**Lab 9 lacate**

**9.2.1 concept**

are 2 stari**: locked si unlocked**

In general e folosit pt a sincroniza thread-uri din acelasi proces

Cand un thread doreste sa intre intr-o reg critica va incerca sa blockeze lacatul ; daca reuseste, trece mai departe ;  **daca lacatul e deja blocat, thread-ul curent va fi blocat pana ce lacatul se deblocheaza**

la iesirea din reg critica , thread-ul ce a blocat lacatul trebuie sa il si deblocheze **( NICI UN ALT THREAD NU POATE DEBLOCA LACATUL!!)**

**9.2.2 initializare/stergere lacat**

**#include <pthread.h>**

**int pthread\_mutex\_init(pthread\_mutex\_t \*restrict mutex,const pthread\_mutexattr\_t \*restrict attr);**

**pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;**

**int pthread\_mutex\_destroy(pthread\_mutex\_t \*mutex);**

**9.2.2.1 pthread\_mutex\_init**

**int pthread\_mutex\_init(pthread\_mutex\_t \*restrict mutex,const pthread\_mutexattr\_t \*restrict attr);**

* Initializeza mutex-ul primit ca pointer prin **attr (**atribute implcite => attr=NULL)
* Daca initializarea s-a realizat cu **succes, functia returneaza 0 🡺 lacatul se afla in starea UNLOCKED**

**9.2.2.2 PTHREAD\_MUTEX\_INITIALIZER**

**🡪** foloist la initializarea mutex-ului cu atribute implicite se folosescte macro-ul PTHREAD\_MUTEX\_INITIALIZER, de tip pthread\_mutex\_t

**9.2.2.3 pthread\_mutex\_destroy**

**int pthread\_mutex\_destroy(pthread\_mutex\_t \*mutex);**

**🡪** folosita la stergerea si eliberarea resurselor mutex-ului

Functia thread-ului fara protectia regiunii critice:

1 void \*thread\_function(void \*arg)

2 {

3 pthread\_mutex\_t \*lock = (pthread\_mutex\_t\*)arg;

4

5 //common region

6 sleep(1);

7

8 //critical region

9 sleep(1);

10

11 return NULL;}

$ time ./l09p1\_mutex

real 0m2,005s

* Cele 4 comenzi se executa in paralel ( inclusiv regiunea critica) ; fiecare thread a domit 2 secunde si cum totul s-a executat in paralel, timpul real de executie a fos de aprox 2 secunde

**9.2.2.4 exemplu**

🡪 4 thread-uri ce executa aceeasi functie

**1 int main()**

**2 {**

**3 int i;**

**4 pthread\_t tids[NR\_THREADS];**

**5 pthread\_mutex\_t lock;**

**6**

**7 if(pthread\_mutex\_init(&lock, NULL) != 0) {** // initializare mutex ( se putea folosi si marcoul)

**8 perror("error initializing the mutex");**

**9 return 1;**

**10 }**

**11 for(i=0; i<NR\_THREADS; i++) {**

**12 pthread\_create(&tids[i], NULL, thread\_function, &lock);**

**13 }**

**14 for(i=0; i<NR\_THREADS; i++) {**

**15 pthread\_join(tids[i], NULL);**

**16 }**

**17 pthread\_mutex\_destroy(&lock);** // lacatul e distrus la final

**18**

**19 return 0;**

**20 }**

**9.2.3 operatii pe lacate**

**#include <pthread.h>**

**int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);**

**int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);**

**int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);**

**9.2.3.1 pthread\_mutex\_lock**

🡪 apel ce realizeaza blocarea lacatului

* Daca lacatul e deja blocat, thread-ul curent va fi pus in asteptare pana la deblocarea acestuia **de catre thread-ul ce il tine blocat**
  + - 1. **pthread\_mutex\_unlock**

**🡪** se realizeaza blocarea lacatului

**🡪 doar thread-ul ce a blocat lacatul are voie sa il deblocheze**

**9.2.3.2 pthread\_mutex\_trylock**

* + se incearca blocarea lacatului; in cazul in care era deja liber, comportamentul e identic.
  + Daca lacatul dera deja blocat, **functia nu deblocheaza thread-ul curent , ci returneaza o valoare diferita de 0**
    - 1. **protejare reg critica**

Pointerul spre lacat se obtine facand cast arg primit la tipul pthread\_mutex\_t (lin3)

Reg critica incepe prin blocarea lacatului si se termina prin deblocarea lui

$ time ./l09p1\_mutex

real 0m5,004s

user 0m0,001s

sys 0m0,003s

* Timpul de executie a fost de aprox 5 secunde; regiunea comuna a fost executata in paralel timp de o secunda;
* Reg critica a fost executata de fiecare thread pe rand timp de o secunda 🡺 4 secunde

