Sincronizzazione 6 Semafori Unix

martedì 25 novembre 2025 13:03

Andiamoli a vedere in dettaglio : **è una struttura dati gestita dal sistema operativo che include un valore numerico**. Ha 3 operazioni associate in modo atomico:

1. **Inizializzazione dell'interno non negativo**
2. **Wait**
   a. Decrementa di un'unità l'intero del semaforo
   b. Se l'intero è negativo, in seguito a decremento, si entro nello stato di wait
   c. Servizio bloccante
   d. Prende il controllo del semaforo
3. **Signal**
   a. Incrementa il valore di un'unità e libera dall'attesa un processo che è in wait (andando n blocco)
   b. Rilascia il gettone

Se binari il valore assume o 0 oppure 1. Quindi vengono visti come distributori di token , il quale numero intero è il numero di token disponibile. In generale o meglio l'idea di sottofondo :

```
type semaphore = record //struct like
                    value: int;
                    L: list of processes;// or threads
                end;

procedue Wait(var s: semaphore):
    if (s.value – 1) < 0 {
        add current process to s.L;
        block current process;
    }else{s.value--;}

procedure Signal(var s: semaphore);
    s.value = s.value + 1;
    if s.L not empty{
        delete a process P from s.L;
        unblock P;
        s.value = s.value – 1;
    }
```

Atomically executed by the kernel software

Atomically executed by the kernel software

Quindi vediamo ora di nuovo il problema del produttore/consumatore usando i semafori:

```
SHARED: item buffer[N]; semaphore S = 1; int counter = 0;

PRODUTTORE
PRIVATE int in = 0; item X;
Repeat
    <produce X>
retry:
    wait(S);
    if counter = N {
        signal(S);
        goto retry;
    }
    else{
        buffer[in] := X;
        in := (in+1)mod(N)
        counter := counter + 1;
        signal(S);
    }
until false

CONSUMATORE
PRIVATE int out = 0; item Y;
Repeat
    wait(S);
    if counter = 0 {
        signal(S);
    }
    else{
        Y := buffer[out];
        out := (out+1)mod(N)
        counter := counter - 1;
        signal(S);
        <consume Y>
    }
until false
```

In generale : **un semaforo è un' array di distributore di gettoni** .Vediamolo ora in UNIX:

1. **Creazione**
   a.

int semget(key_t key, int size, int flag)	
Descrizione	invoca la creazione di unsemaforo
Parametri	1) key: chiave per identificare il semaforo in maniera univoca nel sistema (IPC_PRIVATE e' un caso speciale) 2) size: numero di elementi del semaforo 3) flag: specifica della modalita' di creazione (IPC_CREAT, IPC_EXCL, definiti negli header file sys/ipc.h e sys/shm.h) e dei permessi di accesso
Descrizione	identificatore numerico per il semaforo in caso di successo (descrittore), -1 in caso di fallimento

NOTA
Il descrittore indicizza questa volta una struttura unica valida per qualsiasi processo

2. **Comandi su semaforo**

a.

int semctl(int ds_sem, int sem_num, int cmd, union semun arg)	
Descrizione	invoca l'esecuzione di un comando su un semaforo
Parametri	1) ds_sem: descrittore del semaforo su cui si vuole operare 2) sem_num: indice dell'elemento del semaforo su cui si vuole operare 3) cmd: specifica del comando da eseguire (IPC_RMID, IPC_STAT, IPC_SET, GETALL, SETALL, GETVAL, SETVAL) 4) arg: puntatore al buffer con eventuali parametri per il comando
Ritorno	-1 in caso di fallimento

```
union semuu {
        int     val;            /* usato se cmd == SETVAL  */
        struct semid_ds *buf    /* usato per IPC_STAT e IPC_SET */
        ushort *array;          /* usato se cmd == GETALL o SETALL */
  };
```

b. La union permette di usare un solo parametro alla volta (di taglia più grande tra i campi)

