Shared Memory Segments and POSIX Semaphores [1]

Alex Delis
delis -at+ pitt.edu

October 2012

[1] Acknowledgements to Prof. T. Stamatopoulos, M. Avidor, Prof. A. Deligiannakis, S. Evangelatos, Dr. V. Kanitkar and Dr. K. Tsakalozos.

Mechanisms for IPCs

- Three (more) types of IPCs:
  - Message Queues (not discussed)
  - Shared Memory
  - Semaphores
- Each IPC structure is referred to by a non-negative integer identifier.
  - When an IPC is created, the program responsible for this creation provides a key of type *key_t*.
  - The Operating System converts this key into an IPC identifier.

Keys in the IPC Client-Server Paradigm

$\Rightarrow$ Keys can be created in three ways:

1. The "server" program creates a new structure by specifying a private key that is *IPC_PRIVATE*.
   - Client has to become explicitly aware of this private key.
   - This is often accomplished with the help of a file generated by the server and then looked-up by the client.
2. Server and client do agree on a key value (often defined and hard-coded in the header).
3. Server and client can agree on a pathname to an existing file in the file system *AND* a project-ID (0..255) and then call *ftok()* to convert these two values into a unique key!

Keys

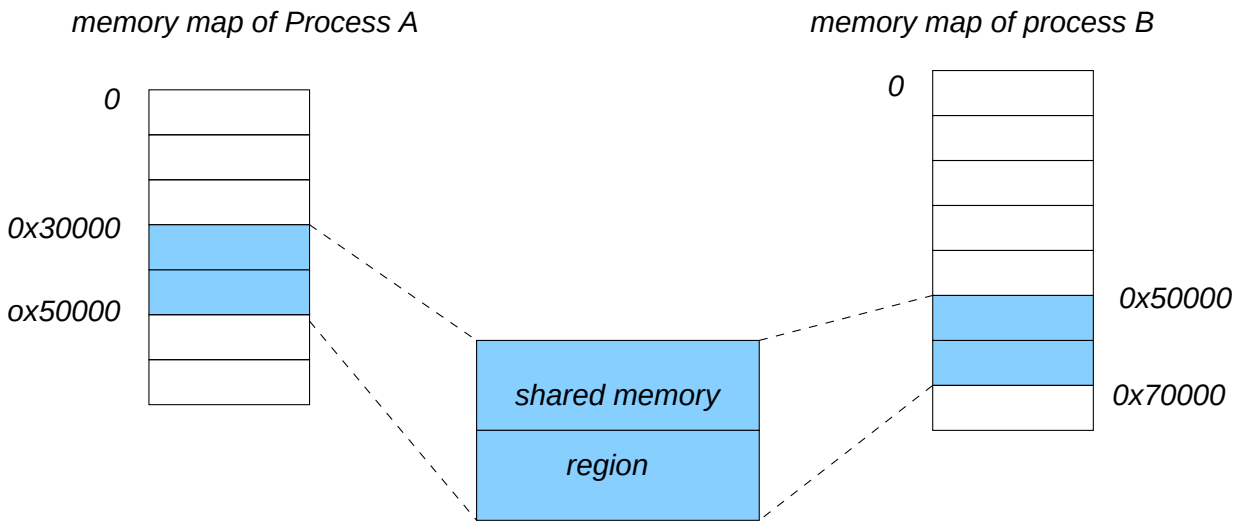
- Keys help identify resources and offer access to the internal structures of the 2 IPC mechanisms (through systems calls):
  - *struct shmid_ds* // for shared segments
  - *struct semid_ds* // for semaphores
- Wrongly accessing resources returns -1
- Access rights for IPC mechanisms: read/write stored in *struct ipc_perm*
- Included header files:
  #include <sys/ipc.h>
  #include <sys/types.h>

