Inhalt

Interprozesskommunikation II

- benannte Semaphore
- shared memory
- message queues

Benannte Semaphore:

- sind systemweit verfügbar und können somit zur Prozesssynchronisation verwendet werden.
- Unix kennt historisch verschiedene benannte Semaphore.

- POSIX definiert zwei Varianten:
  - POSIX:SEM named semaphores
    - etwa wie die bereits behandelten POSIX:SEM unnamed semaphores
  - POSIX:XSI named semaphore sets
    - es werden immer Semaphor-Gruppen angelegt
    - es gibt atomare Operationen für Gruppen

POSIX:SEM named semaphores

- Wie alle named semaphores sind auch diese **persistent**.
  ➔ der Zählwert existiert auch nach einem möglichen Programmabbruch weiter!
- Die Erzeugung erfolgt nicht mit `sem_init(...)` sondern mit:

```
sem_t* sem_open(const char* name, int flags, ...);
```

  - liefert einen Zeiger auf den Semaphor
  - der **`name`**
    - dient zur prozessglobalen Identifikation
    - hat die Form eines Dateinamens und die Rechte einer Datei, ist aber nicht im Filesystem als Datei aufgeführt
    - nur names, die mit / beginnen werden als identisch erkannt

- wird für **flags** der Wert **O_CREAT** angegeben:
  - wird der Semaphor erzeugt, falls er noch nicht existiert **und** es gibt zwei weitere Parameter:

```
sem_t* sem_open(..., mode_t mode, unsigned int value);
```

- **mode** definiert die Zugriffsrechte, z.B. 0644
- **value** definiert den Initialwert des Zählers

Achtung: Falls der Semaphor bereits existiert, werden **mode** und **value** ignoriert.

– Wie bei den unnamed semaphors gibt es:
- **sem_wait(...)** → P-Operation
- **sem_post(...)** → V-Operation

– Schließen mit **sem_close(...)** → existiert weiter

– Entfernen mit **sem_unlink(...)** → existiert nicht weiter

POSIX:XSI IPC definiert:

*: hier nicht weiter behandelt

- semaphore sets*, shared memory u. message queues
- Diese drei Kommunikationsmittel verwenden:
  - ein identisches Verfahren zur systemweiten Identifikation
  - gleichartige Funktionen, z.B. **`semget`**, **`shmget`**, **`msgget`**
- Um den Zustand dieser Betriebsmittel anzuzeigen oder zu löschen gibt es shell-commands:
  - **`ipcs`** → Informationen über sem, shm oder msg anzeigen
  - **`ipcrm`** → sem, shm oder msg entfernen
- Systemweite Identifikation:
  - Die **`xxxget(...)`** – Funktionen erfordern die Angabe eines **`keys`**.
  - Die **`xxxget(...)`** – Funktionen liefern eine **`int`**-ID zur prozessinternen Verwendung.

- Als **key** kann verwendet werden:
  - **IPC_PRIVATE** → prozessintern, anderen Prozessen nicht bekannt
  - ein Wert > 0
- Um systemweit möglichst eindeutige Werte zu erhalten, kann **ftok()** verwendet werden:

```
key_t ftok(const char *path, int id);
```

  - **path** ist der Pfad einer existierenden, zugreifbaren Datei
  - **id** ist ein Wert > 0, nur die unteren 8 Bit sind signifikant

→ **ftok()** liefert für einen identischen Pfad und eine identische **id** immer denselben **key**.

→ Mit einem Pfad und verschiedenen **id**s können mehrere keys für verschiedene resources erzeugt werden.

POSIX:XSI shared memory

1. Mit **`shmget(...)`** und **`ftok(...)`** eine **`id`** anfordern
   - geforderte Größe wird in Byte angeben (**`sizeof(...)`**)
2. Mit „attach“ **`shmat(...)`** ein Speichersegment im eigenen Adressraum abbilden:

```
void* shmat(int shmid, void* shmaddr, int shmflg);
```

   - liefert **`void*`** Zeiger ➔ Struktur beliebig
   - **`shmid`** ist die von **`shmget`** gelieferte **`id`**
   - mit **`shmaddr`** kann der Bereich an einer bestimmten Adresse angefordert werden. Im Normalfall wird NULL angegeben ➔ das System definiert die Adresse.
   - mit **`shmflg`** wird read/read-write access eingestellt
3. Mit „control“ **`shmctl(...)`** kann das shared memory z.B. entfernt oder kopiert werden.

Beispiel zu **`shm`** siehe Vorgabe zur Übung08: **`showPhilos.c`**

POSIX:XSI message queue

- message queues realisieren im Gegensatz zu pipes eine nachrichtenbasierte Kommunikation.
- Die Nachrichten können beliebig strukturiert sein.
- Mehrere Prozesse können senden und empfangen.
- Nachrichten werden mit einer Nummer **`mtype`** (Typ **`long`**) typisiert gesendet.
  Beim Empfang können gezielt bestimmte Nachrichten durch Angabe eines Empfangstyps **`msgtyp`** der Schlange entnommen werden:
  - **`msgtyp`** = 0 → die erste Nachricht (FIFO)
  - **`msgtyp`** > 0 → die erste Nachricht mit **`mtype`** == **`msgtyp`**
  - **`msgtyp`** < 0 → die erste Nachricht mit **`mtype`** <= **`|msgtyp|`**

Vorgehensweise **message queues**:

1. Die Datenstruktur der Nachricht definieren.
   - Das erste Element der Struktur muss der o.g. Nachrichtentyp **`mtype`** vom Typ **`long`** mit einem Wert **`> 0`** sein.
   - Die weitere Struktur ist beliebig, z.B.:

```
typedef struct
{
    long mtype;
    int i;
    double x;
} messageTy;
```

beliebig, auch structs oder arrays

2. Mit **`msgget(...)`** und **`ftok(...)`** eine **`id`** anfordern.
3. Mit **`msgsnd(...)`** die Nachricht senden. Dabei wird die Nachricht in einen Systembereich kopiert:
   - die **`id`** angeben
   - die Nachrichtenlänge **`msgsz`** ohne den long **`mtype`** angeben
   - ein **`msgflag`** angeben mit **`0`** / **`IPC_NOWAIT`** → blockierendes / nicht blockierendes Senden

4. Mit **`msgrcv(...)`** die Nachricht empfangen. Nun wird die Nachricht in einen Puffer des Prozesses kopiert. Die Parameter sind:
   - die **`id`** der Queue
   - der Zeiger **`msgp`** auf den Ziel-Puffer
   - die max. Größe der Kopie, ohne den long **`mtype`**
   - den **`long`**-Empfangstyp **`msgtyp`**
   - ein **`msgflag`** mit **`0`** / **`IPC_NOWAIT`** → blockierendes / nicht blockierendes Empfangen
5. Mit „control“ **`msgctl(...)`** kann der Zustand der queue abgefragt oder auch die queue entfernt werden.

Beispiel zu **`msg`** siehe Vorgabe zur Übung08: **`showPhilos.c`**