SYNKRONISERING

SYNKRONISERING

- Det är inte tillräckligt att processerna kan utbyta information ... man måste också kunna synkronisera processerna så att den information man utbyter är korrekt.
- Mutex – fungerar bra för trådar
- Semaforer – mellan processer eller trådar
  - Lite mer funktionalitet än mutex

SEMAFORER

- Ett sätt att utbyta information är att använda sig av semaforer. Dessa är en speciellt sorts räknare som man **endast** kan modifiera med hjälp av två funktioner ofta kallade: wait (take) och signal (eller release).
  - Ibland ser man också namnen P och V för dessa operationer då de var dessa namn de fick då de uppfanns av Dijkstra. P ="Probeer" försöka (att minska), V= "Verhoog" öka.

SEMAFORER – WAIT

- Wait minskar värdet på en semafor med ett om värdet på den är större än 0.
- Om värdet var mindre än 0 så kommer processen att bli "suspendad" tills värdet blir större än 0 (genom att någon annan process gör signal).

SEMAFORER – SIGNAL

- Ökar värdet på en semafor med ett
  - om då värdet blir större än 0 så kommer en process (om det finns någon) som gjort wait på semaforen att väckas.
  - om den redan är större än 0 så kommer värdet bara att öka med 1.

SEMAFORER – WAIT/SIGNAL

- Atomiska operationer
  - Både wait och signal är odelbara operationer, dvs. det kan inte bli ett processbyte medan de exekveras.
- Ömsesidig uteslutning
  - Med hjälp av semaforer kan man då sätta upp ett "staket" runt en resurs och sedan begränsa hur många processer som kan vara innanför staketet samtidigt.

```
Process1:
semaphore s
initiera semaforen
...
wait(s)
   access granted
signal(s)
...
```

```
Process2
semaphore s
...
wait(s)
   access granted
signal(s)
...
```

POSIX SEMAPHORES

- `#include <semaphore.h>;`
- Länka med
  `-lpthread`
- Låsa en semafor
```
int sem_wait(sem_t *sem);
int sem_trywait(sem_t *);
int sem_timedwait(sem_t *, struct timespec *);
```
- Låsa upp en semafor
```
int sem_post(sem_t *sem);
```

POSIX SEMAPHORES

- Unnamed semaphores (gemensamt minne)
  - Deklarera en semafor
    `sem_t sem;`
  - Initiera en semafor
```
int sem_init(sem_t *sem, int pshared,
             unsigned int value);
```
    `pshared == 0` innebär **inom process (trådar)**, annars mellan processer och då måste semaforen finnas i ett gemensamt minne
  - Ta bort en semafor
    `int sem_destroy(sem_t *sem);`
  - Viktigt att ta bort semaforer eftersom de annars fortsätter ta upp resurser.

POSIX SEMAPHORES

- Named semaphores
  - Deklarera en semafor
    ```
    sem_t *sem;
    ```
  - Öppna/initiera en semafor
    ```
    sem_t *sem_open(char *, int, ...);
    ```
  - Stänga en semafor
    ```
    int sem_close(sem_t *);
    ```
  - Ta bort en semafor
    ```
    int sem_unlink(char *);
    ```
  - Viktigt att ta bort semaforer eftersom de annars fortsätter ta upp resurser.

SYSTEM V SEMAFORER, XSI IPC

- Lite bökiga att vända, vi måste tex skapa en mängd med semaforer
- Skapa en mängd semaforer mha
  ```
  int semget(key_t key, int nsems, int flag);
  ```
- Initiera den mha
  ```
  int semctl(int semid, int semnum, int cmd, ...);
  ```
- Använd den mha
  ```
  int semop(int semid, struct sembuf *array,
            size_t nops);
  ```
- Obs att alla processer måste få veta nyckeln (eller `semid`) på något sätt.

SYSTEM V IPC-ID

- Olika alternativ
  - Bestäm en nyckel och skapa ett id för denna nyckel i båda processerna (nyckeln kan t.ex. lagras i en .h-fil)
  - Låt systemet välja ett då du skapar resursen och publicera detta på något sätt så att andra processer kan komma åt informationen
  - `ftok`
    - Tar fram ett id utifrån en sökväg till en fil och ett id

FTOK

- Med hjälp av funktionen
  ```
  key_t ftok(const char *path, int id)
  ```
  Kan man generera en nyckel som kan användas till delat minne, semaforer och meddelandeköer
- `path` måste referera till en existerande fil då funktionen använder sig av bl.a. inodsnumret för filen för att skapa nyckeln

MER KOMMUNIKATION OCH SYNKNING

MESSAGE QUEUES

- Kan ses som en länkad lista av meddelanden som lagras i kärnan.
- Varje meddelande har en meddelandetyp. Flera olika processer kan kommunicera via samma meddelandekö. Genom att använda olika typ på meddelandena så kan man se till så att rätt process får rätt meddelande.
- Används inte så ofta pga. att de inte är så enkla att använda och inte ger några effektivitets-vinster mot de andra formerna av kommunikation som vi tittat på.

DELAT MINNE MELLAN PROCESSER

- Flera processer kan dela på ett minnesutrymme. Detta är ett väldigt snabbt sätt att kommunicera på då inget data måste kopieras mellan processerna.
- En identifierare till ett delat minnesutrymme kan fås genom funktionen

```
int shmget(key_t key, int size, int shmflg);
```

- För att andra processer ska komma åt minnet måste de ha tillgång till identifieraren samt har rättigheter att komma åt minnet

DELAT MINNE MELLAN PROCESSER

- Minnet kan sedan göras tillgängligt för processen mha funktionen

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

- respektive avmappas mha

```
int shmdt(void *shmaddr);
```

ipcs OCH ipcrm

- Städa upp ipc-resurser efter er!

  `ipcs`

  – Kommando för att visa status för ipc-användning på en dator. Pipa ev. resultatet till grep och leta efter ert användarnamn (för att inte behöva titta igenom så långa listor)

  `ipcrm`

  – Kommando för att ta bort en meddelandekö, semaformängd eller delat minnessegment

RECORD LOCKING

- Låsa en fil (eller del av fil) för access.
  – Görs mha `fcntl`-funktionen
  – Huvudsyfte att se till så att filer inte modifieras/används samtidigt som annat program använder sig av den.
  – Kan även användas för synkronisering, man kan då använda en tom fil och sätta ett lås på första byten (som inte måste existera)

RECORD LOCKING

- Fördelar mot semaforerna
  – Enklare att använda
  – Systemet tar hand om kvarvarande lås då en process avslutas (låsen är hopkopplade med en fil och en process; stängs filen eller processen avslutas släpps låset)
  – Read/write locks
- Se 14.3 i boken för detaljer

ANDRA SÄTT ATT SYNKRONISERA PROCESSER

- Pipor/FIFO
  – Vi kan skriva en byte, respektive läsa en byte i andra änden av en pipa/FIFO för att synkronisera processer
- Signaler kan också användas till synkronisering.
  – Den som ska vänta kör `sigsuspend` eller `sigwait`. och den som ska meddela att det är ok att fortsätta skickar en signal med `kill`