3. **Operazioni su semaforo**

int semop(int ds_sem, struct sembuf oper[], int number)	
Descrizione	invoca l'esecuzione di un comando su una coda di messaggi
Parametri	1) ds_sem: descrittore del semaforo su cui si vuole operare 2) oper: indirizzo dell'array contenente la specifica delle operazioni da eseguire 3) number: numero di argomenti validi nell'array puntato da oper
Ritorno	-1 in caso di fallimento

a.

```
struct sembuf {
        ushort sem_num;
        short  sem_op;  /* 0=sincronzzazione sullo 0 - n=incremento
                           di n - -n=decremento di n */
        short  sem_flg; /* IPC_NOWAIT - SEM_UNDO */
    };
```

Un decremento di N su un semaforo dal valore minore di N provoca blocco del processo chiamante a meno della specifica di IPC_NOWAIT .

SEM_UNDO revoca l'operazione in caso di exit del processo

b. **Implementa sia wait che signal**

c. Se sem_undo e poi fork , il figlio non eredita l'undo

Vediamo un esempio produttore/consumatore con i semafori -> SYSTEM V:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/sem.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <unistd.h>


#define SIZE (100)
#define END (1000000)

#define AUDIT if(0)

long *v;
int sem_free_slot, sem_available_item;
```

```c
void * producer(void){

        struct sembuf oper;
        long data = 0;
        long my_index = 0;
        printf("ready to produce\n");

retry:
        oper.sem_num = my_index;
        oper.sem_op = -1;
        oper.sem_flg = 0;

        semop(sem_free_slot,&oper,1);
        v[my_index] = data;

        oper.sem_num = my_index;
        oper.sem_op = 1;
        oper.sem_flg = 0;

        semop(sem_available_item,&oper,1);
        my_index = (my_index+1)%SIZE;
        data++;
        goto retry;
}
```

```c
void * consumer(void){

        long data = 0;
        long my_index = 0;
        long value;
        struct sembuf oper;
        printf("ready to consume\n");

retry:
        oper.sem_num = my_index;
        oper.sem_op = -1;
        oper.sem_flg = 0;

        semop(sem_available_item,&oper,1);
        value = v[my_index];

        AUDIT
        printf("consumer got value %ld\n",value);

        if(value != data){
                printf("consumer: synch protocol broken at expected value: %ld - real is %ld!!\n",data+1,value);
                exit(EXIT_FAILURE);
        };

        if (value == END){
                printf("ending condition met - last read value is %ld\n",value);
                exit(0);
        }

        oper.sem_num = my_index;
        oper.sem_op = 1;
        oper.sem_flg = 0;

        semop(sem_free_slot,&oper,1);
        my_index = (my_index+1)%SIZE;
        data++;
        goto retry;
}
```

```c
int main(int argc, char** argv){

        int prod, cons;
        int i;
        key_t key = IPC_PRIVATE;

        sem_free_slot = semget(key,SIZE,IPC_CREAT|0666);
        if(sem_free_slot == -1){
                printf("semget error\n");
                exit(EXIT_FAILURE);
        }
        for (i=0; i<SIZE; i++){

                if(semctl(sem_free_slot,i,SETVAL,1)==-1){
                        printf("semctl error\n");
                        exit(EXIT_FAILURE);
                }
        }
        sem_available_item = semget(key,SIZE,IPC_CREAT|0666);
        if(sem_available_item == -1){
                printf("semget error\n");
                exit(EXIT_FAILURE);
        }
        for (i=0; i<SIZE; i++){
                if(semctl(sem_available_item,i,SETVAL,0)==-1){
                        printf("semctl error\n");
                        exit(EXIT_FAILURE);
                }
        }
        v = (long*)mmap(NULL,SIZE*sizeof(long),PROT_READ|PROT_WRITE,MAP_ANONYMOUS|MAP_SHARED,0,0);
        if (v == NULL){
                printf("mmap error\n");
                exit(EXIT_FAILURE);
        }
        prod = fork();
        if (prod == -1){
                printf("fork on producer error\n");
                exit(EXIT_FAILURE);
        }
        if (prod == 0){
                producer();
        }

