EE3233 Systems Programming for Engrs

Reference: M. Kerrisk, The Linux Programming Interface

Lecture 16

System V Shared Memory

ECE Electrical & Computer Engineering

Shared Memory

- allows two or more processes to share the same region of physical memory
- Shared memory becomes part of a process's user-space memory
  - no kernel intervention is required
  - Some method of synchronization is required so that processes don't simultaneously access the shared memory
  - We have learned System V semaphore as a natural method for such synchronization

Overview

- Typical steps of using a shared memory
  - ① **shmget()**: create a new shared memory or obtain identifier of an existing segment
  - ② **shmat()**: attach the shared memory (make the part of the virtual memory of the calling process)
  - ③ use shared memory referring *addr* returned by *shmat()*
  - ④ **shmdt()**: detach the shared memory segment
  - ⑤ **shmctl()**: delete the shared memory segment

Creating or Opening a Shared Memory

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

int shmget(key_t key, size_t size, int shmflg);
                    returns shared memory segment identifier on success, or -1 on error
```

- creates a new shared memory segment or obtains the identifier of an existing segment
- The content of a newly created shared memory segment are initialized to 0
- *key*: unique key. **IPC_PRIVATE** always results in the creation of a new object guaranteed to have a unique identifier
- *size*: desired size of the segment in bytes
- *shmflg*: permissions to be placed or checked. **IPC_CREAT** creates a new segment

Using Shared Memory

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

int *shmat(int shmid, const void *shmaddr, int shmflg);
        returns address at which shared memory is attached on success, or -1 on error
```

- attaches the shared memory segment identified by *shmid* to the calling process's virtual address space
- *shmaddr*
  - if NULL, then segment is attached at a suitable address selected by the kernel
  - if not NULL, and **SHM_RND** is not set, then the segment is attached at the address specified by *shmaddr*
  - if not NULL, and **SHM_RND** is set, then the segment is mapped at the address provided in *shmaddr* rounded down to the nearest multiple of the constant **SHMLBA**(SHared Memory Low Boundary Address)
- *shmflg*
  - To attach a shared memory segment for read-only access, specify **SHM_RDONLY** in *shmflg*
  - If **SHM_RDONLY** is not specified, memory can be both read and modified

Using Shared Memory

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

int *shmdt(const void *shmaddr);
                                        returns 0 on success, or -1 on error
```

- When a process no longer needs to access a shared memory segment, call *shmdt()* to detach
- *shmaddr*
  - identifies the segment to be detached

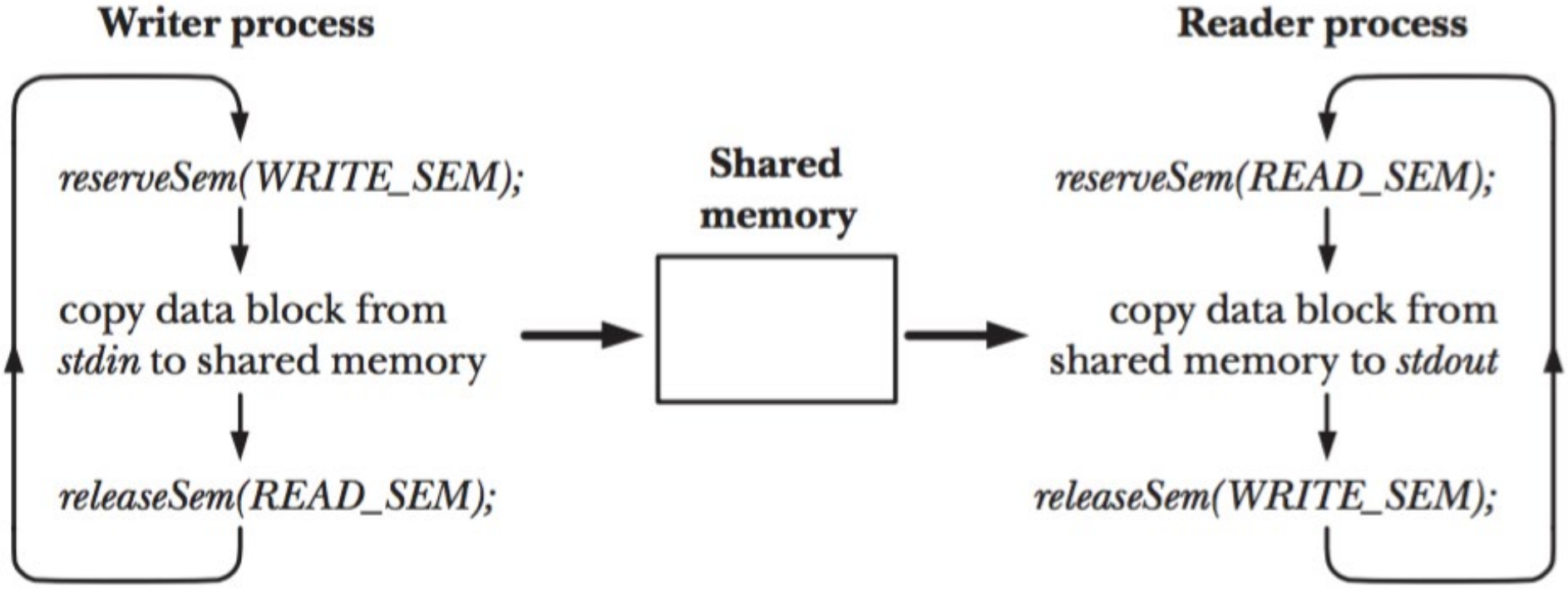
deleting

- use *shmctl()* **IPC_RMID** operation
- During an *exec()*, all attached shared memory segments are detached,
- Shared memory segments are also automatically detached on process termination

Transferring Data via Shared Memory

- [Example] We use shared memory and semaphores
  - consists of two programs: *writer* and *reader*
- Two programs employ a pair of semaphores in a binary semaphore protocol
  - *initSemAvailable()*
  - *initSemInUse()*
  - *reserveSem()*
  - *releaseSem()*

Transferring Data via Shared Memory



svshm/svshm_xfr.h

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include "binary_sems.h"        /* Declares our binary semaphore functions */
#include "tlpi_hdr.h"

#define SHM_KEY 0x1234          /* Key for shared memory segment */
#define SEM_KEY 0x5678          /* Key for semaphore set */

#define OBJ_PERMS (S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP)
                                /* Permissions for our IPC objects */

#define WRITE_SEM 0             /* Writer has access to shared memory */
#define READ_SEM 1              /* Reader has access to shared memory */

#ifndef BUF_SIZE                /* Allow "cc -D" to override definition */
#define BUF_SIZE 1024           /* Size of transfer buffer */
#endif

struct shmseg {                 /* Defines structure of shared memory segment */
    int cnt;                    /* Number of bytes used in 'buf' */
    char buf[BUF_SIZE];         /* Data being transferred */
};
```

