**《操作系统》实验报告**

徽标, 公司名称

描述已自动生成

**题目: 实验四 线程同步互斥**

**学 院 计算机学院**

**班 级 2021211304**

**学 号 2021212171、2021212484**

**姓 名 杨晨、张梓良**

**2023年 11月**

##### 1 实验内容

Linux 提供了内核信号量、用户态信号量两类进程/线程间同步互斥机制，用户态信号量又包括Pthread /POSIX 信号量、System V信号量。

查阅资料，参照样例程序，采用内核信号量、Pthread信号量、System V信号量机制（三选一），为下面三个问题（三选一）设计正确的解决方案，并编程实现该方案。观察程序运行结果，分析并发进程/线程的运行行为和结果，验证方案正确性。

**题目1. 生产流水线**

An assembly line is to produce a product C with n1=4 part As, and n2=3 part Bs. The worker of machining A and worker of machining B produce m1=2 part As and m2=1 part B independently each time. Then the m1 part As or m2 part B will be moved to a station, which can hold at most N=12 of part As and part Bs altogether. The produced m1 part As must be put onto the station simultaneously. The workers must exclusively put a part on the station or get it from the station. In addition, the worker to make C must get all part of As and Bs, i.e. n1 part As and n2 part Bs, for one product C once.

Using semaphores to coordinate the three workers who are machining part A, part B and manufacturing the product C without deadlock.

It is required that

(1) definition and initial value of each semaphore, and

(2) the algorithm to coordinate the production process for the three workers should be given.

##### 2 实验原理

本题目是生产者-消费者问题的扩展，需要注意题目要求：(a) worker A 一次生产m1=2个A，必须一次性放入工作台，(b) worker C必须一次性获得所需的n1=4个A和n2=3个B。因此，如果当前工作台空位小于m1，worker A被阻塞；如果当前工作台没有足够的A、B，worker C 被阻塞。

采用类似于多次wait(empty)的操作，为worker A从工作台获取多个空位是不合理的。例如，假设当前工作台已经放置了N=11个零件，只有1个空位。worker A生产出2个A，worker，B 生产出1个B。如果worker A先通过两次wait(empty)操作申请两个工作台空位，将被阻塞，生产的零件A没有放入工作台。随后worker B再执行wait(empty) 申请空位也将被阻塞，而此时工作台有1个空位。

因此，正确的解决方案是，

（1）利用用户空间计数变量countA、countB、Numempty，配合互斥信号量mutexA、mutexB、mutexNumempty，统计工作台中零件A、零件B、空单元的数目，当有足够多的零件A、B和工作台空位时，再一次性申请/获取多个所需的零件A、零件B、工作台空位，即将worker A所需的m1个工作台空位作为一个整体一次性地申请，worker C所需的工作台中的n1个A和n2个B作为一个整体一次性地申请；

（2）当所需工作台空位不满足时，worker A和worker B应主动阻塞自身(suspend/block)，防止忙等待。当worker C所需的工作台中A和B不满足时，也应主动阻塞自身；

（3）当worker A、worker B分别生产并放入新的零件A、B后，考虑唤醒由于所需零件不足而处于阻塞态的worker C；同样地，当worker C从工作台取出零件A、零件B后，考虑唤醒因没有足够空位而处于阻塞态的worker A、worker B。

**（4）worker A、worker B向工作台中放入零件还应该考虑至少能够保证有4个A和3个B在工作台中，使得worker C能够取出所需零件而不造成死锁。因此worker A放入零件时应考虑能否保证工作台中至少有3个B，work B放入零件时应考虑能否保证工作台中至少有4个A。综上，worker A最多向工作台中放入12-3=9个A；work B最多向工作台中放入12-4=8个B。**

##### 3 设计思路

**二元信号量：**

mutexA, mutexB // 控制对共享变量countA, countB的互斥读写

mutexEmpyt =1 // 控制对工作台和共享变量empty的互斥访问

**多元信号量：**

suspendA = 0, suspendB = 0, suspendC = 0 // 用于挂起worker A, B, C

**共享变量：**

int countA, countB = 0 // 统计工作台中零件A、零件B的数目

int empty = 12 // 统计工作台的空单元数目

**workerA**

1. void workerA()
2. {
3. while(1)
4. {
5. 制作2个A零件;
6. wait(mutexA); *// 互斥读写countA*
7. wait(mutexEmpty); *// 互斥访问工作台并互斥读写empty*
8. if (empty >= 2 && countA <= 7)
9. {
10. 向工作台中同时放入2个A零件;
11. empty -= 2;
12. countA += 2;
13. signal(mutexEmpty);
14. signal(mutexA);
15. if (C被阻塞 && 当前工作台中的零件满足C的需求)
16. signal(suspendC); *// 解挂C*
17. }
18. else
19. {
20. signal(mutexEmpty);
21. signal(mutexA);
22. wait(suspendA); *// 控制台中无足够空间或不允许再放入A零件，挂起workerA*
23. }
24. }
25. }

