|  |  |
| --- | --- |
| 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 № 5**

|  |  |
| --- | --- |
| **Тема**  Взаимодействие параллельных процессов  **Студент** Жигалкин Д.Р  **Группа** ИУ7-55Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_**  **Преподаватель** Рязанова Н.Ю |  |

Москва.

2020 г.

**Задание на лабораторную работу:**

1. Написать программу, реализующую задачу «Производство-потребление» по алгоритму Э. Дейкстры с тремя семафорами: двумя считающими и одним бинарным. В программе должно создаваться не менее 3х процессов - производителей и 3х процессов – потребителей. В программе надо обеспечить случайные задержки выполнения созданных процессов. В программе для взаимодействия производителей и потребителей буфер создается в разделяемом сегменте. Обратите внимание на то, чтобы не работать с одиночной переменной, а работать именно с буфером, состоящим их N ячеек по алгоритму. Производители в ячейки буфера записывают буквы алфавита по порядку. Потребители считывают символы из доступной ячейки. После считывания буквы из ячейки следующий потребитель может взять букву из следующей ячейки.
2. Написать программу, реализующую задачу «Читатели – писатели» по монитору Хоара с четырьмя функциями: Начать\_чтение, Закончить\_чтение, Начать\_запись, Закончить\_запись. В программе всеми процессами разделяется одно единственное значение в разделяемой памяти. Писатели ее только инкрементируют, читатели могут только читать значение.

Для реализации взаимоисключения используются семафоры

**Общее требование к обеим программам.**

В программах осуществляется консольный вывод, т.е. никакого интерфейса не нужно. Работающая программа должна выводить на экран какой процесс что записал, какой процесс что считал.

**Задача «Производство-потребление»**

Листинг 1.1 — Программа, реализующая алгоритм задачи «Производство-потребление»