The *ftok()* system call

- converts a pathname and a project identifier to an IPC-key
- *key_t ftok(const char *pathname, int proj_id)*
- ```
  if ( (thekey=ftok("/tmp/ad.tempfile", 23)) == -1)
      perror("Cannot create key from /tmp/ad.
          tempfile");
  ```
- The file */tmp/ad.tempfile* must be accessible by the invoking process.
  ```

Shared Memory

- A shared memory region is a portion of physical memory that is shared by multiple processes.



- In this region, structures can be set up by processes and others may read/write on them.
- Synchronization (when it is required) is achieved with the help of semaphores.

Creating a shared segment with *shmget()*

- *#include <sys/ipc.h>*
  #include <sys/shm.h>

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

- returns the identifier of the shared memory segment associated with the value of the argument *key*.
- the returned size of the segment is equal to *size* rounded up to a multiple of *PAGE_SIZE*.
- *shmflg* helps designate the access rights for the segment (*IPC_CREAT* and *IPC_EXCL* are used in a way similar to that of message queues).
- If *shmflg* specifies *both IPC_CREAT* and *IPC_EXCL* and a shared memory segment already exists for *key*, then *shmget()* fails with *errno* set to *EEXIST*.

Attach- and Detach-ing a segment:*shmat()/shmdt()*

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

- attaches the shared memory segment identified by *shmid* to the address space of the calling process.
- If *shmaddr* is NULL, the OS chooses a suitable (unused) address at which to attach the segment (frequent choice).
- Otherwise, *shmaddr* must be a page-aligned address at which the attach occurs.

*int shmdt(const void *shmaddr)*

- detaches the shared memory segment located at the address specified by *shmaddr* from the address space of the calling process.

The system call *shmctl()*

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

- performs the control operation specified by *cmd* on the shared memory segment whose identifier is given in *shmid*.
- The *buf* argument is a pointer to a *shmid_ds* structure:

```
struct shmid_ds {
    struct ipc_perm shm_perm;    /* Ownership and permissions */
    size_t          shm_segsz;   /* Size of segment (bytes) */
    time_t          shm_atime;   /* Last attach time */
    time_t          shm_dtime;   /* Last detach time */
    time_t          shm_ctime;   /* Last change time */
    pid_t           shm_cpid;    /* PID of creator */
    pid_t           shm_lpid;    /* PID of last shmat(2)/shmdt(2) */
    shmatt_t        shm_nattch;  /* No. of current attaches */
    ...
};
```

The system call *shmctl()*

Usual values for *cmd* are:

- *IPC_STAT*: copy information from the kernel data structure associated with *shmid* into the *shmid_ds* structure pointed to by *buf*.
- *IPC_SET*: write the value of some member of the *shmid_ds* structure pointed to by *buf* to the kernel data structure associated with this shared memory segment, updating also its *shm_ctime* member.
- *IPC_RMID*: mark the segment to be destroyed. The segment will be destroyed after the last process detaches it (i.e., *shm_nattch* is zero).

Use Cases of Calls

- Only one process creates the segment:

```
int id;
id = shmget(IPC_PRIVATE, 10, 0666);
if ( id == -1 ) perror("Creating");
```

- Every (interested) process attaches the segment:

```
int *mem;
mem = (int *) shmat (id, (void *)0, 0);
if ( (int)mem == -1 ) perror("Attachment");
```

- Every process detaches the segment:

```
int err;
err = shmdt((void *)mem);
if ( err == -1 ) perror("Detachment");
```

- Only one process has to remove the segment:

```
int err;
err = shmctl(id, IPC_RMID, 0);
if ( err == -1 ) perror("Removal");
```

Creating and accessing shared memory ("shareMem1.c")

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

int main(int argc, char **argv){
    int id=0, err=0;
    int *mem;

    id = shmget(IPC_PRIVATE,10,0666);  /* Make shared memory segment */
    if (id == -1) perror ("Creation");
    else printf("Allocated. %d\n",(int)id);

    mem = (int *) shmat(id, (void*)0, 0); /* Attach the segment */
    if ((int) mem == -1) perror("Attachment.");
    else printf("Attached. Mem contents %d\n",*mem);

    *mem=1; /* Give it initial value */
    printf("Start other process. >"); getchar();

    printf("mem is now %d\n", *mem); /* Print out new value */

    err = shmctl(id, IPC_RMID, 0); /* Remove segment */
    if (err == -1) perror ("Removal.");
    else printf("Removed. %d\n", (int)(err));
    return 0;
}
```

Creating and accessing shared memory ("shareMem2.c")

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

int main(int argc, char **argv) {
    int id, err;
    int *mem;

    if (argc <= 1) { printf("Need shared memory id. \n"); exit(1); }

    sscanf(argv[1], "%d", &id); /* Get id from command line. */
    printf("Id is %d\n", id);

    mem = (int *) shmat(id, (void*) 0,0); /* Attach the segment */
    if ((int) mem == -1) perror("Attachment.");
    else printf("Attached. Mem contents %d\n",*mem);