**workerB**

1. void workerB()
2. {
3. while(1)
4. {
5. 制作1个B零件;
6. wait(mutexB); *// 互斥读写countB*
7. wait(mutexEmpty); *// 互斥访问工作台并互斥读写empty*
8. if (empty >= 1 && countB <= 7)
9. {
10. 向工作台中放入1个B零件;
11. empty -= 1;
12. countA += 1;
13. signal(mutexEmpty);
14. signal(mutexB);
15. if (C被阻塞 && 当前工作台中的零件满足C的需求)
16. signal(suspendC); *// 解挂C*
17. }
18. else
19. {
20. signal(mutexEmpty);
21. signal(mutexB);
22. wait(suspendB); *// 控制台中无足够空间或不允许再放入B零件，挂起workerB*
23. }
24. }
25. }

**workerC**

1. void workerC()
2. {
3. while(1)
4. {
5. wait(mutexA); *// 互斥读写countA*
6. wait(mutexB); *// 互斥读写countB*
7. if (countA >=4 && countB >=3) *// 有所需的零件*
8. {
9. wait(mutexEmpty); *// 互斥访问工作台并互斥读写empty*
10. 同时取出4个零件A，3个零件B;
11. countA -= 4;
12. countB -= 3;
13. empty += 7;
14. signal(mutexEmpty);
15. signal(mutexB);
16. signal(mutexA);
17. 组装一个零件C;
18. if (A被阻塞) signal(suspendA); *// 解挂A*
19. if (B被阻塞) signal(suspendB); *// 解挂B*
20. }
21. else
22. {
23. signal(mutexB);
24. signal(mutexA);
25. wait(suspendC); *// 工作台中没有足够的零件，挂起workerC*
26. }
27. }
28. }

##### 4 Pthread 线程同步互斥机制及API

Pthread/POSIX 提供了两种线程同步互斥机制：互斥锁和信号量，互斥锁相当于教科书中的binary semphore，取值为 0、1；信号量则是教科书中的counting semaphore，取值为整数。

**互斥锁**

互斥锁控制对共享资源的原子操作，互斥锁有两种状态，即上锁和解锁。在同一个时刻只能有一个线程掌握共享资源对应的互斥锁，拥有上锁状态的线程能够对共享资源进行操作。若其它线程希望上锁一个已经被上锁的互斥锁，则该线程就会被挂起，直到上锁的线程释放掉互斥锁。

Pthread 提供了以下操作互斥锁机的基本函数：

* 互斥锁初始化：pthread\_mutex\_init()
* 互斥锁上锁：pthread\_mutex\_lock()
* 互斥锁判断上锁：pthread\_mutex\_trylock()
* 互斥锁解锁：pthread\_mutex\_unlock()
* 消除互斥锁：pthread\_mutex\_destroy()

有三种类型的互斥锁：**快速互斥锁、递归互斥锁和检错互斥锁**，三种锁的区别在于：当其它未占有互斥锁的线程希望得到互斥锁时是否需要阻塞等待。快速互斥锁是指调用线程会阻塞直至拥有互斥锁的线程解锁为止；递归互斥锁能够成功地返回，并且增加调用线程在互斥上加锁的次数而检错互斥锁则为快速互斥锁的非阻塞版本，它会立即返回并返回一个错误信息。**默认属性为快速互斥锁**。

**信号量**

信号量为教科书中所述的counting semaphore，即计数信号量，即可用于进程/线程间互斥、也可以应用于进程间同步。

进程/线程通过wait()、signal操作（或、PV操作），改变信号量的值，获取共享资源的访问权限，实现进程/线程同步互斥。

Pthread 线程库提供的信号量访问操作有：

* sem\_init()，创建一个信号量，并初始化它的值；
* sem\_wait()和 sem\_trywait()，相当于 wait/P 操作，在信号量>0时，它们能将信号量的值减 1。两者的区别在于信号量，sem\_wait(0) 将会阻塞进程/线程，而sem\_trywait 则会立即返回。
* sem\_post()，相当于 signal/V 操作，它将信号量的值加 1，**同时发出信号来唤醒等待的进程/线程。**
* sem\_getvalue()，得到信号量的值。
* sem\_destroy()，删除信号量。

