Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет Компьютерных Технологий и Управления

Кафедра Информатики и прикладной математики

**Системное программное обеспечение**

Лабораторная работа №1

Выполнил:

Хлопков Д. С.

Группа: Р3217

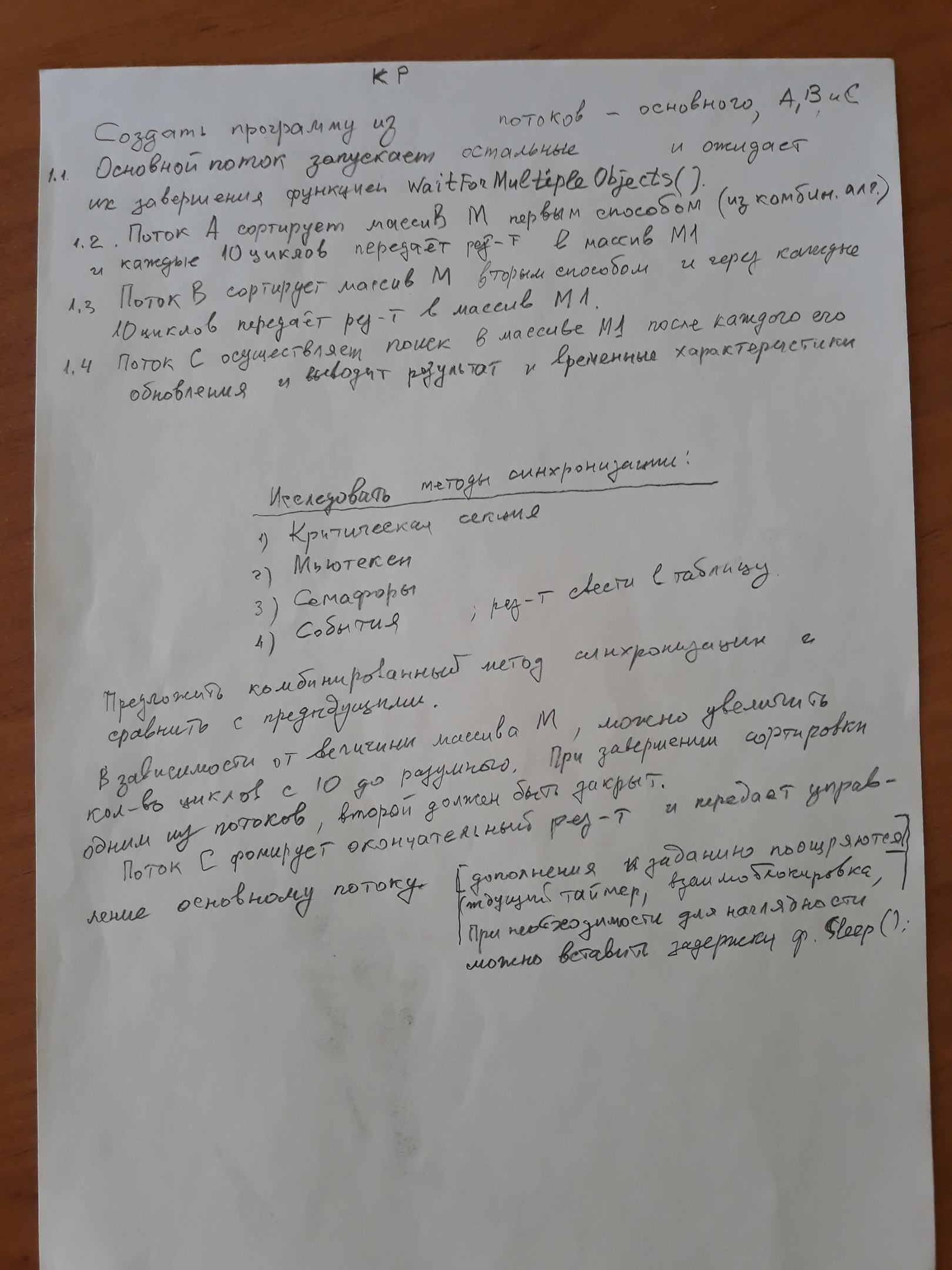
Преподаватель:

Зыков Анатолий Геннадьевич

Санкт-Петербург

2018 г.