**1 void \*thread\_function(void \*arg)**

**2 {**

**3 pthread\_mutex\_t \*lock = (pthread\_mutex\_t\*)arg;**

**5 //common region**

**6 sleep(1);**

**8 //critical region**

**9 pthread\_mutex\_lock(lock);**

**10 sleep(1);**

**11 pthread\_mutex\_unlock(lock);**

**12**

**13 return NULL;**

**14 }**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* + 1. **VARIABILE CONDITIONATE**
* Problema: thread-ul trebuie sa astepte indeplinirea unei conditii pt a trece mai departe
* Vb implicate in conditie sunt protejate de un lacat 🡺 thread-ul trb sa blocheze lacatul ca sa poata acesa vb
* Thread-ul ce ar putea modifica variabilele implicate in conditie are nevoie de acelasi lacat
* Daca thread-ul care asteapta indeplinirea conditiei ar tine lacatul blocat s-ar ajunge la deadlock
* Vb conditionate se bazeaza pe primitiva **wait ce poate debloca lacatul pe perioada cat thread-ul curent ateapta, dand sansa altor thread-uri sa modifice vb implicate in conditie**
* Atunci cand un alt thread **realizeaza indeplinirea conditiei**, **poate semnaliza vb cond prin primitiva signal si thread-ul ce asteapta va fi trezit**

**9.2.5 initializare/stergere vb cond**

**#include <pthread.h>**

**int pthread\_cond\_init(pthread\_cond\_t \*restrict cond,**

**const pthread\_condattr\_t \*restrict attr);**

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

**int pthread\_cond\_destroy(pthread\_cond\_t \*cond);**

* + 1. **pthread\_cond\_init**
* initializeaza vb conditionata primita ca pointer folosit atributele **attr (=NULL** daca se folosesc atrbuitele implicite)
  + - 1. **PTHREAD\_COND\_INITIALIZER**

**🡪** macro pentru initializarea unei vb cond cu atrbute implicite

* + - 1. **pthread\_cond\_destroy**

**🡪** elibereaza resuresele vb conditionate primite ca param

* + 1. **operatii pe variabile conditionate**

**#include <pthread.h>**

**int pthread\_cond\_wait(pthread\_cond\_t \*restrict cond,**

**pthread\_mutex\_t \*restrict mutex);**

**int pthread\_cond\_timedwait(pthread\_cond\_t \*restrict cond,**

**pthread\_mutex\_t \*restrict mutex,**

**const struct timespec \*restrict abstime);**

**int pthread\_cond\_signal(pthread\_cond\_t \*cond);**

**int pthread\_cond\_broadcast(pthread\_cond\_t \*cond);**

* + - 1. **pthread\_cond\_wait**
* pthread\_cond\_wait() pune thread-ul curent in asteptare pana ce se apeleza functia pthread\_cond\_signal() sau \_broadcast() **cu aceeasi variabila conditionata ca parametru al celor 2 functii**
* parametrul **mutex** trebuie sa fie un pointer spre un lacat valid ,  **care a fost blocat in prealabil de thr curent.**
* La intrarea **in starea de asteptare** a thread-ului curent **lacatul va fi deblocat** si la trezirea thread-ului si reluarea lui (dar inainte de revenirea din wait) , el **va fi blocat din nou**

**9.2.6.1 pthread\_cond\_timewait**

**🡪** pthreac\_cond\_timewait() are un comportament similar cu pthread\_cond\_wait() dar foloseste in pluc parametrul **abstime** care specifica un **moment de timp in viitor**

**🡪 daca pana la momentul curent de timp specificat nu se primeste semnalul , thread-ul va fi trezit oricum**

**9.2.6.2 pthread\_cond\_signal**

🡪trezeste **primul thread pus in asteptare** de vb conditionata primita ca param

**9.2.6.3 pthread\_cond\_broadcast**

**🡪** trezeste **TOATE** thread-urile puse in asteptare de vb conditionata

**9.2.6.3 exemplu 🡪** op: deposit si withdraw ; balanta contului nu poate fi negativa deci un thread trb sa astepte ca suma dorita sa existe in cont ; 🡪 7 thr (0,2,3,4 extrag 7 lei si 1,5,6 depun 11 lei)

**1 #define NR\_THREADS 7**

**1 #define AMOUNT\_WITHDRAW 7**

**3 void \*thread\_withdraw(void \*arg)**

**4 {**

**5 TH\_STRUCT \*s = (TH\_STRUCT\*)arg;**

**6 usleep(10000 \* s->id);**

**8 pthread\_mutex\_lock(s->lock);**

**9 while(balance < AMOUNT\_WITHDRAW) {**

**10 printf("[Th%d] Not enough money (%d). Will wait...\n", s->id, balance);**

**11 pthread\_cond\_wait(s->cond, s->lock);**

**12 }**

**13 balance -= AMOUNT\_WITHDRAW;**

**14 printf("[Th%d] Withdrawn %d. Current balance is %d.\n",**

**15 s->id, AMOUNT\_WITHDRAW, balance);**

**16 pthread\_mutex\_unlock(s->lock);**

**18 return NULL;**

**19 }**

**🡪functia thread\_withdraw** protejeaza accesul la variabila balance folosind lacatul lock

🡪 cat timp balanta e mai mica decat suma dorita (lin9) thread-ul curent trb sa astepte sa se mai depuna bani (lin11)

