J 8. Übung

J.1 Überblick

- Besprechung 6. Aufgabe (timed)
  (die 5. Aufgabe wird erst in der nächsten Woche besprochen)
- select
- Shared Memory
- Semaphore

J.2 select

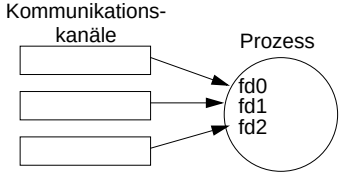
- Prototyp

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *errorfds, struct timeval *timeout);

void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

- blockiert so lange, bis einer der Filedeskriptoren bereit ist oder bis die timeout-Zeit abgelaufen ist
- timeout kann auch **`NULL`** sein (endlos warten)

J.2 select

- Prozesse die mit Pipes und Sockets arbeiten, müssen oft von verschiedenen Filedeskriptoren lesen bzw. schreiben

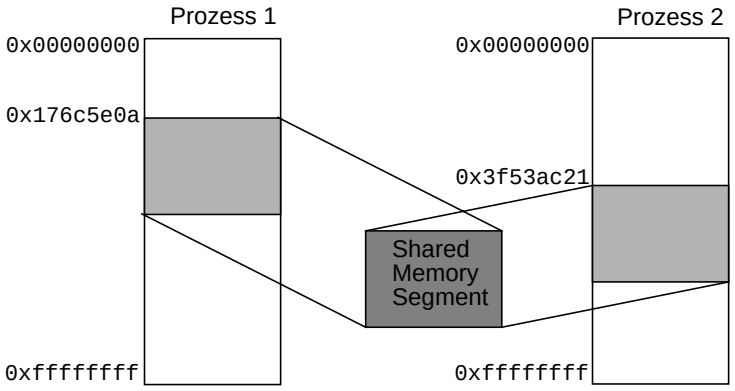


- diese Lese- bzw. Schreiboperationen können blockieren
- Lösungsmöglichkeiten
  - Angabe von **`O_NOBLOCK`** bei open; Problem: aktives Warten
  - fork eines Prozesses pro Filedeskriptor; Problem: teuer
  - Verwenden von **`select()`**

J.2 select

- select markiert FD als lesbar aber read gibt 0 zurück?
  - andere Seite hat die Verbindung geschlossen

J.3 Shared Memory



1 Anlegen des Segments: shmget

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
 ...
key_t  key;  /* Schlüssel */
int  shmflg; /* Flags */
int  shmid;  /* ID des Speichersegments */
int  size;   /* Größe des Speichersegments */
  ...
key = ftok("/etc/passwd", 42);
if (key == (key_t)-1) { /* Fehlerbehandlung */ }
size = 4096;
shmflg = 0666 | IPC_CREAT; /* Lesen/Schreiben für alle */
if ((shmid = shmget (key, size, shmflg)) == -1) {
   /* Fehlerbehandlung */
}

printf("shmget: id=%d\n", shmid);
```


1 shmget

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

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

- Schlüssel=**IPC_PRIVATE**: Segment ist prozesslokal
- Flags enthält **IPC_CREAT**: Segment wird erzeugt, falls es noch nicht existiert
- **IPC_CREAT | IPC_EXCL**: Segment wird neu erzeugt, liefert Fehler (**errno=EEXIST**), falls Segment schon existiert

2 Mappen des Segments in Datensegment des Prozesses (attach): shmat

```
#include <sys/types.h>
#include <sys/shm.h>

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

- shmaddr=0: System wählt Adresse aus
- shmflg:
  - SHM_RDONLY: Segment nur lesbar attached
- Rückgabewert: Startadresse des Segments