1. #include <stdio.h>
2. #include <unistd.h>
3. #include <sys/sem.h>
4. #include <sys/stat.h>
5. #include <sys/shm.h>
6. #include <stdlib.h>
7. #include <sys/wait.h>
8. #define COUNT\_PRODUCERS 4
9. #define COUNT\_CONSUMERS 4
10. #define BIN\_SEM 0
11. #define BUFFER\_EMPTY 1
12. #define BUFFER\_FULL 2
13. #define BUF\_SIZE 5
14. struct sembuf producer\_P[] =
15. {
16. {BUFFER\_EMPTY, -1, 0},
17. {BIN\_SEM, -1, 0}
18. };
19. struct sembuf producer\_V[] =
20. {
21. {BIN\_SEM, 1, 0},
22. {BUFFER\_FULL, 1, 0}
23. };
24. struct sembuf consumer\_P[] =
25. {
26. {BUFFER\_FULL, -1, 0},
27. {BIN\_SEM, -1, 0}
28. };
29. struct sembuf consumer\_V[] =
30. {
31. {BIN\_SEM, 1, 0},
32. {BUFFER\_EMPTY, 1, 0}
33. };
34. void Producer(int nomer, int sem\_id, char\* buf, int\* pos, int\* character)
35. {
36. while (1)
37. {
38. // производит единичный объект
39. // ждет, когда освободится хотя бы одна ячейка буфера
40. // и когда или другой производитель,
41. // или потребитель выйдет из критической секции
42. semop(sem\_id, producer\_P, 2);
44. // положить в буфер
45. buf[\*pos] = 'a' + \*character;
46. printf("Producer #%d -> ", nomer);
47. printf("put buffer[%d] = %c\n", \*pos, buf[\*pos]);
48. \*pos = \*pos == BUF\_SIZE - 1 ? 0 : \*pos + 1;
49. \*character = \*character == 25 ? 0 : \*character + 1;
50. // освобождение критической секции и инкремент количества заполненных ячеек
51. semop(sem\_id, producer\_V, 2);
52. sleep(rand() % 2);
53. }
54. }
55. void Consumer(int nomer, int sem\_id, char\* buf, int\* pos)
56. {
57. while (1)
58. {
59. // ждет, когда будет заполнена хотя бы одна ячейка буфера
60. // и когда или потребитель,
61. // или другой производитель выйдет из критической секции
62. semop(sem\_id, consumer\_P, 2);
64. // взять из буфера
65. printf("Consumer #%d <- ", nomer);
66. printf("take buffer[%d] = %c\n", \*pos, buf[\*pos]);
67. \*pos = \*pos == BUF\_SIZE - 1 ? 0 : \*pos + 1;
69. // освобождение критической секции и инкремент количества пустых ячеек
70. semop(sem\_id, consumer\_V, 2);
71. sleep(rand() % 4);
72. }
73. }
74. int main()
75. {
76. int perms = S\_IRWXU | S\_IRWXG | S\_IRWXO;
77. int shm\_id;
78. int sem\_id;
79. char \*mem\_ptr = -1;
80. // возвращает идентификатор разделяемому сегменту памяти
81. if ((shm\_id = shmget(IPC\_PRIVATE, BUF\_SIZE \* sizeof(char) + 3 \* sizeof(int), IPC\_CREAT | perms)) == -1)
82. {
83. perror("Unable to create a shared area.\n");
84. return 1;
85. }
86. // возвращает указатель на сегмент разделяемой памяти
87. if ((mem\_ptr = shmat(shm\_id, NULL, 0)) == -1)
88. {
89. perror("Can't attach memory.\n");
90. return 1;
91. }
92. // создание набора семафоров
93. if ((sem\_id = semget(IPC\_PRIVATE, 3, perms)) == -1)
94. {
95. perror("Can’t semget.\n");
96. return 1;
97. }
98. // изменение управляющих параметров набора семафоров
99. semctl(sem\_id, BIN\_SEM, SETVAL, 1);
100. semctl(sem\_id, BUFFER\_EMPTY, SETVAL, BUF\_SIZE);
101. semctl(sem\_id, BUFFER\_FULL, SETVAL, 0);
102. // создание процессов
103. int count\_processes = 0;
104. pid\_t pid;
106. int\* producer\_pos = mem\_ptr + BUF\_SIZE;
107. int\* character = producer\_pos + 1;
108. int\* consumer\_pos = producer\_pos + 2;
109. \*producer\_pos = 0;
110. \*consumer\_pos = 0;
111. \*character = 0;
112. for (int i = 0; i < COUNT\_PRODUCERS; i++)
113. {
114. if ((pid = fork())== -1)
115. {
116. perror("Can’t fork.\n");
117. return 1;
118. }
119. if (!pid)
120. Producer(i + 1, sem\_id, mem\_ptr, producer\_pos, character);
121. else
122. count\_processes++;
123. }
124. for (int i = 0; i < COUNT\_CONSUMERS; i++)
125. {
126. if ((pid = fork())== -1)
127. {
128. perror("Can’t fork.\n");
129. return 1;
130. }
131. if (!pid)
132. Consumer(i + 1, sem\_id, mem\_ptr, consumer\_pos);
133. else
134. count\_processes++;
135. }
136. // ожидание завершения процессов
137. int status;
138. for (int i = 0; i < count\_processes; i++)
139. {
140. wait(&status);
141. if (!WIFEXITED(status))
142. printf("exit-error, code = %d\n", status);
143. }
145. // освобождение ресурсов
146. shmdt(mem\_ptr);
147. semctl(sem\_id, 0, IPC\_RMID);
148. shmctl(shm\_id, IPC\_RMID, 0);
149. return 0;
150. }

Результат работы программы «Производство-потребление»:



**Задача «Читатели-писатели»**

Листинг 1.2 — Программа, реализующая алгоритм задачи «Читатели-писатели»

