**lab8 semafoare**

**8.2.1 concept**

semafor=**mecanism de sincronizare a executiei mai multor procese ce folosesc aceeasi resursa**

regiune critica=**zona din codul unui program in care resursa comuna e utilizata**

accesul unui proces la o regiune critica trebuie restrictionat

* componente semafor :

**🡪perm=intreg --> nr de permisiuni asociate cu semaforul respectiv**

**🡪q =coada --> lista de procese ce asteapta sa primeasca permisiuni de la semaforul respectiv**

* manipularea componentelor:

**🡪 wait (down/P) --> aceasta primitiva incearca sa decrementeze numarul de permisiuni daca acest nr e o val pozitiva, altfel procesul curent e pus in coada de asteptare si suspendat**

**while(perm==0){**

**q.push(crt\_proc);**

**crt\_proc.suspend();**

**}**

**perm=perm-1;**

**🡪post (up/V) --> incrementeaza numarul de permisiuni, iar primul proces din coada e trezit pentru a-si continua executia**

**perm=perm+1;**

**if(!q.empty()){**

**proc = q.pop();**

**proc.resume();**

**}**

Obs: pseudocodul pentru cele 2 operatii se executa in mod atomic

**8.2.2 crearea,deschiderea,inchiderea semafoarelor**

**#include <fcntl.h>**

**#include <sys/stat.h>**

**#include <semaphore.h>**

**sem\_t \*sem\_open(const char \*name, int oflag);**

**sem\_t \*sem\_open(const char \*name, int oflag, mode\_t mode, unsigned int value);**

**int sem\_close(sem\_t \*sem);**

**int sem\_unlink(const char \*name);**

**🡪 !!! obligatoriu optiunea -pthread la compilare**

**8.2.2.1 sem\_open 🡪 deschide un semafor cu nume**

sem\_t \*sem\_open(const char \*name, int oflag);

sem\_t \*sem\_open(const char \*name, int oflag, mode\_t mode, unsigned int value);

🡪 daca mai multe procese apeleaza **sem\_open pe acelasi nume**, **vor primi referinte la acelasi semafor**

🡪 oflag - masca de biti ce contine valori privind modul de deschidere

**🡪 O\_CREAT** - daca semaforul nu exista deja, e creat

**🡪 O\_EXCL** - daca folosim O\_CREAT si semaforul exista deja, apelul de sistem va esua

🡪 daca folosim **O\_CREAT** trebuie sa specificam inca 2 parametri : **mode(permisiuni semafor) si value(numarul de permisiuni initiale ale semaforului)**

**8.2.2.2 sem\_close**

int sem\_close(sem\_t \*sem);

**🡪 inchide referinta spre un semafor; el va continua sa existe in sistem si va putea fi redeschid ulterior**

**8.2.2.3 sem\_unlink**

**🡪 sterge semaforul din sitem; daca mai exista referinte spre el, semaforul se va sterge doar cand ultima referinta se inchide, dar semaforul nu mai poate fi deschis intre timp**

**8.2.2.4 exemplu**

**1void write\_log(const char \*message)**

**2 {**

**3 sem\_t \*logSem = NULL; // pointeaza la o structura de tip semafor**

**4**

**5 logSem = sem\_open("/l08p1\_log\_semaphore", O\_CREAT, 0644, 1); // returneaza NULL in caz de esec**

**6 if(logSem == NULL) {**

**7 perror("Could not aquire the semaphore");**

**8 return;**

**9 }**

**10**

**11 ...**

**12**

**13 sem\_close(logSem);**

**14 }**

**8.2.3 operatii pe semafoare**

**#include <semaphore.h>**

**int sem\_wait(sem\_t \*sem);**

**int sem\_trywait(sem\_t \*sem);**

**int sem\_post(sem\_t \*sem);**

**8.2.3.1 sem\_wait**

**🡪 marcheaza intrarea in regiunea critica si corespunde primitivei wait**

🡪 **la apelul sem\_wait procesul/threadul curent poate sa isi continue executa obtinand permisiune de la semafor sau sa fie blocat si pus in coada de asteptare pana cand permsiunea ceruta va fi disponibila**

**8.2.3.2 sem\_post**

**🡪 apelul marcheaza iesirea din reg critica si returneaza permisiunea obt prin sem\_wait**

**8.2.3.3 sem\_trywait**

**🡪 pentru situatii in care vrem sa obtinem o permisiune de la un semafor dar in cazul in care acesta nu e disponibil sa nu fim blocati, ci sa putem continua executia**

* **daca permisiunea e disponibila are acelasi efect cu sem\_wait , atfel nu va bloca procesul sau thread-ul curent, ci va returna o valoare de eroare dif de 0**