svshm/svshm_xfr_writer.c

```
#include "semun.h"                  /* Definition of semun union */
#include "svshm_xfr.h"

int
main(int argc, char *argv[])
{
    int semid, shmid, bytes, xfrs;
    struct shmseg *shmp;
    union semun dummy;

①   semid = semget(SEM_KEY, 2, IPC_CREAT | OBJ_PERMS);
    if (semid == -1)
        errExit("semget");
    if (initSemAvailable(semid, WRITE_SEM) == -1)
        errExit("initSemAvailable");
    if (initSemInUse(semid, READ_SEM) == -1)
        errExit("initSemInUse");

②   shmid = shmget(SHM_KEY, sizeof(struct shmseg), IPC_CREAT | OBJ_PERMS);
    if (shmid == -1)
        errExit("shmget");

    shmp = shmat(shmid, NULL, 0);
    if (shmp == (void *) -1)
        errExit("shmat");
```

svshm/svshm_xfr_writer.c

```
/* Transfer blocks of data from stdin to shared memory */

③   for (xfrs = 0, bytes = 0; ; xfrs++, bytes += shmp->cnt) {
④       if (reserveSem(semid, WRITE_SEM) == -1)         /* Wait for our turn */
            errExit("reserveSem");

⑤       shmp->cnt = read(STDIN_FILENO, shmp->buf, BUF_SIZE);
        if (shmp->cnt == -1)
            errExit("read");

⑥       if (releaseSem(semid, READ_SEM) == -1)          /* Give reader a turn */
            errExit("releaseSem");

        /* Have we reached EOF? We test this after giving the reader
           a turn so that it can see the 0 value in shmp->cnt. */

⑦       if (shmp->cnt == 0)
            break;
    }
```

```
    /* Wait until reader has let us have one more turn. We then know
       reader has finished, and so we can delete the IPC objects. */

⑧   if (reserveSem(semid, WRITE_SEM) == -1)
        errExit("reserveSem");

⑨   if (semctl(semid, 0, IPC_RMID, dummy) == -1)
        errExit("semctl");
    if (shmdt(shmp) == -1)
        errExit("shmdt");
    if (shmctl(shmid, IPC_RMID, 0) == -1)
        errExit("shmctl");

    fprintf(stderr, "Sent %d bytes (%d xfrs)\n", bytes, xfrs);
    exit(EXIT_SUCCESS);
}
```

svshm/svshm_xfr_reader.c

```
#include "svshm_xfr.h"

int
main(int argc, char *argv[])
{
    int semid, shmid, xfrs, bytes;
    struct shmseg *shmp;

    /* Get IDs for semaphore set and shared memory created by writer */

①   semid = semget(SEM_KEY, 0, 0);
    if (semid == -1)
        errExit("semget");

    shmid  = shmget(SHM_KEY, 0, 0);
    if (shmid == -1)
        errExit("shmget");

②   shmp = shmat(shmid, NULL, SHM_RDONLY);
    if (shmp == (void *) -1)
        errExit("shmat");
```

svshm/svshm_xfr_reader.c

```
        /* Transfer blocks of data from shared memory to stdout */

③       for (xfrs = 0, bytes = 0; ; xfrs++) {
④           if (reserveSem(semid, READ_SEM) == -1)          /* Wait for our turn */
                errExit("reserveSem");

⑤           if (shmp->cnt == 0)                         /* Writer encountered EOF */
                break;
            bytes += shmp->cnt;

⑥           if (write(STDOUT_FILENO, shmp->buf, shmp->cnt) != shmp->cnt)
                fatal("partial/failed write");

⑦           if (releaseSem(semid, WRITE_SEM) == -1)         /* Give writer a turn */
                errExit("releaseSem");
        }

⑧       if (shmdt(shmp) == -1)
            errExit("shmdt");

        /* Give writer one more turn, so it can clean up */

⑨       if (releaseSem(semid, WRITE_SEM) == -1)
            errExit("releaseSem");

        fprintf(stderr, "Received %d bytes (%d xfrs)\n", bytes, xfrs);
        exit(EXIT_SUCCESS);
    }
```

Results

```
$ wc -c /etc/services                                   Display size of test file
764360 /etc/services
$ ./svshm_xfr_writer < /etc/services &
[1] 9403
$ ./svshm_xfr_reader > out.txt
Received 764360 bytes (747 xfrs)                        Message from reader
Sent 764360 bytes (747 xfrs)                            Message from writer
[1]+  Done                    ./svshm_xfr_writer < /etc/services
$ diff /etc/services out.txt
$
```

Transferring Data via Shared Memory