) == NULL)
139. {
140. perror("CreateThread");
141. **return** EXIT\_FAILURE;
142. }
143. }
145. **return** EXIT\_SUCCESS;
146. }
148. **void** closeHandleThreads(**HANDLE**\* threads, **int** threads\_count)
149. {
150. **for** (**int** i = 0; i < threads\_count; i++)
151. {
152. CloseHandle(threads[i]);
153. }
154. }
156. **int** main(**void**)
157. {
158. setbuf(stdout, NULL);
160. **int** rc = EXIT\_SUCCESS;
162. **if** ((rc = createHandles()) != EXIT\_SUCCESS || (rc = createThreads(writers, WRITERS\_COUNT, writer)) != EXIT\_SUCCESS || (rc = createThreads(readers, READERS\_COUNT, reader)) != EXIT\_SUCCESS)
163. {
164. **return** rc;
165. }
167. WaitForMultipleObjects(WRITERS\_COUNT, writers, TRUE, INFINITE);
168. WaitForMultipleObjects(READERS\_COUNT, readers, TRUE, INFINITE);
170. closeHandleThreads(writers, WRITERS\_COUNT);
171. closeHandleThreads(readers, READERS\_COUNT);
172. CloseHandle(mutex);
173. CloseHandle(can\_read);
174. CloseHandle(can\_write);
176. **return** rc;
177. }