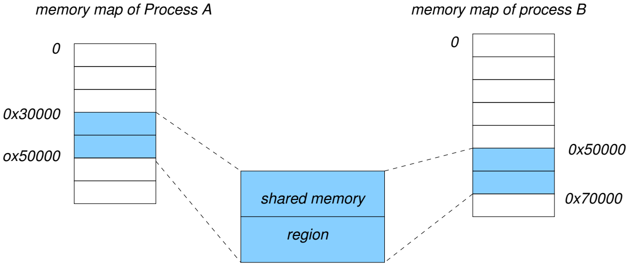
Shared Memory

- 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 among processes using the segment (if 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:

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

- Every (interested) process attaches the segment:

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

- Every process detaches the segment:

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

- Only one process has to remove the segment:

```c
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":

```
antoulas@sazerac:~/SharedSegments$ ./shareMem1
Allocated. 1769489
Attached. Mem contents 0
Start other process. >
```

- Executing "shareMem2":

```
antoulas@sazerac:~/SharedSegments$ ./shareMem2 1769489
Id is 1769489
Attached. Mem contents 1
Changed mem is now 2
Detachment 0
antoulas@sazerac:~/SharedSegments$
```

- Providing the final input to "shareMem1":

```
Start other process. >s
mem is now 2
Removed. 0
antoulas@sazerac:~/SharedSegments$
```

Semaphores

- Fundamental mechanism that facilitates synchronization and coordinated accessing of resources placed in shared memory.
- A semaphore is an integer whose value is never allowed to fall below zero.
- *Two operations* can be atomically 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.

System-V Semaphores

- In general, (System-V) system calls create sets of semaphores:
  - The kernel warrants atomic operations on these sets.
  - Should we have more than one resources to protect, we can "lock" all of them simultaneously.

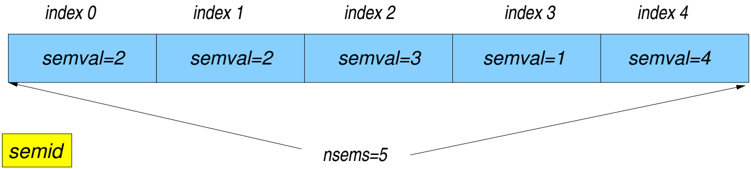
Creating a set of Semaphores

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

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

- returns the semaphore set identifier associated with the argument `key`.
- A new set of `nsems` semaphores is created if `key` has the value `IPC_PRIVATE` **OR** if no existing semaphore set is associated with `key` and `IPC_CREAT` is specified in `semflg`.
- `semflg` helps set the access right for the semaphore set.
- If `semflg` specifies both `IPC_CREAT` and `IPC_EXCL` and a semaphore set already exists for `key`, then `semget()` fails with `errno` set to `EEXIST`.

Structure of a Semaphore Set



index 0	*index 1*	*index 2*	*index 3*	*index 4*
semval=2	*semval=2*	*semval=3*	*semval=1*	*semval=4*

semid

nsems=5

Associated with each (single) semaphore in the set are the following values:

- `semval`: the semaphore value, always a positive number.
- `sempid`: *pid* of the process that last “acted” on semaphore.
- `semcnt`: number of processes waiting for the semaphore to reach value greater that its current one.
- `semzcnt`: number of processes waiting for the semaphore to reach value zero.

Operating on a Set of Semaphores

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

- performs operations on *selected* semaphores in the set indicated by `semid`.
- each of the `nsops` elements in the array pointed to by `sops` specifies an operation to be performed on a single semaphore on the set.

Operating on a Set of Semaphores

- The elements of the `struct sembuf` have as follows:

```
struct sembuf{
    unsigned short sem_num;  /* semaphore number */
    short          sem_op;   /* semaphore operation */
    short          sem_flg;  /* operation flags */
    };
```

- In the above:
  - `sem_num` identifies the ID of the specific semaphore on the set on which `sem_op` operates.
  - The value of `sem_op` is set to:
    - < 0 for locking
    - > 0 for unlocking
  - `sem_flg` often set to 0.

The semctl() system call