1. #include <stdio.h>
2. #include <unistd.h>
3. #include <sys/sem.h>
4. #include <sys/stat.h>
5. #include <sys/shm.h>
6. #include <stdlib.h>
7. #include <sys/wait.h>
8. #define COUNT\_READERS 4
9. #define COUNT\_WRITERS 4
10. #define ACTIVE\_READERS 0
11. #define ACTIVE\_WRITER 1
12. #define WAITING\_WRITERS 2
13. #define WAITING\_READERS 3
14. struct sembuf start\_writer[] =
15. {
16. {WAITING\_WRITERS, 1, 0},
17. {ACTIVE\_READERS, 0, 0},
18. {ACTIVE\_WRITER, 0, 0},
19. {ACTIVE\_WRITER, 1, 0},
20. {WAITING\_WRITERS, -1, 0},
21. };
22. struct sembuf stop\_writer[] =
23. {
24. {ACTIVE\_WRITER, -1, 0}
25. };
26. void start\_write(int sem\_id)
27. {
28. semop(sem\_id, start\_writer, 5);
29. }
30. void stop\_write(int sem\_id)
31. {
32. semop(sem\_id, stop\_writer, 1);
33. }
34. struct sembuf start\_reader[] =
35. {
36. {WAITING\_READERS, 1, 0},
37. {WAITING\_WRITERS, 0, 0},
38. {ACTIVE\_WRITER, 0, 0},
39. {ACTIVE\_READERS, 1, 0},
40. {WAITING\_READERS, -1, 0},
41. };
42. struct sembuf stop\_reader[] =
43. {
44. {ACTIVE\_READERS, -1, 0}
45. };
46. void start\_read(int sem\_id)
47. {
48. semop(sem\_id, start\_reader, 5);
49. }
50. void stop\_read(int sem\_id)
51. {
52. semop(sem\_id, stop\_reader, 1);
53. }
54. void Reader(int nomer, int sem\_id, int\* buf)
55. {
56. while (1)
57. {
58. start\_read(sem\_id);
59. printf("Reader #%d -> ", nomer);
60. printf("read %d\n", \*buf);
61. stop\_read(sem\_id);
62. sleep(rand() % 2);
63. }
64. }
65. void Writer(int nomer, int sem\_id, int\* buf)
66. {
67. while (1)
68. {
69. start\_write(sem\_id);
70. printf("Writer #%d -> ", nomer);
71. printf("write %d\n", ++(\*buf));
72. stop\_write(sem\_id);
73. sleep(rand() % 3);
74. }
75. }
76. int main()
77. {
78. int perms = S\_IRWXU | S\_IRWXG | S\_IRWXO;
79. int shm\_id;
80. int sem\_id;
81. int \*mem\_ptr;
82. // возвращает идентификатор разделяемому сегменту памяти
83. if ((shm\_id = shmget(IPC\_PRIVATE, sizeof(int), IPC\_CREAT | perms)) == -1)
84. {
85. perror("Unable to create a shared area.\n");
86. return 1;
87. }
88. // возвращает указатель на сегмент разделяемой памяти
89. if ((mem\_ptr = shmat(shm\_id, NULL, 0)) == -1)
90. {
91. perror("Can't attach memory.\n");
92. return 1;
93. }
94. // создание набора семафоров
95. if ((sem\_id = semget(IPC\_PRIVATE, 4, perms)) == -1)
96. {
97. perror("Can’t semget.\n");
98. return 1;
99. }
100. // изменение управляющих параметров набора семафоров
101. semctl(sem\_id, ACTIVE\_READERS, SETVAL, 0);
102. semctl(sem\_id, ACTIVE\_WRITER, SETVAL, 0);
103. semctl(sem\_id, WAITING\_READERS, SETVAL, 0);
104. semctl(sem\_id, WAITING\_WRITERS, SETVAL, 0);
105. // создание процессов
106. int count\_processes = 0;
107. pid\_t pid;
108. \*mem\_ptr = 0;
109. for (int i = 0; i < COUNT\_WRITERS; i++)
110. {
111. if ((pid = fork())== -1)
112. {
113. perror("Can’t fork.\n");
114. return 1;
115. }
116. if (!pid)
117. Writer(i + 1, sem\_id, mem\_ptr);
118. else
119. count\_processes++;
120. }
121. for (int i = 0; i < COUNT\_READERS; i++)
122. {
123. if ((pid = fork())== -1)
124. {
125. perror("Can’t fork.\n");
126. return 1;
127. }
128. if (!pid)
129. Reader(i + 1, sem\_id, mem\_ptr);
130. else
131. count\_processes++;
132. }
133. // ожидание завершение процессов
134. int status;
135. for (int i = 0; i < count\_processes; i++)
136. {
137. wait(&status);
138. if (!WIFEXITED(status))
139. printf("exit-error, code = %d\n", status);
140. }
142. // освобождение ресурсов
143. shmdt(mem\_ptr);
144. semctl(sem\_id, 0, IPC\_RMID);
145. shmctl(shm\_id, IPC\_RMID, 0);
146. return 0;
147. }

Результат работы программы «Читатели-писатели»:

