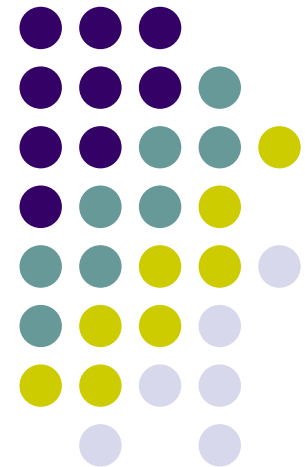


Einführung in die Systemprogrammierung

4

Interprozesskommunikation

Semaphoren



Semaphoren

- Nicht negative Zählvariable (Datentyp unsigned short)
- Wird dekrementiert bei Eintritt in kritischen Abschnitt
- Wird inkrementiert bei Verlassen des kritischen Abschn.
- Werden benutzt zur Synchronisation, wenn mehrere Prozesse auf gemeinsame Ressource zugreifen
- Schritte bei Zugriff auf gemeinsame Ressource:
 - Überprüfen des Semaphors
 - Ist $s > 0$: Prozess dekrementiert s und tritt in kritischen Abschnitt ein (atomare Operation)
 - Ist $s == 0$: Prozess wartet bis $s > 0$ wird
 - Verlassen des kritischen Abschnitts
 - Prozess inkrementiert s

Eigenschaften von System V-Semaphoren

- Semaphoren sind als Semaphorenmengen implementiert. dies erlaubt detaillierte Aufteilung von kritischen Bereichen
- Kreieren und Initialisierung von Semaphoren sind zwei unabhängige (nicht atomare) Funktionen
- Semaphore existieren immer weiter, auch wenn kein Prozess darauf zugreift. => Semaphoren müssen wieder freigegeben werden.
- Limits von Semaphormengen
 - SEMVMX: maximaler Wert eines Semaphors (typisch: 32767)
 - SEMMNI: maximale Anzahl von Semaphormengen (typisch: 128)
 - SEMMNS: maximale Anzahl von Semaphoren (typisch: 3000)
 - SEMMSL: max. Anzahl Semaphoren/Semaphorenmenge (typisch 250)

semid_ds – Status eines Semaphors

- Zu jedem Semaphor existiert eine semid_ds-Struktur:

```
struct semid_ds{
    struct ipc_perm sem_perm; // IDs und Zugriffsrechte
    struct sem *sem_base      // Adr. des 1. Semaphors
    ushort sem_nsems;        //Anz. Semaphoren d. Menge
    time_t sem_otime;        //Zeit d. letzten semop-Aufrufs
    time_t sem_ctime;        //Zeit d. letzten Änderung
}
```

- Die Struktur sem enthält:

```
struct sem{
    ushort semval;          // Semaphorwert (>=0)
    pid_t sempid;          // PID des zuletzt zugr. Prozesses
    ushort semncnt;        // Anz. wartender Proz. bis semval>0
    ushort semzcnt;        // Anz. wartender Proz. bis semval==0
}
```

semget – Öffnen oder Kreieren einer Semaphormenge

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget (key_t schlüssel, int nsem, int flag);
```

- Rückgabewerte:
 - Bei Erfolg: Semaphor-ID (Kennung)
 - Bei Fehler: -1
- *nsem*: Anzahl der Semaphoren in der Menge.
 - Muss beim Erzeugen angegeben werden
 - Beim Öffnen existierender Semaphore: 0
- *flag*:
 - IPC_CREAT: zum kreieren eines neuen Objekts mit dem angegebenen Schlüssel

semctl –Setzen einer Semaphorvariablen

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl (int kennung, int semnum, int kdo, union semun
arg);
```

- *semnum*: Index des Semaphors (0 .. *nsem*-1)
- *kdo*: legt die durchzuführende Aktion fest:
 - SETVAL: Setzen einer Semaphorvariablen
- *arg*:
 - Für SETVAL: (int)Wert (zu setzender Wert)

semctl –Löschen einer Semaphormenge

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl (int kennung, int semnum, int kdo, union semun
arg);
```

- *kdo*: legt die durchzuführende Aktion fest:
 - IPC_RMID: Löschen der Semaphormenge mit allen ihren Daten
- *arg*: wird benötigt für IPC_STAT und IPC_SET. Für IPC_RMID ist hier NULL einzusetzen
- **Hinweis: Semaphormengen werden nicht automatisch mit dem Beenden eines Prozesses gelöscht, sondern müssen explizit gelöscht werden!**

semop – Durchführung von Operationen auf Semaphormenge

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop (int kennung, struct sembuf semoparray[], size_t
nops);
```

- semop führt eine Reihe von Operationen (übergeben in Array) auf Semaphormenge aus.
- semop gibt zurück:
 - Bei Erfolg: 0
 - Bei Fehler: -1
- semid: Semaphor-ID

semop – Durchführung von Operationen auf Semaphormenge

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop (int kennung, struct sembuf semoparray[], size_t
nops);
```

- *semoparray*: Adresse eines Arrays von Semaphoroperationen. Elemente des Arrays sind vom Typ `struct sembuf`:

```
struct sembuf{
    ushort sem_num; // Nr. de. Semaphorvar. in Menge
                    // ((0 .. nsems-1)
    short sem_op; // Operation
    short sem_flg; // IPC_NOWAIT, SEM_UNDO
}
```

Operation sem_op

- sem_op >0: Ressource freigeben
 - Wert von sem_op wird auf den Wert der Semaphorvariablen (semval) addiert
- sem_op <0: Ressource anfordern
 - semval \geq |sem_op|:
 - sem_op wird von semval subtrahiert
 - semval < |sem_op|:
 - IPC_NOWAIT gesetzt: semop-Aufruf beendet mit einem Fehler
 - nicht gesetzt: semncnt wird inkrementiert und aufrufender Prozess wird suspendiert bis semval \geq |sem_op| wird. Dann wird semncnt wieder dekrementiert und sem_op von semval subtrahiert

Implementierungsbeispiele

```
/* Eine gemeinsame Headerdatei sema.h*/  
#ifndef SEMA  
#define SEMA  
  
/* Vereinbarer Schlüssel für gemeinsames Objekt */  
#define SEM_MYKEY    10003  
  
#endif
```

Beispiel

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include "sema.h";

static int sem_id;
static struct sembuf semaphor;
int main (void)
{
    // Erzeugen einer Semaphormenge mit 1 Element
    if ((sem_id= semget(SEM_MYKEY,1,IPC_CREAT)) == -1)
        // Fehlerbehandlung ;
    // Initialisieren des Semaphors
    if (semctl(sem_id,0,SETVAL,(int)1) == -1)
        //Fehlerbehandlung;
```

Beispiel

```
// Eintritt in kritischen Abschnitt
semaphor.sem_op=-1;
semaphor.sem_flg = SEM_UNDO;
if (semop(sem_id,&semaphor,1) ==-1)
    //Fehlerbehandlung;

// Kritischer Abschnitt
// ...
// Verlassen des kritischen Abschnitts:
semaphor.sem_op=1;
semaphor.sem_flg = SEM_UNDO;
if (semop(sem_id,&semaphor,1) ==-1)
    //Fehlerbehandlung ;

// Weiterer Code

exit(0);
}
```