```
int semctl(int semid, int semnum, int cmd,
            [union semun arg])
```

- performs the control operation specified by cmd on the semnum-th semaphore of the set identified by semid.
- The 4th parameter above –if it exists– has the following layout:

```
union semun {
    int              val;    /* Value for SETVAL */
    struct semid_ds *buf;    /* Buffer for IPC_STAT, IPC_SET */
    unsigned short  *array;  /* Array for GETALL, SETALL */
    struct seminfo  *__buf;  /* Buffer for IPC_INFO (Linux-specific) */
};
```

The semid_ds structure

- The semaphore data structure semid_ds, is as follows:

```
struct semid_ds {
     struct ipc_perm sem_perm;  /* Ownership and permissions */
     time_t          sem_otime; /* Last semop time */
     time_t          sem_ctime; /* Last change time */
     unsigned short  sem_nsems; /* No. of semaphores in set */
     };
```

semctl()

Values for the cmd parameter:

- IPC_STAT: copy information from the kernel data structure associated with semid into the semid_ds structure pointed to by arg.buf.
- IPC_SET: write the value of some member of the semid_ds structure pointed to by arg.buf to the kernel data structure associated with this semaphore set; its sem_ctime member gets updated as well.
- IPC_SETALL: Set semval for all semaphores of the set using arg.array, updating also the sem_ctime member of the semid_ds structure associated with the set.
- IPC_GETALL: Return to semval the current values of all semaphores of the set arg.array.
- IPC_RMID: remove the semaphore set while awakening all processes blocked by the respective semop().

A server program using Semaphores

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

#define SHMKEY (key_t)4321
#define SEMKEY (key_t)9876
#define SHMSIZE 256
#define PERMS 0600

union semnum{
    int val;
    struct semid_ds *buff;
    unsigned short *array; };

main(){
    int shmid, semid; char line[128], *shmem;
    struct sembuf oper[1]={0,1,0};
    union semnum arg;

    if ((shmid = shmget (SHMKEY, SHMSIZE, PERMS | IPC_CREAT)) < 0) {
        perror("shmget"); exit(1); }
    printf("Creating shared memory with ID: %d\n",shmid);
    /* create a semaphore */
    if ((semid = semget(SEMKEY, 1, PERMS| IPC_CREAT)) <0) {
        perror("semget"); exit(1); }
    printf("Creating a semaphore with ID: %d \n",semid);
    arg.val=0;
```

A server program using Semaphores (continued)

```
    /* initialize semaphore for locking */
    if (semctl(semid, 0, SETVAL, arg) <0) {
        perror("semctl");
        exit(1);
        }
    printf("Initializing semaphore to lock\n");

    if ( (shmem = shmat(shmid, (char *)0, 0)) == (char *) -1) {
        perror("shmem");
        exit(1);
        }
    printf("Attaching shared memory segment \nEnter a string: ");
    fgets(line, sizeof(line), stdin);
    line[strlen(line)-1]='\0';

    /* Write message in shared memory */
    strcpy(shmem, line);

    printf("Writing to shared memory region: %s\n", line);

    /* Make shared memory available for reading */
    if ( semop(semid, &oper[0], 1) < 0 ) {
        perror("semop");
        exit(1);
        }
    shmdt(shmem);
    printf("Releasing shared memory region\n");
    }
```

A client program using semaphore

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

#define SHMKEY (key_t)4321
#define SEMKEY (key_t)9876
#define SHMSIZE 256
#define PERMS 0600

main(){
    int shmid, semid;
    char *shmem;
    struct sembuf oper[1]={0,-1,0};

    if ((shmid = shmget (SHMKEY, SHMSIZE, PERMS )) < 0) {
        perror("shmget"); exit(1); }
    printf("Accessing shared memory with ID: %d\n",shmid);

    /* accessing a semaphore */
    if ((semid = semget(SEMKEY, 1, PERMS )) <0) {
        perror("semget"); exit(1); }
    printf("Accessing semaphore with ID: %d \n",semid);
```

A client program using semaphore (continued)

