<https://blog.csdn.net/waldmer/article/details/23422943>

pthread\_cond\_wait

多线程的条件变量

条件变量是利用线程间共享的[全局变量](http://baike.baidu.com/view/261041.htm)进行同步的一种机制，主要包括两个动作：一个线程等待"条件变量的条件成立"而挂起；另一个线程使"条件成立"（给出条件成立信号）。为了防止竞争，[条件变量](http://baike.baidu.com/view/4025952.htm)的使用总是和一个[互斥锁](http://baike.baidu.com/view/1461738.htm)结合在一起。

创建和注销

条件变量和[互斥锁](http://baike.baidu.com/view/1461738.htm)一样，都有[静态](http://baike.baidu.com/view/612026.htm)动态两种创建方式，静态方式使用PTHREAD\_COND\_INITIALIZER[常量](http://baike.baidu.com/view/346799.htm)，如下：

pthread\_cond\_t cond=PTHREAD\_COND\_INITIALIZER

动态方式调用[pthread\_cond\_init](http://baike.baidu.com/view/3750966.htm)()函数，API定义如下：

int [pthread\_cond\_init](http://baike.baidu.com/view/3750966.htm" \t "_blank)(pthread\_cond\_t \*cond, pthread\_condattr\_t \*cond\_attr)

尽管POSIX标准中为[条件变量](http://baike.baidu.com/view/4025952.htm)定义了属性，但在LinuxThreads中没有实现，因此cond\_attr值通常为NULL，且被忽略。

等待和激发

int pthread\_cond\_wait(pthread\_cond\_t \*cond, pthread\_mutex\_t \*mutex)

在调用pthread\_cond\_wait()前必须由本[线程](http://baike.baidu.com/view/1053.htm)加锁（pthread\_mutex\_lock()），而在更新条件[等待队列](http://baike.baidu.com/view/4083731.htm)以前，mutex保持锁定状态，并在线程挂起进入等待前解锁。在条件满足从而离开pthread\_cond\_wait()之前，mutex将被重新加锁，以与进入pthread\_cond\_wait()前的加锁动作对应。

pthread\_cond\_signal()激活一个等待该条件的线程，存在多个等待线程时按入队顺序激活其中一个

1. */\**
2. *\*主线程创建子线程（当前子线程为stop停止状态），5秒后主线程唤醒子线程，*
3. *\*10秒后主线程挂起子线程。15秒后主线程再次唤醒子线程，20秒后主线程执行完毕，*
4. *\*等待子线程退出。*
5. *\*/*
6. #include <stdio.h>
7. #include <string.h>
8. #include <pthread.h>
10. #define RUN 1
11. #define STOP 0
13. pthread\_mutex\_t mut = PTHREAD\_MUTEX\_INITIALIZER;
14. pthread\_cond\_t cond = PTHREAD\_COND\_INITIALIZER;
16. int status = STOP;
18. void \*thread\_function(void)
19. {
20. static int i = 0;
21. while(1)
22. {
23. pthread\_mutex\_lock(&mut);
24. while(!status)
25. {
26. pthread\_cond\_wait(&cond, &mut);
27. }
28. pthread\_mutex\_unlock(&mut);
29. printf("child pthread %d\n", i++);
30. if (i == 20)
31. {
32. break;
33. }
34. sleep(1);
35. }
36. }
38. void thread\_resume()
39. {
40. if (status == STOP)
41. {
42. pthread\_mutex\_lock(&mut);
43. status = RUN;
44. pthread\_cond\_signal(&cond);
45. printf("pthread run!\n");
46. pthread\_mutex\_unlock(&mut);
47. }
48. else
49. {
50. printf("pthread run already\n");
51. }
52. }
54. void thread\_pause()
55. {
56. if (status == RUN)
57. {
58. pthread\_mutex\_lock(&mut);
59. status = STOP;
60. printf("thread stop!\n");
61. pthread\_mutex\_unlock(&mut);
62. }
63. else
64. {
65. printf("pthread pause already\n");
66. }
67. }
69. int main()
70. {
71. int err;
72. static int i = 0;
73. pthread\_t child\_thread;
75. if (pthread\_mutex\_init(&mut, NULL) != 0)
76. {
77. printf("mutex init error\n");
78. }
79. if (pthread\_cond\_init(&cond, NULL) != 0)
80. {
81. printf("cond init error\n");
82. }
84. err = pthread\_create(&child\_thread,NULL,(void \*)thread\_function,NULL);
85. if (err != 0)
86. {
87. printf("can't create thread:%s\n", strerror(err));
88. }
90. while(1)
91. {
92. printf("father pthread %d\n", i++);
93. sleep(1);
94. if (i == 5)
95. {
96. thread\_resume();
97. }
98. if (i == 10)
99. {
100. thread\_pause();
101. }
102. if (i == 15)
103. {
104. thread\_resume();
105. }
106. if (i == 20)
107. {
108. break;
109. }
110. }
111. if (0 == pthread\_join(child\_thread, NULL))
112. {
113. printf("child thread is over\n");
114. }
116. return 0;
117. }

版权声明：本博客所有文章均为原创，欢迎交流，欢迎转载；转载请勿篡改内容，并且注明出处，谢谢！ https://blog.csdn.net/waldmer/article/details/23422943