Problema 2

Vlad Theodor și Petercă Adrian

November 2021

1 Punctul A

Algoritmul pentru 2 threaduri functioneaza corect.
Algoritmul pentru N threaduri functioneaza corect, iar explicatia este urmatoarea:

1.1 Pentru producatori

- un producator se blocheaza in cazul in care TAIL - HEAD == QSIZE (adica avem QSIZE elemente in coada, deci e plina)
- in situatia in care exista mai mult de 1 producator, TAIL - HEAD == QSIZE este o conditie necesara de blocare, dar nu sufiecienta (deoarece toti producatorii vor avea aceeasi verificare). Astfel, este necesara obtinerea lock-ului pentru a garanta accesul unic la resursa partajata (coada), iar la final eliberarea acestuia.

1.2 Pentru consumatori

- un consumator se blocheaza in cazul in care TAIL == HEAD (adica avem 0 elemente in coada - i.e. ea este vida)
- in situatia in care exista mai mult de 1 consumator, la fel ca in cazul producatorului, conditia din WHILE este necesara dar nu suficienta (daca nu avem elemente in coada, e clar ca trebuie sa asteptam. Dar daca avem elemente, ne trebuie un lock pe resursa partajata). Astfel, un consumator blocheaza coada intr-o stare sigura (coada nu este vida), consuma un element si returneaza lacatul.

1.3 Situatii speciale

- producatorii reusesc sa umple coada foarte rapid, iar consumatorii nu apuca sa isi puna lacatul pe resursa.
  Aceasta situatie poate persista pana in momentul in care coada este plina, caz in care producatorii se blocheaza, un consumator va elimina un element din coada, iar producatorii vor reveni la procesul initial.
- consumatorii reusesc sa goleasca coada foarte rapid, iar producatorii nu au timp sa o umple. La fel ca situatia precedenta, cand coada e goala, consumatorii se blocheaza la while (si nu la lock!) caz in care un producator va adauga un element in coada.

Observatie importanta: fata de primul algoritm (cel cu un singur lacat, pentru grupele TPM1-TPM2), algoritmul acesta este mai 'thread-oriented', deoarece nu punem lacatul pe toata coada (practic, nu blocam toata resursa) ci blocam fix portiunea care ne intereseaza. Un producator

si un consumator isi pot face treaba separat, neinfluentandu-se negativ (aceasta este asigurata de conditiile din WHILE).

2 Punctul B

Presupunem ca avem doua threaduri A si B.
Initial, variabilele arata astfel:

```
flag = [False, False]
label = [0, 0]
```

Atat A cat si B apeleaza lock(0), respectiv lock(1) simultan. Atunci, rezulta:

```
flag = [True, True]
label = [1, 1]
```

Daca nu ar exista conditia de verificare atat pentru label, cat si pentru indecsi, atunci am avea situatia in care $label[0] > label[1]$ este **FALSE**, deci ambele threaduri ar intra in sectiunea critica in acelasi timp, ceea ce nu este corect.

Insa prin folosirea indecsilor, se garanteaza o ordine prestabilita (A inainte de B), iar primul care se va executa va fi A, urmat de B.

3 Punctul C

Daca facem lock() in interiorul lui TRY-CATCH, in situatia in care lock() arunca o exceptie (asta inseamna ca NU am reusit sa punem lacatul) in FINALLY vom apela unlock() pe un lacat nepus, ceea ce este o practica gresita. Din acest motiv se pune lock() inainte de TRY-CATCH.