3 Freigeben des Segments (detach): shmdt

```
#include <sys/types.h>
#include <sys/shm.h>

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

- Rückgabewert:
  - 0 im Erfolgsfall, -1 im Fehlerfall

4 Kontrolle des Segments: shmctl

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

- **SHM_LOCK**: Sperren des Speichersegments im Speicher (Superuser)
- **SHM_UNLOCK**: Freigeben (Superuser)
- **IPC_STAT**: Status Informationen abfragen (Leserecht erforderlich)
- **IPC_SET**: Benutzer/Gruppenkennzeichnung, Zugriffsrechte des Segments setzen (nur erlaubt für Besitzer, Erzeuger oder Superuser)
- **IPC_RMID**: Löschen des Segments (nur erlaubt für Besitzer, Erzeuger oder Superuser)
  - Falls Prozesse das Segment noch attached haben, wird das Segment erst beim letzten Detach freigegeben. Neue Attachments sind nicht mehr erlaubt.

5 Shared Memory und Zeiger

- Segmente liegen in den einzelnen Prozessen möglicherweise an verschiedenen virtuellen Adressen (evtl. auch mehrfach innerhalb eines Prozesses)
- Daten dürfen in diesem Fall nicht absolut verzeigert werden
- mögliche Lösungen:
  - ➤ Zeiger relativ zum Segmentanfang
  - ➤ Struktur für shm-Aufbau definieren und Struktur-Zeiger auf Segmentanfang legen

6 Beispiel

```
struct shm_s {
  char message[128];
};
```

```
int main(int argc, char *argv[]) {
  int  shmid;
  struct shm_s *shm;
  char msg[128];
  key_t key;

  key = ftok("/etc/passwd", 42); /* Fehlerbehandlung */
  if ((shmid = shmget(key, sizeof(struct shm_s), 0666 | IPC_CREAT | IPC_EXCL)) == -1){
     /* Fehlerbehandlung */
  }
  shm = (struct shm_s*) shmat(shmid, 0, 0); /* Fehlerbehandlung */
  for(;;) {
    fgets(msg, 128, stdin);
    sprintf(shm->message, "%s", msg);
  }
}
```

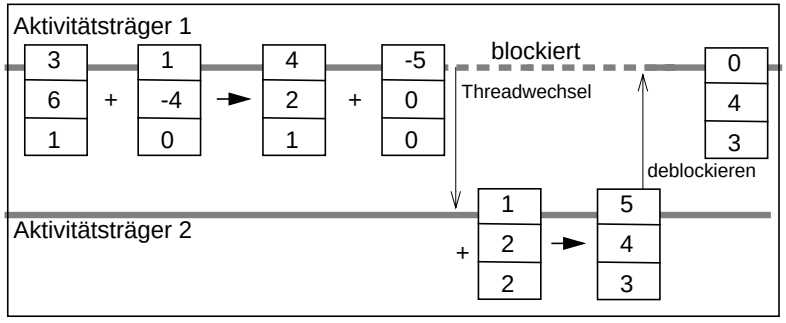
Erzeuger

```
int main(int argc, char *argv[]) {
  int  shmid;
  struct shm_s *shm;

  if ((shmid = shmget (ftok("/etc/passwd", 42), sizeof(struct shm_s), 0)) == -1) {
     /* Fehlerbehandlung */
  }

  shm = (struct shm_s*) shmat(shmid, 0, 0); /* Fehlerbehandlung */

  for(;;) {
    printf("%.128s\n", shm->message);
  }
}
```

Verbraucher

J.4 Semaphore

- Einfache P/V-Semaphore: zwei atomare Operationen:
  - P: blockiere, wenn Semaphorwert gleich 0, sonst erniedrige um 1
  - V: erhöhe Semaphorwert um 1 und evtl. wecke einen an der Semaphore blockierten Aktivitätsträger auf
- allgemeiner: Vektoradditionssystem



1 Erzeugen von Semaphoren

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

int semget(key_t key, int nsems, int semflg);
```

- **key**: Schlüssel ähnlich Shared Memory
- **nsems**: Größe des Semaphor-Vektors
- **semflg**: Flags ähnlich Shared Memory

2 Semaphore-Operationen: semop

```
int semop(int semid, struct sembuf *sops, size_t nsops);
```

- **struct sembuf** enthält Parameter einer Semaphor-Einzeloperation:

```
struct sembuf {
    short sem_num;    /* Nummer der Semaphore */
    short sem_op;     /* Operation */
    short sem_flg;    /* Flags */
}
```