    *mem=2; /* Give it a different value */
    printf("Changed mem is now %d\n", *mem);

    err = shmdt((void *) mem); /* Detach segment */
    if (err == -1) perror ("Detachment.");
    else printf("Detachment %d\n", err);
    return 0;
}
```

Running the two programs:

- Starting off with executing "shareMem1":

```
ad@ad-desktop:~/Set008/src/SharedSegments$ ./shareMem1
Allocated. 1769489
Attached. Mem contents 0
Start other process. >
```

- Executing "shareMem2":

```
ad@ad-desktop:~/Set008/src/SharedSegments$ ./shareMem2 1769489
Id is 1769489
Attached. Mem contents 1
Changed mem is now 2
Detachment 0
ad@ad-desktop:~/Set008/src/SharedSegments$
```

- Providing the final input to "shareMem1":

```
Start other process. >s
mem is now 2
Removed. 0
ad@ad-desktop:~/Set008/src/SharedSegments$
```

Semaphores

- Fundamental mechanism that facilitates the synchronization of accessing resources placed in shared memory.
- A semaphore is an integer whose value is never allowed to fall below zero.
- *Two operations* can be performed on a semaphore:
  - increment the semaphore value by one (*UP* or *V()* ala Dijkstra).
  - decrement a semaphore value by one (*DOWN* or *P()* ala Dijkstra). If the value of semaphore is currently zero, then the invoking process will block until the value becomes greater than zero.

POSIX Semaphores

- *#include <semaphore.h>*
- *sem_init, sem_destroy, sem_post, sem_wait, sem_trywait*
- *int sem_init(sem_t *sem, int pshared, unsigned int value);*
  - The above initializes a semaphore.
  - Compile either with -lrt or -lpthread
  - *pshared* indicates whether this semaphore is to be shared between the threads of a process, or between processes:
    - zero: semaphore is shared between the **threads of a process**; should be located at an address visible to **all threads**.
    - non-zero: semaphore is shared **among processes** and should be located in a region of shared memory.

POSIX Semaphore Operations

- *sem_wait()*, *sem_trywait()*
  - *int sem_wait(sem_t *sem);*
  - *int sem_trywait(sem_t *sem);*
  - Perform $\mathrm{P}(\mathrm{s})$ operation.
  - *sem_wait* blocks; *sem_trywait* will fail rather than block.
- *sem_post()*
  - *int sem_post(sem_t *sem);*
  - Perform $\mathrm{V}(\mathrm{s})$ operation.
- *sem_destroy()*
  - *int sem_destroy(sem_t *sem);*
  - Destroys a semaphore.

Creating and using a POSIX Semaphore

```c
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/ipc.h>

extern int errno;

int main(int argc, char **argv)
{
    sem_t sp; int retval;

    /* Initialize the semaphore. */
    retval = sem_init(&sp,1,2);
    if (retval != 0) {
        perror("Couldn't initialize."); exit(3); }

    retval = sem_trywait(&sp);
    printf("Did trywait. Returned %d >\n",retval); getchar();

    retval = sem_trywait(&sp);
    printf("Did trywait. Returned %d >\n",retval); getchar();

    retval = sem_trywait(&sp);
    printf("Did trywait. Returned %d >\n",retval); getchar();

    sem_destroy(&sp);
    return 0;
}
```

Executing the Program

```
ad@ad-desktop:~/src/PosixSems$ ./semtest
Did trywait. Returned 0 >

Did trywait. Returned 0 >

Did trywait. Returned -1 >

ad@ad-desktop:~/src/PosixSems$
```

Example: semtest3.c

```c
/* semtest3.c: POSIX Semaphore test example using shared memory */
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SEGMENTSIZE sizeof(sem_t)
#define SEGMENTPERM 0666

int main(int argc, char **argv)
{
    sem_t *sp;
    int retval;
    int id, err;

    /* Make shared memory segment. */
    id = shmget(IPC_PRIVATE, SEGMENTSIZE, SEGMENTPERM);
    if (id == -1) perror("Creation");
    else printf("Allocated %d\n", id);

