Critical Sections Lab Assignment

Diaconescu Bogdan-Florin
CR 3.2 A, Anul 3

November 2019

1 Problem statement

Implementarea algoritmului lui Dekker in Promela utilizand 2 procese.

2 Implementation/Solution

Am creat un array "flags" ale carui elemente le-am initializat la inceput cu false si care vor fi utilizare de ambele procese. Variabila "count" este folosita pentru a verifica daca algoritmul rezolva problema excluderii mutuale. Aceasta variabila indica numarul procesului care se afla in sectiunea critica.Atunci cand un proces intra in sectiunea critica, se incrementeaza valoarea lui "count" si se decrementeaza la iesirea procesului din sectiunea critica.

La linia 5 sunt create 2 procese prin folosirea sintaxei "active [2] proctype". Numerotarea proceselor se face incepand de la 0. La linia 8 am utilizat 2 variabile i si j de tipul "_pid" care retin numar de identificare al proceselor. Ambele variabile sunt declarate local in interirorul fiecarui proces.

Cand un proces ajunge la linia 19, acesta nu va trece mai departe pana cand nu este indeplinita conditia respectiva. De asemnea, pentru a evita blocarea programului in bucla "do" sau "if", am explicat ce trebuie sa faca aceasta in cazul in care o conditie nu este indeplinita(linia 21 si linia 23). Instructiunea "skip" indica faptul ca, in cazul in care nu este indeplinita conditia initiala se trece mai departe, pe cand instructiunea "break" este folosita pentru a iesi din bucla "do".

3 Experimental data

Algoritmul lui Dekker rezolva problema sectiunii critice. Acesta permite ca 2 procese sa utilizeze resurse comune fara sa existe conflict intre ele. Mai jos am explicat algoritmul intr-un mod cat mai simplu.

Algorithm 1

1: $flag[2] \leftarrow false$
2: $turn \leftarrow false$
3: $count \leftarrow 0$
3: **function** ACTIVE PROCTYPE MUTEX(())
4: $\_pidi \leftarrow \_pid$
5: $\_pidj \leftarrow 1 - \_pid$
6: infinite loop
7: $flag[i] \leftarrow true$
8: **while** flag[j] == true **do**
9: **if** turn == j **then**
10: $flag[i] \leftarrow false$
11: $(turn! = j)$
12: **else**
skip
13: **end if**
14: *break*
15: **end while**
16: $count + +$
17: $ASSERT(count == 1)$
18: $count - -$
19: $turn \leftarrow j$
20: $flag[i] \leftarrow false$
21: goto again

Algorithm 2

1: $flag[0] \leftarrow true$ // "I want to enter."
2: **while** flag[1] == true **do**
// "If you want to enter
if turn!=0 **then** // and if it's your turn
3: $flag[0] \leftarrow false$ // I don't want to enter any more."
5: **while** turn != 0 **do**
// "If it's your turn
6: **end while**// I'll wait."
7: $flag[0] \leftarrow true$ // "I want to enter."
8: **end if**
9: **end while**
10: Enter CS! // CS
11: $turn \leftarrow 0$ // "You can enter next."
12: $flag[0] \leftarrow false$ // "I don't want to enter any more."
=0

4 Results & Conclusions

Am implementat algoritmul lui Dekker in Promela, rezolvand problema excluderii mutuale. Daca un proces vrea sa intre in sectiunea critica, acesta va astepta pana cand ii vine randul si pana cand celalalt proces a iesit din zona critica.