- **nsops** gibt an, wie viele Operationen ausgeführt werden sollen
- die **semop**-Operation blockiert, wenn mindestens eine der Einzeloperationen blockieren würde (**semop** ist atomar!)
- eine Einzeloperation blockiert:
  - ➤ bei **sem_op** < 0: wenn Wert der einzelnen Semaphore dadurch < 0 würde
  - ➤ bei **sem_op** = 0: wenn der Wert der einzelnen Semaphore ungleich 0 ist (soll dagegen Wert einer einzelnen Sem. nicht verändert werden, wird für diese Semaphore einfach keine **sembuf**-Struktur bei **semop** übergeben)

3 Kontrolle der Semaphoren: semctl

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

int semctl(int semid, int semnum, int cmd, . . .);
```

- Beispiele
  - **IPC_RMID**: Löschen des Semaphorenvektors

    ```
    semctl(semid, 0, IPC_RMID);
    ```
  - **GETVAL:** Abfragen des Wertes einer Semaphore
  - **SETVAL**: Setzen einer Semaphore
  - **GETALL:** Abfragen der Werte aller Semaphoren
  - **SETALL**: Setzen aller Semaphoren

4 Semaphore erzeugen, initialisieren, Operation

```
int main(int argc, char *argv[]) {
  int semid;
  ushort vals[3];
  struct sembuf sops[2];
  key_t key;
  union semun {
    int val;
    struct semid_ds *buf;
    ushort *array;
  } arg;

  key = ftok("/etc/passwd", 42);
  if ((semid = semget (key, 3, 0666 | IPC_CREAT | IPC_EXCL)) == -1) {
    perror("semget");
    exit(EXIT_FAILURE);
  }

  vals[0] = 3;
  vals[1] = 4;
  vals[2] = 1;
  arg.array = vals;

  if (semctl(semid, 0, SETALL, arg)==-1) {
    perror("semctl");
    exit(1);
  }

  sops[0].sem_num = 1;
  sops[0].sem_op = -5; /* dieser Wert führt zur Blockierung */
  sops[0].sem_flg = 0;

  sops[1].sem_num = 0;
  sops[1].sem_op =  1;
  sops[1].sem_flg = 0;

  if (semop(semid, sops, 2)==-1) {
    perror("semop"); /* Fehlerbehandlung, z.B.: errno==EINTR -> Wiederaufsetzen von semop */
    exit(1);
  }
}
```


5 Semaphore: anfordern, Operation

```
int main(int argc, char *argv[]) {
  int semid;
  struct sembuf sops[2];
  key_t key;

  key = ftok("/etc/passwd", 42);
  if ((semid = semget (key, 3, 0)) == -1) {
     exit(EXIT_FAILURE);
  }

  sops[0].sem_num = 1;
  sops[0].sem_op =  1;
  sops[0].sem_flg = 0;

  if (semop(semid, sops, 1)==-1) {
    perror("semop");
    exit(1);
  }
}
```


6 Semaphore: Wert ermitteln

```
int main(int argc, char *argv[]) {
  int semid;
  key_t key;
  ushort vals[3];
  union semun {
    int val;
    struct semid_ds *buf;
    ushort *array;
  } arg;
  int i;


  key = ftok("/etc/passwd", 42);
  if (key==-1) { perror("ftok"); exit(EXIT_FAILURE);}
  if ((semid = semget (key, 3, 0)) == -1) {
    perror("semget");
    exit(EXIT_FAILURE);
  }

  arg.array = vals;

  if (semctl(semid, 0, GETALL, arg)==-1) {
    perror("semctl");
    exit(1);
  }

  for(i=0;i<3;i++) {
    printf("%d: %d\n", i, vals[i]);
  }
}
```


J.5 Nützliche Programme für IPC

- **`ipcs`**: Anzeige des Status von IPC Ressourcen (Message Queues, Shared Memory, Semaphore)
- **`ipcrm`**: Entfernen von IPC Ressourcen
  - z.B. **`ipcrm -m <shmid>`**

J.6 Der Ringpuffer