

4 массива структур

1. #define BIN\_SEM   0
2. #define BUFFER\_EMPTY 1
3. #define BUFFER\_FULL  2
5. truct sembuf producerP[2] = { {BUFFER\_EMPTY, DEC, SEM\_UNDO},
6. {BIN\_SEM, DEC, SEM\_UNDO}};
8. **struct** sembuf producerV[2] = { {BIN\_SEM, INC, SEM\_UNDO},
9. {BUFFER\_FULL, INC, SEM\_UNDO}};
11. **struct** sembuf consumerP[2] = { {BUFFER\_FULL,  DEC, SEM\_UNDO},
12. {BIN\_SEM,   DEC, SEM\_UNDO}};
14. **struct** sembuf consumerV[2] = { {BIN\_SEM, INC, SEM\_UNDO},
15. {BUFFER\_EMPTY, INC, SEM\_UNDO}};

18. void producer(**const** **int** id)
19. {
20. **while**(1)
21. {
22. sleep(randint(0, PRODUCERS\_DELAY));
24. **if** (semop(semId, producerP, 2) == -1)
25. {
26. perror("semop");
27. exit(1);
28. }
30. /\* положить в буфер \*/
31. **int** symbol = 'A' + \*shm\_producer\_count % ('Z' - 'A');
32. \*(shm + \*shm\_producer\_count) = symbol;
33. printf("Producer-%d (pid %d) produces %c\n", id, getpid(), symbol);
34. (\*shm\_producer\_count)++;
36. **if** (semop(semId, producerV, 2) == -1)
37. {
38. perror("semop");
39. exit(1);
40. }
41. }
42. }
44. **void** consumer(**const** **int** id)
45. {
46. **while**(1)
47. {
48. sleep(randint(0, CONSUMERS\_DELAY));
50. **if** (semop(semId, consumerP, 2) == -1)
51. {
52. perror("semop");
53. exit(1);
54. }
55. /\* взять из буфера \*/
56. printf("\t\t\t\t\tConsumer %d (pid %d) consumes %c\n", id, getpid(), \*(shm + \*shm\_consumer\_count));
57. (\*shm\_consumer\_count)++;
59. **if** (semop(semId, consumerV, 2) == -1)
60. {
61. perror("semop");
62. exit(1);
63. }
64. }
65. }

Полная программа:

1. #include <signal.h>
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <sys/stat.h>
5. #include <sys/sem.h>
6. #include <sys/shm.h>
7. #include <time.h>
8. #include <unistd.h>
9. #include <sys/wait.h>
11. #define BIN\_SEM   0
12. #define BUFFER\_EMPTY 1
13. #define BUFFER\_FULL  2
15. #define DEC -1
16. #define INC 1
18. #define PRODUCERS\_COUNT 3
19. #define CONSUMERS\_COUNT 3
21. #define PRODUCERS\_DELAY 3
22. #define CONSUMERS\_DELAY 2
24. **struct** sembuf producerP[2] = { {BUFFER\_EMPTY, DEC, SEM\_UNDO},
25. {BIN\_SEM, DEC, SEM\_UNDO}};
27. **struct** sembuf producerV[2] = { {BIN\_SEM, INC, SEM\_UNDO},
28. {BUFFER\_FULL, INC, SEM\_UNDO}};
30. **struct** sembuf consumerP[2] = { {BUFFER\_FULL,  DEC, SEM\_UNDO},
31. {BIN\_SEM,   DEC, SEM\_UNDO}};
33. **struct** sembuf consumerV[2] = { {BIN\_SEM, INC, SEM\_UNDO},
34. {BUFFER\_EMPTY, INC, SEM\_UNDO}};