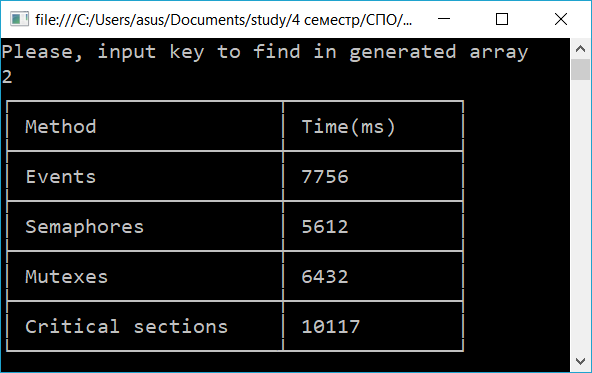
**Задание:**

****

**Листинг программы:**

1. **class** SemaphoreController<T> : IThreadController<T> **where** T : System.IComparable<T>
2. {
3. Semaphore waitChangeHandler = new Semaphore(1,1);
4. **volatile** **bool** firstSorting = **true**, secondSorting = **true**;
5. **private** Sorter<T> sorter;
6. **private** Finder<T> finder;
7. **volatile** T[] m1;
8. AutoResetEvent findEvent = new AutoResetEvent(**false**);
9. **public** SemaphoreController()
10. {
11. sorter = new Sorter<T>(**this**);
12. finder = new Finder<T>();
13. }
15. **public** **void** RunFind(**object** obj)
16. {
17. T key = (T)obj;
18. Thread.Sleep(2);
19. **while** (firstSorting || secondSorting)
20. {
21. findEvent.WaitOne();
22. waitChangeHandler.WaitOne();
23. **try**
24. {
25. T foundKey = finder.Find(key, **this**.m1);
26. }
27. **catch** (KeyNotFoundException e)
28. {
29. }
30. waitChangeHandler.Release();
31. }
32. }
33. **public** **void** Change(T[] arr)
34. {
36. waitChangeHandler.WaitOne();
37. **if** (m1 == **null**)
38. m1 = new T[arr.Length];
39. Array.Copy(arr, m1, arr.Length);
40. waitChangeHandler.Release();
41. findEvent.**Set**();
42. }
44. **public** **void** RunSortMerge(**object** obj)
45. {
46. T[] arr = (T[])obj;
47. sorter.SortMerge(arr);
48. firstSorting = **false**;
49. }
51. **public** **void** RunSortShell(**object** obj)
52. {
53. T[] arr = (T[])obj;
54. sorter.SortShell(arr);
55. secondSorting = **false**;
56. }
57. }
58. **class** MutexController<T> : IThreadController<T> **where** T : System.IComparable<T>
59. {
60. Mutex waitChangeHandler = new Mutex();
61. **volatile** **bool** firstSorting = **true**, secondSorting = **true**;
62. **private** Sorter<T> sorter;
63. **private** Finder<T> finder;
64. **volatile** T[] m1;
65. AutoResetEvent findEvent = new AutoResetEvent(**false**);
66. **public** MutexController()
67. {
68. sorter = new Sorter<T>(**this**);
69. finder = new Finder<T>();
70. }
72. **public** **void** RunFind(**object** obj)
73. {
74. T key = (T)obj;
75. **while** (firstSorting || secondSorting)
76. {
77. findEvent.WaitOne();
78. waitChangeHandler.WaitOne();
79. **try**
80. {
81. T foundKey = finder.Find(key, **this**.m1);
82. }
83. **catch** (KeyNotFoundException e)
84. {
85. }
86. waitChangeHandler.ReleaseMutex();
87. }
88. }
89. **public** **void** Change(T[] arr)
90. {
91. waitChangeHandler.WaitOne();
92. **if** (m1 == **null**)
93. m1 = new T[arr.Length];
94. Array.Copy(arr, m1, arr.Length);
95. waitChangeHandler.ReleaseMutex();
96. findEvent.**Set**();
97. }
99. **public** **void** RunSortMerge(**object** obj)
100. {
101. T[] arr = (T[])obj;
102. sorter.SortMerge(arr);
103. firstSorting = **false**;
104. }
106. **public** **void** RunSortShell(**object** obj)
107. {
108. T[] arr = (T[])obj;
109. sorter.SortShell(arr);
110. secondSorting = **false**;
111. }
112. }
113. **class** EventController<T> : IThreadController<T> **where** T : System.IComparable<T>
114. {
115. AutoResetEvent waitChangeHandler = new AutoResetEvent(**false**);
116. AutoResetEvent waitFindHandler = new AutoResetEvent(**true**);
117. **volatile** **bool** firstSorting = **true**, secondSorting = **true**;