    /* Attach the segment. */
    sp = (sem_t *) shmat(id,(void*) 0, 0);
    if ((int) sp == -1) { perror("Attachment."); exit(2);}
```

Example: semtest3.c

```
    /* Initialize the semaphore. */
    retval = sem_init(sp,1,2);
    if (retval != 0) {
        perror("Couldn't initialize.");
        exit(3);
    }
    retval = sem_trywait(sp);
    printf("Did trywait. Returned %d >\n", retval); getchar();

    retval = sem_trywait(sp);
    printf("Did trywait. Returned %d >\n", retval); getchar();

    retval = sem_trywait(sp);
    printf("Did trywait. Returned %d >\n", retval); getchar();

    sem_destroy(sp);

    /* Remove segment. */
    err = shmctl(id, IPC_RMID, 0);
    if (err == -1) perror("Removal.");
    else printf("Removed. %d\n",err);
    return 0;
}
```

Example: semtest3a.c

```c
/* semtest3a.c POSIX Semaphore test example using shared memory */
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SEGMENTSIZE sizeof(sem_t)
#define SEGMENTPERM 0666

extern int errno;

int main(int argc,char **argv)
{
    sem_t *sp;
    int retval;
    int id, err;

    if (argc <= 1) { printf("Need shmem id. \n"); exit(1); }

    /* Get id from command line. */
    sscanf(argv[1], "%d", &id);
    printf("Allocated %d\n", id);

    /* Attach the segment. */
    sp = (sem_t *) shmat(id,(void*) 0, 0);
    if ((int) sp == -1) { perror("Attachment."); exit(2);}
```

Example: semtest3a.c

```
    /* Initialize the semaphore. */
    retval = sem_init(sp,1,1);
    if (retval != 0) {
        perror("Couldn't initialize.");
        exit(3);
    }

    retval = sem_trywait(sp);
    printf("Did trywait. Returned %d >\n", retval); getchar();

    retval = sem_trywait(sp);
    printf("Did trywait. Returned %d >\n", retval); getchar();

    retval = sem_trywait(sp);
    printf("Did trywait. Returned %d >\n", retval); getchar();

    /* Remove segment. */
    err = shmdt((void *) sp);
    if (err == -1) perror ("Detachment.");
    return 0;
}
```

Running the two programs: semtest and semtest3a

- Starting off with executing "semtest":

```
ad@serifos:~/Recitation4/src$ ./semtest3
Allocated 14549024
Did trywait. Returned 0 >
```

- Executing "semtest3a" in another tty:

```
ad@serifos:~/Recitation4/src$ ./semtest3a 14549024
Allocated 14549024
Did trywait. Returned 0 >

Did trywait. Returned -1 >
```

- Continue with "semtest3":

```
Did trywait. Returned -1 >

Did trywait. Returned -1 >
```

Running the two programs: semtest and semtest3a

- Continue with "semtest3a":

```
Did trywait. Returned -1 >
```

- Follow up with "semtest3":

```
Removed. 0
ad@serifos:~/Recitation4/src$
```

- Finish with "semtest3a":

```

ad@serifos:~/Recitation4/src$
```

Initialize and Open a **named** *Semaphore*

- *sem_t *sem_open(const char *name, int oflag);*
  *sem_t *sem_open(const char *name, int oflag,*
  mode_t mode, unsigned int value);
- creates a new POSIX semaphore OR opens an existing semaphore whose name is *name*.
- *oflag* specifies flags that control the operation of the call
  - *O_CREAT* creates the semaphore;
  - provided that both *O_CREAT* and *O_EXCL* are specified, an *error* is returned if a semaphore with *name* already exists.
- if *oflag* is *O_CREAT* then **two more arguments** have to be used:
  - *mode* specifies the permissions to be placed on the new semaphore.
  - *value* specifies the initial value for the new semaphore.

More on Named POSIX Semaphores

- A named semaphore is identified by a (persistent) name that has the form /this_is_a_sample_named_semaphore.
  - consists of an initial slash followed by a (large) number of character (but no slashes).
- If you want to "see" (list) all named sempahores in your (Linux) system look at directory */dev/shm*
- *int sem_close(sem_t *sem)*
  - closes the named semaphore referred to by *sem* freeing the system resources the invoking process has used.
- *int sem_unlink(const char *name)*
  - removes the named semaphore in question.
- *int sem_getvalue(sem_t *sem, int *sval)*
  - obtains the current value of semaphore..
  - the cheater API-call!

Named POSIX Semaphore

