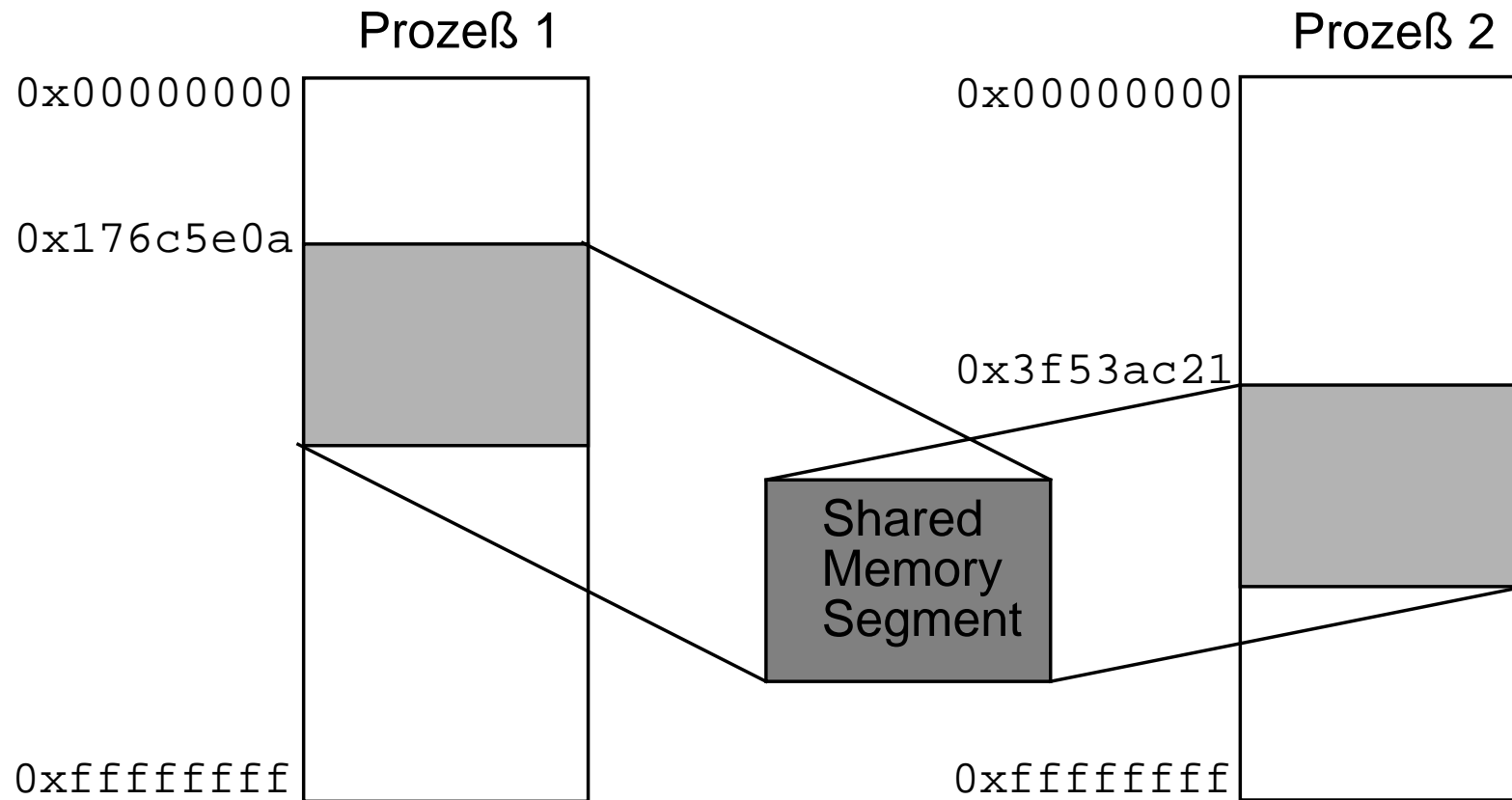


12. Tafelübung

- Shared Memory
- Semaphoren

Shared Memory



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);
```

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

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

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

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.

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 Lösung: Zeiger relativ zum Segmentanfang

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

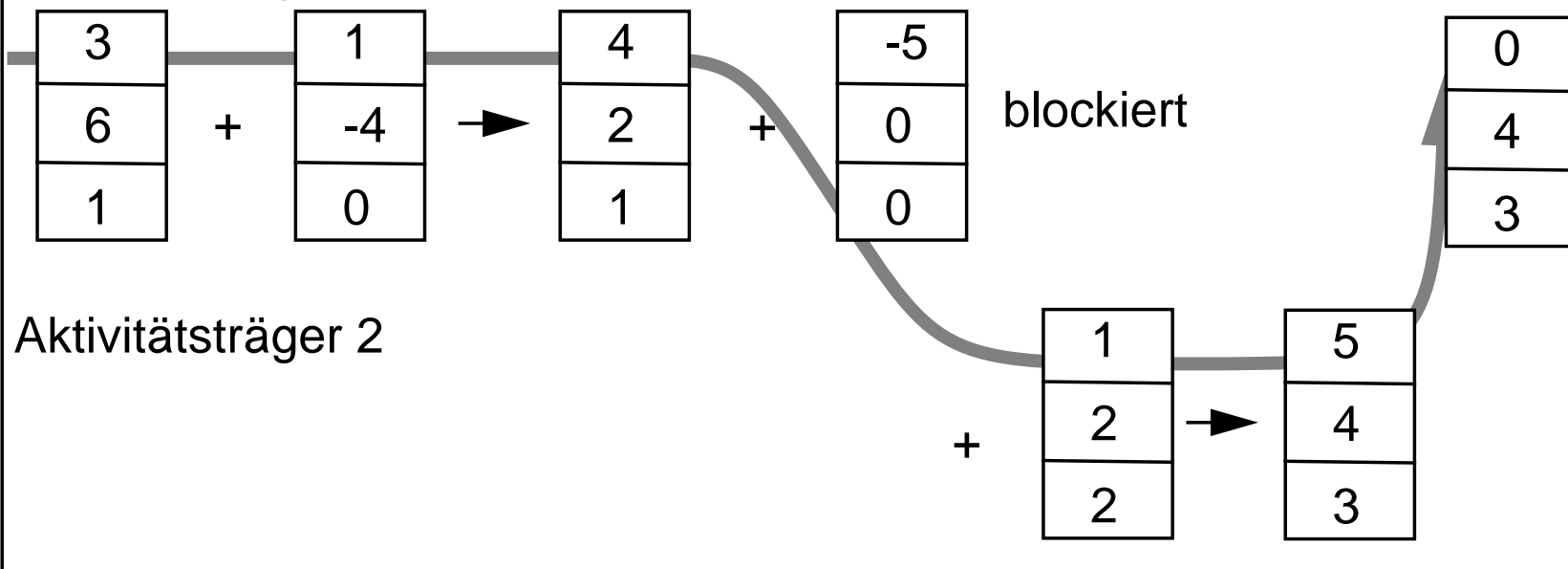
```
int main(int argc, char *argv[]) {  
    int shmid;  
    struct shm_s *shm;  
  
    if ((shmid = shmget (ftok("/etc/passwd", 42), sizeof(struct shm_s), 0)) == -1) {  
        /* Fehlerbehandlung */  
    }  
  
    shm = (struct shm_s*) shmat(shmid, 0, 0); /* Fehlerbehandlung */  
  
    for(;;) {  
        printf("%.128s\n", shm->message);  
    }  
}
```

Verbraucher

Semaphoren

- 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

Aktivitätsträger 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

Semaphore-Operationen: semop

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

- struct sembuf enthält Parameter einer Semaphoroperation:

```
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 Operation blockiert, wenn mindestens eine der Einzeloperationen blockieren würde (**semop** ist atomar!)
- sem_op < 0: P-Operation
- sem_op > 0: V-Operation
- sem_op = 0: Test auf 0

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

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>`

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);
    }
}
```

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);
    }
}
```


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]);
    }
}
```

Der Ringpuffer