37. #define N 26 /\*'Z' - 'A' \*/
39. #define PERMS S\_IRWXU | S\_IRWXG | S\_IRWXO
41. **int** semId = -1;
42. **int** shmId = -1;
44. **int** \*shm = NULL;
45. **int** \*shm\_producer\_count = NULL;
46. **int** \*shm\_consumer\_count = NULL;
47. **int** \*shm\_symbol\_now = NULL;
49. **int** randint(**int** a, **int** b)
50. {
51. **return** a + rand() % (b - a + 1);
52. }
54. **int** semrel(**int** semId)
55. {
56. **return** semctl(semId, 0, IPC\_RMID, 0);
57. }
59. **int** shmrel(**int** semId)
60. {
61. **return** shmctl(shmId, IPC\_RMID, NULL);
62. }
64. **void** forkChildren(**const** **int** n, **void** (\*func)(**const** **int**))
65. {
66. **for** (**int** i = 0; i < n; ++i)
67. {
68. **const** pid\_t pid = fork();
69. **if** (pid == -1)
70. {
71. perror("fork");
72. exit(1);
73. }
74. **else** **if** (pid == 0)
75. {
76. **if** (func)
77. func(i);
78. exit(1);
79. }
80. }
81. }
83. **void** waitChildren(**const** **int** n)
84. {
85. **for** (**int** i = 0; i < n; ++i)
86. {
87. **int** status;
88. **const** pid\_t child\_pid = wait(&status);
89. **if** (child\_pid == -1)
90. {
91. perror("wait");
92. exit(1);
93. }
94. **if** (WIFEXITED(status))
95. printf("Process %d returns %d\n", child\_pid, WEXITSTATUS(status));
96. **else** **if** (WIFSIGNALED(status))
97. printf("Process %d terminated with signal %d\n", child\_pid, WTERMSIG(status));
98. **else** **if** (WIFSTOPPED(status))
99. printf("Process %d stopped due signal %d\n", child\_pid, WSTOPSIG(status));
100. }
101. }
103. **void** producer(**const** **int** id)
104. {
105. **while**(1)
106. {
107. sleep(randint(0, PRODUCERS\_DELAY));
109. **if** (semop(semId, producerP, 2) == -1)
110. {
111. perror("semop");
112. exit(1);
113. }
115. /\* положить в буфер \*/
116. **int** symbol = 'A' + \*shm\_producer\_count % ('Z' - 'A');
117. \*(shm + \*shm\_producer\_count) = symbol;
118. printf("Producer-%d (pid %d) produces %c\n", id, getpid(), symbol);
119. (\*shm\_producer\_count)++;
121. **if** (semop(semId, producerV, 2) == -1)
122. {
123. perror("semop");
124. exit(1);
125. }
126. }
127. }
129. **void** consumer(**const** **int** id)
130. {
131. **while**(1)
132. {
133. sleep(randint(0, CONSUMERS\_DELAY));
135. **if** (semop(semId, consumerP, 2) == -1)
136. {
137. perror("semop");
138. exit(1);
139. }
140. /\* взять из буфера \*/
141. printf("\t\t\t\t\tConsumer %d (pid %d) consumes %c\n", id, getpid(), \*(shm + \*shm\_consumer\_count));
142. (\*shm\_consumer\_count)++;
144. **if** (semop(semId, consumerV, 2) == -1)
145. {
146. perror("semop");
147. exit(1);
148. }
149. }
150. }
152. **void** initSemaphore()
153. {
154. /\* два считающих семафора + один бинарный \*/
155. semId = semget(IPC\_PRIVATE, 3, IPC\_CREAT | PERMS);
157. **if** (semId == -1)
158. {
159. perror("semget");
160. exit(1);
161. }
162. /\*количество заполненных ячеек равно 0\*/
163. /\*Все ячейки буфера изначально пусты \*/
164. **if** (semctl(semId, BIN\_SEM,   SETVAL, 1) == -1 ||
165. semctl(semId, BUFFER\_EMPTY, SETVAL, N) == -1 ||
166. semctl(semId, BUFFER\_FULL,  SETVAL, 0) == -1)
167. {
168. perror("semctl");
169. exit(1);
170. }
171. }
173. **void** createSharedMemory()
174. {
175. // (N + 3) \* sizeof(int) - kích thước
176. //IPC\_PRIVATE - tạo seg mới
177. shmId = shmget(IPC\_PRIVATE, (N + 2) \* **sizeof**(**int**), IPC\_CREAT | PERMS);
178. **if** (shmId == -1)
179. {
180. perror("shmget");
181. exit(1);
182. }
183. shm = shmat(shmId, 0, 0);
184. **if** (shm == (**void** \*) -1)
185. {
186. perror("shmat");
187. exit(1);
188. }
189. shm\_producer\_count = shm;
190. shm\_consumer\_count = shm + 1;
191. \*shm\_producer\_count = 0;
192. \*shm\_consumer\_count = 0;
193. shm = shm + 2;
194. }
196. **int** main()
197. {
198. initSemaphore();
199. createSharedMemory();
201. forkChildren(PRODUCERS\_COUNT, producer);
202. forkChildren(CONSUMERS\_COUNT, consumer);
204. waitChildren(PRODUCERS\_COUNT + CONSUMERS\_COUNT);
206. shmrel(semId);
207. semrel(semId);
208. }