**8.2.3.4 continuare exemplu**

**1 void write\_log(const char \*message)**

**2 { 3 sem\_t \*logSem = NULL;**

**4 int fd = -1;**

**5**

**6 logSem = sem\_open("/l08p1\_log\_semaphore", O\_CREAT, 0644, 1);**

**7 if(logSem == NULL) {**

**8 perror("Could not aquire the semaphore");**

**9 return;**

**10 }**

**11**

**12 //entering the critical region**

**13 sem\_wait(logSem);**

**14**

**15 fd = open("log.txt", O\_CREAT | O\_WRONLY | O\_APPEND, 0644);**

**16 if(fd < 0) {**

**17 perror("Could not open log for writing");**

**18 } else {**

**19 write(fd, message, strlen(message));**

**20 close(fd);**

**21 }**

**22**

**23 //exiting the critical region**

**24 sem\_post(logSem);**

**25**

**26 sem\_close(logSem);**

**27 }**

* observatii: reg critica e delimitata de sem\_wait si sem\_post; 2 thr/proc diferite care folosesc acest semafor nu pot fi simultan cu executia intre sem\_wait si sem\_post

**8.2.4 semafoare anonime**

**#include <semaphore.h>**

**int sem\_init(sem\_t \*sem, int pshared, unsigned int value);**

**int sem\_destroy(sem\_t \*sem);**

**8.2.4.1 sem\_init**

🡪 primeste ca parametru pointer la o vb de tip semafor **; initializarea unui semafor se face o singura data**

* **pshared** = trb sa fie 0 daca vrem sa folosim semaforul in procesul curent si 1 daca vrem sa poata fi transmis si altor procese
* **value =** val initiala a semaforului

**8.2.4.1 sem\_destroy**

**🡪 elibereaza resursele alocate acestuia**

**8.2.4.2 exemplu**

**1 typedef struct {**

**2 int value;**

**3 sem\_t \*logSem;**

**4 } TH\_STRUCT;**

**1 int main(void)**

**2 {**

**3 sem\_t logSem;**

**4 int i;**

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

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

**7**

**8 //initialize random number generator**

**9 srand(time(NULL));**

**10 //initialize the semaphore**

**11 if(sem\_init(&logSem, 0, 1) != 0) {**

**12 perror("Could not init the semaphore");**

**13 return -1;**

**14 }**

**15 //create the threads**

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

**17 params[i].value = 10 \* (i + 1);**

**18 params[i].logSem = &logSem;**

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

**20 }**

**21 //wait for the threads to finish**

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

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

**24 }**

**25 //destroy the semaphore**

**26 sem\_destroy(&logSem);**

**27**

**28 return 0;**

**29 }**

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

**2 {**

**3 TH\_STRUCT \*s = (TH\_STRUCT\*)p;**

**4 int i, count;**

**5 char message[20];**

**6**

**7 for(i=0; i<3; i++) {**

**8 usleep(1000 \* (rand() % 20));**

**9 count = s->value + i + 1;**

**10 snprintf(message, 20, "Log entry %d\n", count);**

**11 write\_log(message, s->logSem);**

**12 }**

**13 return NULL;**

**14 }**

**Intrebari:**

1. **ce se intampla daca apelam sem\_wait pe un sem cu 0 permisiuni**

* **procesul/threadul ce a apelat wait e blocat pana cand semaforul primeste permisiuni noi**

1. **in ce conditii alegem sa folosim un sem cu nume fata de unul anonim?**

* **Folosim un semafor cu nume cand avem de sincronizat thread-uri din procese diferite**

1. **Care e diferenta dintre sem\_close si sem\_unlink?**

* **Sem close inchide referinta la semafor ( si ulterior poate fi redeschisa) in timp ce sem\_unlink sterge semaforul(daca semaforul are mai multe referinte deschise spre el, atunci el se va sterge cand ultima referinta se inchide dar sem nu mai poate fi redeschis intre timp)**

1. **Pentru a sincroniza mai multe thread-uri se folosesc un sem ce apeleaza sem\_wait la inceputul functiei si sem\_post la final.**

**De ce e in general gresita aceasta abordare?**

* **?? nu e recomandata parasirea functiei cu return din zona critica deoarece nu am mai ajunge sa apelam sem\_post si semaforul ar ramane blocat**

**5. Codul de mai jos se apeleaz˘a pe mai multe thread-uri pentru a construi un vector global v**

**cu numere prime. Explicat, i ce se ˆıntˆampl˘a dac˘a un thread este preemptat ˆıntre liniile 3 s, i 4**

**s, i se execut˘a codul unui alt thread? Cum at, i folosi un semafor pentru a evita cursa critic˘a?**

**1 for(i=start; i<=end; i++) {**

**2 if(is\_prime(i)) {**

**3 size++;**

**4v[size] = i;**

* 1. **}**
  2. **}**

**Daca semaforul e intrerupt intre liniile 3 si 4 si se executa codul unui alt thread la reintoarecerea in semaforul anterior intrerupt size va avea alta valoarea si I se va pune la index gresit**

**1 for(i=start; i<=end; i++) {**

**sem\_wait();**

**2 if(is\_prime(i)) {**

**3 size++;**

**4v[size] = i;**

**5 }**

**Sem\_close();**

**6 }**