        cons = fork();
        if (cons == -1){
                printf("fork on consumer error\n");
                exit(EXIT_FAILURE);
        }
        if (cons == 0){
                consumer();
        }
        wait(NULL);
        semctl(sem_free_slot,IPC_RMID,0);
        semctl(sem_available_item,IPC_RMID,0);
        exit(0);
}
```

Esecuzione corretta. Con **ipcstat** posso vedere i semafori presenti. I semafori devono essere rimossi esplicitamente. Vediamo ora delle varianti (POSIX):

- ✓ `sem_t sem_name;`
- ✓ `sem_t *sem_open(const char *name, int oflag)`
- ✓ `int sem_init(sem_t *sem, int pshared, unsigned int value);`
- ✓ `sem_wait(&sem_name);`
- ✓ `sem_post(&sem_name);`
- ✓ `sem_getvalue(sem_t *sem, int *valp);`
- ✓ `sem_unlink(const char *name);`

Implementazione basata su pseudo files

Puo' inizializzare anche semafori unnamed

Nota : per i semafori o secondo nomenclatura SYSTEM V oppure POSIX . Vediamo esempio

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/sem.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <fcntl.h>
#include <unistd.h>



#define SIZE (1000)
#define END (10000000)

#define AUDIT if(0)

long *v;

sem_t *sem_free_slot[SIZE], *sem_available_item[SIZE];
```

```c
void * producer(void){

        long data = 0;
        long my_index = 0;

        printf("ready to produce\n");

retry:

        sem_wait(sem_free_slot[my_index]);

        v[my_index] = data;

        sem_post(sem_available_item[my_index]);

        my_index = (my_index+1)%SIZE;
        data++;

        goto retry;


}
```

```c
void * consumer(void){

        long data = 0;
        long my_index = 0;
        long value;

        printf("ready to consume\n");

retry:
        sem_wait(sem_available_item[my_index]);
        value = v[my_index];

        AUDIT
        printf("consumer got value %ld\n",value);

        if(value != data){
                printf("consumer: synch protocol broken at expected value: %ld
                 - real is %ld!!\n",data,value);
                exit(-1);
        };

        if (value == END){
                printf("ending condition met - last read value is %ld\n",value);
                exit(0);
        }

        sem_post(sem_free_slot[my_index]);

        my_index = (my_index+1)%SIZE;

        data++;

        goto retry;

}
```

```c
int main(int argc, char** argv){

        int prod, cons;
        int i;
        char buff[128];

        for (i=0; i<SIZE; i++){
                sprintf(buff,"slot%d\0",i);
                sem_unlink(buff);
                sem_free_slot[i] = sem_open(buff,O_CREAT,0666,1);
                if (sem_free_slot[i] == NULL){
                        printf("cannot create free slot semaphore at iteration %d\n",i);
                        exit(EXIT_FAILURE);
                }
        //sem_unlink(buff);
        }

        for (i=0; i<SIZE; i++){
                sprintf(buff,"item%d\0",i);
                sem_unlink(buff);
                sem_available_item[i] = sem_open(buff,O_CREAT,0666,0);
                if (sem_available_item[i] == NULL){
                        printf("cannot create available slot semaphore at iteration %d\n",i);
                        exit(EXIT_FAILURE);
                }
        //      sem_unlink(buff);
        }

        v = (long*)mmap(NULL,SIZE*sizeof(long),PROT_READ|PROT_WRITE,MAP_ANONYMOUS|MAP_SHARED,0,0);
        if (v == NULL){
                printf("mmap error\n");
                exit(EXIT_FAILURE);
        }

        prod = fork();
        if (prod == -1){
                printf("fork on producer error\n");
                exit(EXIT_FAILURE);
        }

        if (prod == 0){
                producer();
        }

        cons = fork();
        if (cons == -1){
                printf("fork on consumer error\n");
                exit(EXIT_FAILURE);
        }

        if (cons == 0){
                consumer();
        }
        wait(NULL);
        exit(0);
}
```

Se invece uso i semafori senza nome , utilizzo la mmap shared.