##### 5 代码实现

1. #include <stdlib.h>
2. #include <stdio.h>
3. #include <semaphore.h>
4. #include <pthread.h>
5. #include <errno.h>
6. */\* 共享变量 \*/*
7. int countA = 0; *// 工作台中零件A的数量*
8. int countB = 0; *// 工作台中零件B的数量*
9. int empty = 12; *// 工作台中空位的数量*
10. */\* 互斥锁 \*/*
11. pthread\_mutex\_t mutexA;     *// 对countA的互斥锁*
12. pthread\_mutex\_t mutexB;     *// 对countB的互斥锁*
13. pthread\_mutex\_t mutexEmpty; *// 对empty的互斥锁同时控制对工作台的互斥访问*
14. */\* 信号量 \*/*
15. sem\_t suspendA;   *// 用于A线程的挂起*
16. sem\_t suspendB;   *// 用于B线程的挂起*
17. sem\_t suspendC;   *// 用于C线程的挂起*
18. sem\_t isSuspendA; *// 用于判断A线程是否被挂起*
19. sem\_t isSuspendB; *// 用于判断B线程是否被挂起*
20. sem\_t isSuspendC; *// 用于判断C线程是否被挂起*
21. */\* workerA生产两个零件A并放入工作台 \*/*
22. void \*workerA(void)
23. {
24. while (1)
25. {
26. sleep(1);
27. pthread\_mutex\_lock(&mutexA);
28. pthread\_mutex\_lock(&mutexEmpty);
29. if (empty >= 2 && countA <= 7) *// 工作台有空位且能保证放下3个零件B*
30. {
31. *// 向工作台放入两个零件A*
32. empty -= 2;
33. countA += 2;
34. printf("====workerA====\n");
35. printf("countA:%d empty=%d\n", countA, empty);
36. int value;
37. sem\_getvalue(&isSuspendC, &value);
38. if (value == 0 && (countA >= 4 && countB >= 3)) *// 如果C线程被挂起且工作台中有足够的零件A和零件B*
39. {
40. sem\_post(&suspendC); *// 唤醒C线程*
41. }
42. pthread\_mutex\_unlock(&mutexEmpty);
43. pthread\_mutex\_unlock(&mutexA);
44. }
45. else
46. {
47. pthread\_mutex\_unlock(&mutexEmpty);
48. pthread\_mutex\_unlock(&mutexA);
49. *// 挂起A线程*
50. sem\_wait(&isSuspendA);
51. printf("A is suspended\n");
52. sem\_wait(&suspendA);
53. printf("A is resumed\n");
54. sem\_post(&isSuspendA);
55. }
56. }
57. }
58. */\* workerB生产一个零件B并放入工作台 \*/*
59. void \*workerB(void)
60. {
61. while (1)
62. {
63. sleep(1);
64. pthread\_mutex\_lock(&mutexB);
65. pthread\_mutex\_lock(&mutexEmpty);
66. if (empty >= 1 && countB <= 7) *// 工作台有空位并能保证放下4个零件A*
67. {
68. *// 向工作台放入一个零件B*
69. empty -= 1;
70. countB += 1;
71. printf("====workerB====\n");
72. printf("countB:%d empty=%d\n", countB, empty);
73. int value;
74. sem\_getvalue(&isSuspendC, &value);
75. if (value == 0 && (countA >= 4 && countB >= 3)) *// 如果C线程被挂起且工作台中有足够的零件A和零件B*
76. {
77. sem\_post(&suspendC); *// 唤醒C线程*
78. }
79. pthread\_mutex\_unlock(&mutexEmpty);
80. pthread\_mutex\_unlock(&mutexB);
81. }
82. else
83. {
84. pthread\_mutex\_unlock(&mutexEmpty);
85. pthread\_mutex\_unlock(&mutexB);
86. *// 挂起B线程*
87. sem\_wait(&isSuspendB);
88. printf("B is suspended\n");
89. sem\_wait(&suspendB);
90. printf("B is resumed\n");
91. sem\_post(&isSuspendB);
92. }
93. }
94. }
95. */\* workerC从工作台中取出4个零件A和3个零件B \*/*
96. void \*workerC(void)
97. {
98. while (1)
99. {
100. sleep(2);
101. pthread\_mutex\_lock(&mutexA);
102. pthread\_mutex\_lock(&mutexB);
103. if (countA >= 4 && countB >= 3) *// 工作台中有足够的零件A和零件B*
104. {
105. pthread\_mutex\_lock(&mutexEmpty);
106. *// 从工作台中取出4个零件A和3个零件B*
107. countA -= 4;
108. countB -= 3;
109. empty += 7;
110. printf("====workerC====\n");
111. printf("countA:%d countB:%d empty=%d\n", countA, countB, empty);
112. pthread\_mutex\_unlock(&mutexEmpty);
113. pthread\_mutex\_unlock(&mutexB);
114. pthread\_mutex\_unlock(&mutexA);
115. int value1, value2;
116. sem\_getvalue(&isSuspendA, &value1);
117. sem\_getvalue(&isSuspendB, &value2);
118. if (value1 == 0) *// 如果A线程被挂起*
119. {
120. sem\_post(&suspendA); *// 唤醒A线程*
121. }
122. if (value2 == 0) *// 如果B线程被挂起*
123. {
124. sem\_post(&suspendB); *// 唤醒B线程*
125. }
126. }
127. else
128. {
129. pthread\_mutex\_unlock(&mutexB);
130. pthread\_mutex\_unlock(&mutexA);
131. *// 挂起C线程*
132. sem\_wait(&isSuspendC);
133. printf("C is suspended\n");
134. sem\_wait(&suspendC);
135. printf("C is resumed\n");
136. sem\_post(&isSuspendC);
137. }
138. }
139. }
140. int main(void)
141. {
142. */\* 线程的标识符 \*/*
143. pthread\_t A, B, C;
144. int ret = 0;
145. */\* 初始化互斥锁 \*/*
146. pthread\_mutex\_init(&mutexA, NULL);
147. pthread\_mutex\_init(&mutexB, NULL);
148. pthread\_mutex\_init(&mutexEmpty, NULL);
149. */\* 初始化信号量 \*/*
150. sem\_init(&suspendA, 0, 0);
151. sem\_init(&suspendB, 0, 0);
152. sem\_init(&suspendC, 0, 0);
153. sem\_init(&isSuspendA, 0, 1);
154. sem\_init(&isSuspendB, 0, 1);
155. sem\_init(&isSuspendC, 0, 1);
156. */\* 分别创建线程ABC \*/*
157. ret = pthread\_create(&A, NULL, (void \*)workerA, NULL);
158. if (ret != 0)
159. {
160. perror("workerA\_create");
161. }
162. ret = pthread\_create(&B, NULL, (void \*)workerB, NULL);
163. if (ret != 0)
164. {
165. perror("workerB\_create");
166. }
167. ret = pthread\_create(&C, NULL, (void \*)workerC, NULL);
168. if (ret != 0)
169. {
170. perror("workerB\_create");
171. }
172. sleep(3);
173. */\* 等待线程ABC的结束 \*/*
174. pthread\_join(A, NULL);
175. pthread\_join(B, NULL);
176. pthread\_join(C, NULL);
177. return 0;
178. }