🡪 **NU** putem astepta folosind un lacat sau un semafor  **deoarece suntem in interiorul reg critice, am blocat lock deci nici un alt thread nu ar putea ajunge sa modifice balanta**

**🡪 in schimb, vb conditionata realizeaza deblocarea lacatului pe perioada in care thread-ul curent e blocat, permitand altor thread-uri sa modifice vb balance**

**3 typedef struct {**

**4 int id;**

**5 pthread\_mutex\_t \*lock;**

**6 pthread\_cond\_t \*cond;**

**7 } TH\_STRUCT;**

**9 int main()**

**10 {**

**11 int i;**

**12 TH\_STRUCT params[NR\_THREADS];**

**13 pthread\_t tids[NR\_THREADS];**

**14 pthread\_mutex\_t lock = PTHREAD\_MUTEX\_INITIALIZER;**

**15 pthread\_cond\_t cond = PTHREAD\_COND\_INITIALIZER;**

**16**

**17 for(i=0; i<NR\_THREADS; i++) {**

**18 params[i].id = i;**

**19 params[i].lock = &lock;**

**20 params[i].cond = &cond;**

**21 if(i == 0 || i == 2 || i == 3 || i == 4) {**

**22 pthread\_create(&tids[i], NULL, thread\_withdraw, &params[i]);**

**23 } else {**

**24 pthread\_create(&tids[i], NULL, thread\_deposit, &params[i]);**

**25 } }**

**28 for(i=0; i<NR\_THREADS; i++) {**

**29 pthread\_join(tids[i], NULL);**

**30 }**

**32 pthread\_mutex\_destroy(&lock);**

**33 pthread\_cond\_destroy(&cond);**

**35 return 0;**

**36 }**

🡪 thread\_deposit() are un comportament similar cu thread\_withdraw()

🡪depunerea banilor se poate face oricand , deci thread-ul nu trebuie sa astepte

🡪 in caz ca suma e suficienta pentru mai multe retraferi (lin12) sunt trezite **toate thread-urile,** prin **broadcast** ; daca numa e suficienta pentru o singura tranzactie, atunci un singur thread e trezit(lin 15)

🡪 faptul ca thread-ul asteapta la inceput un timp dp cu id-ul acestuia , **creste sansele ca intrarea in reg critica sa se faca in ordinea id-urilor** (acest lucru nu e garantat)

**1 #define AMOUNT\_DEPOSIT 11**

**2**

**3 void \*thread\_deposit(void \*arg)**

**4 {**

**5 TH\_STRUCT \*s = (TH\_STRUCT\*)arg;**

**6 usleep(10000 \* s->id);**

**8 pthread\_mutex\_lock(s->lock);**

**9 balance += AMOUNT\_DEPOSIT;**

**10 printf("[Th%d] Deposited %d. Current balance is %d.\n",**

**11 s->id, AMOUNT\_DEPOSIT, balance);**

**12 if(balance >= 2 \* AMOUNT\_WITHDRAW) {**

**13 pthread\_cond\_broadcast(s->cond);**

**14 } else if (balance >= AMOUNT\_WITHDRAW) {**

**15 pthread\_cond\_signal(s->cond);**

**16 }**

**17 pthread\_mutex\_unlock(s->lock);**

**19 return NULL;**

**20 }**

Rezultate:

[Th0] Not nough money (0). Will wait...

[Th1] Deposited 11. Current balance is 11.

[Th0] Withdrawn 7. Current balance is 4.

[Th2] Not enough money (4). Will wait...

[Th3] Not enough money (4). Will wait...

[Th4] Not enough money (4). Will wait...

[Th5] Deposited 11. Current balance is 15.

[Th2] Withdrawn 7. Current balance is 8.

[Th4] Withdrawn 7. Current balance is 1.

[Th3] Not enough money (1). Will wait...

[Th6] Deposited 11. Current balance is 12.

[Th3] Withdrawn 7. Current balance is 5.

**Intrebari recap**

**1.diferenta semafor si lacat?**

R: 🡪 un lacat nu poate fi deblocat decat de thread-ul ce l-a blocat, in schimb un semafor poate fi deblocat atata timp cat exista permisiune, indiferent de ce semafor a acordat acea permisiune

🡪 in regiunea critica intr-un lacat se poate afla exact 1 thread, in schimb la semafoare se pot afla atatea thread-uri cate permisiuni am initializat semaforul

2. **optiuni pentru initializarea unui lacat/vb conditionata**

**R:** lacat**: pthread\_mutex\_init(mutex,atribute)** / **PTHREAD\_MUTEX\_INITIALIZER** pentru cu atr implicite

Vb conditionata: **pthread\_cont\_init(cond,attr) / PTHREAD\_CONT\_INITIALIZER**  -- // --

3. de ce pthread\_cond\_wait primeste un mutex ca parametru? Ce face cu el?

🡪 parametru e folosit pt a sti ce lacat trebuie blocat in cazul in care conditia nu e satisfacuta

4. diferenta signal si broadcast ?

🡪 signal – trezeste primul thread pus In asteptare de vb cond

🡪 broadcast – trezeste toate thread-urile puse in asteptare de vb cond

5. de ce while?