```
#include        <stdio.h>
...
#include        <sys/stat.h>
#include        <semaphore.h>

int main(int argc, char *argv[]){
const char *semname;
int op=0; int val=0;

if (argc==3)  {
        semname=argv[1]; op=atoi(argv[2]);
        }
else    {
        printf("usage: nameSem nameOfSem Operation\n"); exit(1);
        }

sem_t *sem=sem_open(semname, O_CREAT|O_EXCL, S_IRUSR|S_IWUSR, 0);

if (sem!= SEM_FAILED)
        printf("created new semaphore!\n");
else if (errno== EEXIST ) {
        printf("semaphore appears to exist already!\n");
        sem = sem_open(semname, 0);
        }
else ;

assert(sem != SEM_FAILED);
sem_getvalue(sem, &val);
printf("semaphore's before action value is %d\n",val);
```

Named Posix Semaphore

```
if ( op == 1 ) {
        printf("incrementing semaphore\n");
        sem_post(sem);
        }
else if ( op == -1 ) {
        printf("decrementing semaphore\n");
        sem_wait(sem);
        }
else if ( op == 2 ){
        printf("clearing up named semaphore\n");
        sem_close(sem);  // close the sem
        sem_unlink(semname); // remove it from system
        exit(1);
        }
else    printf("not defined operation! \n");

sem_getvalue(sem, &val);
printf("semaphore's current value is %d\n",val);
sem_close(sem);
return(0);
}
```

Execution Outcome

```
ad@serifos:~/PosixSems$ ls /dev/shm/
pulse-shm-1024070233  pulse-shm-1294442337  pulse-shm-2927836935
pulse-shm-1274848112  pulse-shm-2305588894  pulse-shm-3888866544
ad@serifos:~/PosixSems$ ./namedSem /delis 1
created new semaphore!
semaphore's before action value is 0
incrementing semaphore
semaphore's current value is 1
ad@serifos:~/PosixSems$ ls /dev/shm/
pulse-shm-1024070233  pulse-shm-1294442337  pulse-shm-2927836935  sem.delis
pulse-shm-1274848112  pulse-shm-2305588894  pulse-shm-3888866544
ad@serifos:~/PosixSems$ ./namedSem /delis -1
semaphore appears to exist already!
semaphore's before action value is 1
decrementing semaphore
semaphore's current value is 0
ad@serifos:~/PosixSems$ ./namedSem /delis 2
semaphore appears to exist already!
semaphore's before action value is 0
clearing up named semaphore
ad@serifos:~/PosixSems$ ls /dev/shm/
pulse-shm-1024070233  pulse-shm-1294442337  pulse-shm-2927836935
pulse-shm-1274848112  pulse-shm-2305588894  pulse-shm-3888866544
ad@serifos:~/PosixSems$ ./namedSem /delis 1
created new semaphore!
semaphore's before action value is 0
incrementing semaphore
semaphore's current value is 1
```

Execution Outcome

```
ad@serifos:~/PosixSems$ ./namedSem /delis 1
semaphore appears to exist already!
semaphore's before action value is 1
incrementing semaphore
semaphore's current value is 2
ad@serifos:~/PosixSems$ ls /dev/shm/
pulse-shm-1024070233  pulse-shm-1294442337  pulse-shm-2927836935  sem.delis
pulse-shm-1274848112  pulse-shm-2305588894  pulse-shm-3888866544
ad@serifos:~/PosixSems$ ./namedSem /delis -1
semaphore appears to exist already!
semaphore's before action value is 2
decrementing semaphore
semaphore's current value is 1
ad@serifos:~/PosixSems$ ./namedSem /delis -1
semaphore appears to exist already!
semaphore's before action value is 1
decrementing semaphore
semaphore's current value is 0
ad@serifos:~/PosixSems$ ./namedSem /delis 2
semaphore appears to exist already!
semaphore's before action value is 0
clearing up named semaphore
ad@serifos:~/PosixSems$ ls /dev/shm/
pulse-shm-1024070233  pulse-shm-1294442337  pulse-shm-2927836935
pulse-shm-1274848112  pulse-shm-2305588894  pulse-shm-3888866544
ad@serifos:~/PosixSems$
```