CS241 Lawrence Angrave L15 – CSP Solutions. Condition Variables.

Name these concepts:

"Only one process(/thread) can be in the CS at a time"

"If waiting, then another process can only enter the CS a finite number of times"

"If no other process is in the Critical Section then the process can immediately enter the CS"

**Candidate** #4

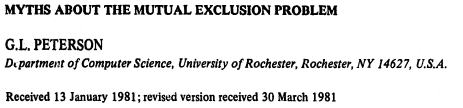
|  |  |
| --- | --- |
| raise my flag  if your flag is raised, wait until my turn  // *Do Critical Section stuff*  turn = *yourid*  lower my flag | raise my flag  if your flag is raised, wait until my turn  // *Do Critical Section stuff*  turn = *yourid*  lower my flag |

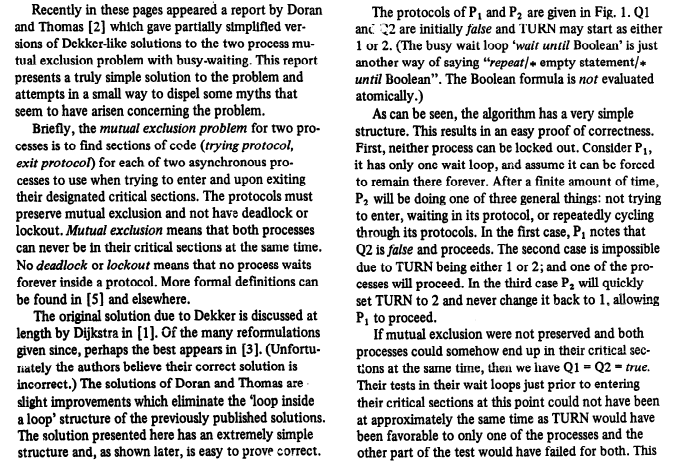
// Threads do other stuff and then will repeat in the future

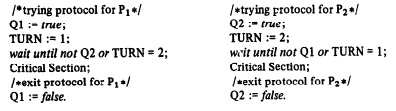
Problems with 4?

**Candidate** #5

|  |  |
| --- | --- |
| raise my flag  while(your flag is raised) :  if it's your turn to win :  lower my flag  wait while your turn  raise my flag  // *Do Critical Section stuff*  set your turn to win  lower my flag | raise my flag  while(your flag is raised) :  if it's your turn to win :  lower my flag  wait while your turn  raise my flag  // *Do Critical Section stuff*  set your turn to win  lower my flag |







**4 Give two reasons why even implementing a 'correct solution' in C might still fail.**

5. What is *Spurious Wakeup*?

6. Complete the following code.

1. pthread\_mutex\_t m = PTHREAD=MUTEX\_INITIALIZER;
2. pthread\_cond\_t cv1, = PTHREAD\_COND\_INITIALIZER,cv2 = ...;
3. double array[10];
4. int n = 0;
5. // blocks while full (n ==10)
6. void push(double v) {
7. p\_m\_lock(&m)
8. while(n == 10) ?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
9. array[n++] = v;
10. if( \_\_\_\_) p\_cond\_signal(\_\_\_\_\_\_);
11. p\_m\_unlock(&m)
12. }
13. // blocks while empty (n == 0)
14. double pop() {
15. p\_m\_lock(&m)
16. while(n == 0) ?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
17. double result = array[--n];
18. if(\_\_\_\_\_\_\_\_) ?
19. p\_m\_unlock(&m);
20. return result;
21. }

8. (Interview question) Use a CV to implement a simple version of a *counting* *semaphore*

Note a real semaphore might implement a queue of waiting threads to ensure fairness (and avoid *starvation*).

|  |  |
| --- | --- |
| sem\_init(sem\_t \*s, int shared, int value) {  } | typedef struct sem\_t {  } sem\_t; |
| sem\_post(sem\_t\*s) {  } | sem\_wait(sem\_t\*s) {  } |

9. What is a barrier?