**在实际代码实现过程中，为了判断进程A，B，C是否被挂起，引入信号量isSuspendA, isSuspendB, isSuspendC。通过Pthread库提供的sem\_getvalue获取上述信号量的值，若值为0说明相应进程被挂起，反之则说明相应进程未被挂起。**

##### 6 运行结果

程序能够稳定运行20分钟以上，证明不存在死锁情况。

程序运行部分结果如下：

====workerA====

countA:2 empty=10

====workerB====

countB:1 empty=9

C is suspended

====workerA====

countA:4 empty=7

====workerB====

countB:2 empty=6

====workerA====

countA:6 empty=4

====workerB====

countB:3 empty=3

C is resumed

====workerA====

countA:8 empty=1

====workerB====

countB:4 empty=0

====workerC====

countA:4 countB:1 empty=7

====workerA====

countA:6 empty=5

====workerB====

countB:2 empty=4

====workerA====

countA:8 empty=2

====workerB====

countB:3 empty=1

====workerC====

countA:4 countB:0 empty=8

====workerA====

countA:6 empty=6

====workerB====

countB:1 empty=5

====workerA====

countA:8 empty=3

====workerB====

countB:2 empty=2

C is suspended

A is suspended

====workerB====

countB:3 empty=1

C is resumed

====workerB====

countB:4 empty=0

====workerC====

countA:4 countB:1 empty=7

A is resumed

====workerB====

countB:2 empty=6

====workerA====

countA:6 empty=4

程序运行结果符合预期，同时当所需工作台空位不满足时，worker A和worker B会主动阻塞自身(suspend/block)，防止忙等待。当worker C所需的工作台中A和B不满足时，也会主动阻塞自身。同时被挂起的进程也会在所需条件满足时被其他进程解挂。