118. **private** Sorter<T> sorter;
119. **private** Finder<T> finder;
120. **volatile** T[] m1;
121. **public** EventController()
122. {
123. sorter = new Sorter<T>(**this**);
124. finder = new Finder<T>();
125. }
127. **public** **void** RunFind(**object** obj)
128. {
129. T key = (T)obj;
130. **while** (firstSorting || secondSorting) {
131. waitChangeHandler.WaitOne();
132. waitChangeHandler.Reset();
133. **try**
134. {
135. T foundKey = finder.Find(key, **this**.m1);
136. }
137. **catch** (KeyNotFoundException e)
138. {
139. }
140. waitFindHandler.**Set**();
141. }
142. }
143. **public** **void** Change(T[] arr)
144. {
145. waitFindHandler.WaitOne();
146. waitFindHandler.Reset();
147. **if** (m1 == **null**)
148. m1 = new T[arr.Length];
149. Array.Copy(arr, m1, arr.Length);
150. waitChangeHandler.**Set**();
151. }
153. **public** **void** RunSortMerge(**object** obj)
154. {
155. T[] arr = (T[]) obj;
156. sorter.SortMerge(arr);
157. firstSorting = **false**;
158. }
160. **public** **void** RunSortShell(**object** obj)
161. {
162. T[] arr = (T[])obj;
163. sorter.SortShell(arr);
164. secondSorting = **false**;
165. }
166. }
167. **class** CriticalSectionController<T> : IThreadController<T> **where** T : System.IComparable<T>
168. {
169. **bool** changingInitially = **true**;
170. **volatile** **bool** firstSorting = **true**, secondSorting = **true**;
171. **private** Sorter<T> sorter;
172. **private** Finder<T> finder;
173. **volatile** **static** **object** changeLocker = new **object**();
174. AutoResetEvent findEvent = new AutoResetEvent(**false**);
175. **volatile** T[]  m1;
176. **public** CriticalSectionController()
177. {
178. sorter = new Sorter<T>(**this**);
179. finder = new Finder<T>();
180. }
181. **public** **void** RunFind(**object** obj)
182. {
183. T key = (T)obj;
184. **while** (firstSorting || secondSorting)
185. {
186. **bool** findingInitially = **true**;
187. findEvent.WaitOne();
188. **lock** (changeLocker)
189. {
190. **if** (findingInitially)
191. {
192. Monitor.PulseAll(changeLocker);
193. findingInitially = **false**;
194. }
195. **try**
196. {
197. T foundKey = finder.Find(key, **this**.m1);
199. }
200. **catch** (KeyNotFoundException e)
201. {
203. }
204. Monitor.PulseAll(changeLocker);
205. }
207. }
208. }
209. **public** **void** Change(T[] arr)
210. {
211. **lock** (changeLocker)
212. {
213. **if** (!changingInitially)
214. Monitor.Wait(changeLocker, 100);
215. **else** changingInitially = **false**;
216. **if** (m1 == **null**)
217. m1 = new T[arr.Length];
218. Array.Copy(arr, m1, arr.Length);
219. findEvent.**Set**();
220. }
221. }
223. **public** **void** RunSortMerge(**object** obj)
224. {
225. T[] arr = (T[])obj;
226. sorter.SortMerge(arr);
227. firstSorting = **false**;
228. }
230. **public** **void** RunSortShell(**object** obj)
231. {
232. T[] arr = (T[])obj;
233. sorter.SortShell(arr);
234. secondSorting = **false**;
235. }
236. }

**Пример выполнения программы:**



**Вывод:**

В ходе выполнения курсовой работы были изучены средства синхронизации потоков такие как: критические секции, мьютексы, семафоры и события. На практике были изучены различия между ними, а также исследовано время выполнения в зависимости от способа синхронизации.