```
    if ( (shmem = shmat(shmid, (char *) 0, 0)) == (char *) -1 ) {
        perror("shmat"); exit(1); }
    printf("Attaching shared memory segment\n");

    printf("Asking for access to shared memory region \n");
    if (semop(semid, &oper[0], 1) <0)  {
        perror("semop"); exit(1); }
    printf("Reading from shared memory region: %s\n", shmem);

    /* detach shared memeory */
    shmdt(shmem);

    /* destroy shared memory */
    if (shmctl(shmid, IPC_RMID, (struct shmid_ds *)0 ) <0) {
        perror("semctl"); exit(1); }
    printf("Releasing shared segment with identifier %d\n", shmid);

    /* destroy semaphore set */
    if (semctl(semid, 0, IPC_RMID, 0) <0 ) {
        perror("semctl"); exit(1); }
    printf("Releasing semaphore with identifier %d\n", semid);
    }
```

Running the server and the client

The server:

```
antoulas@sazerac:~/SysProMaterial/Set008/src/V-Sems$ ./sem-server
Creating shared memory with ID: 22511641
Creating a semaphore with ID: 327688
Initializing semaphore to lock
Attaching shared memory segment
Enter a string:
```

The client:

```
antoulas@sazerac:~/SysProMaterial/Set008/src/V-Sems$ ./sem-client
Accessing shared memory with ID: 22511641
Accessing semaphore with ID: 327688
Attaching shared memory segment
Asking for access to shared memory region
```

Running the programs

- Server:

```
antoulas@sazerac:~/src/V-Sems$ ./sem-server
Creating shared memory with ID: 22511641
Creating a semaphore with ID: 327688
Initializing semaphore to lock
Attaching shared memory segment
Enter a string: THIS IS A TEST ONLY A TEST
Writing to shared memory region: THIS IS A TEST ONLY A TEST
Releasing shared memory region
antoulas@sazerac:~/src/V-Sems$
```

- Client:

```
antoulas@sazerac:~/src/V-Sems$ ./sem-client
Accessing shared memory with ID: 22511641
Accessing semaphore with ID: 327688
Attaching shared memory segment
Asking for access to shared memory region
Reading from shared memory region: THIS IS A TEST ONLY A TEST
Releasing shared segment with identifier 22511641
Releasing semaphore with identifier 327688
antoulas@sazerac:~/src/V-Sems$
```

Access to Critical Section

```
#include <stdio.h> /* Example code using semaphores and shared memory */
#include <stdlib.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/ipc.h>

/* Union semun */
union semun {
   int val;                     /* value for SETVAL */
   struct semid_ds *buf;        /* buffer for IPC_STAT, IPC_SET */
   unsigned short *array;       /* array for GETALL, SETALL */
};

void free_resources(int shm_id, int sem_id) {
   /* Delete the shared memory segment */
   shmctl(shm_id,IPC_RMID,NULL);
   /* Delete the semaphore */
   semctl(sem_id,0,IPC_RMID,0);
}

int sem_P(int sem_id) {      /* Semaphore P - down operation, using semop */
   struct sembuf sem_d;

   sem_d.sem_num = 0;
   sem_d.sem_op = -1;
   sem_d.sem_flg = 0;
   if (semop(sem_id,&sem_d,1) == -1) {
       perror("# Semaphore down (P) operation "); return -1; }
   return 0;
}
```

Access to Critical Section

```c
/* Semaphore V - up operation, using semop */
int sem_V(int sem_id) {
   struct sembuf sem_d;

   sem_d.sem_num = 0;
   sem_d.sem_op = 1;
   sem_d.sem_flg = 0;
   if (semop(sem_id,&sem_d,1) == -1) {
       perror("# Semaphore up (V) operation "); return -1; }
   return 0;
}

/* Semaphore Init - set a semaphore's value to val */
int sem_Init(int sem_id, int val) {
   union semun arg;

   arg.val = val;
   if (semctl(sem_id,0,SETVAL,arg) == -1) {
       perror("# Semaphore setting value "); return -1; }
   return 0;
}
```

Access to Critical Section

```c
int main () {
   int shm_id; int sem_id; int t = 0; int *sh; int pid;

   /* Create a new shared memory segment */
   shm_id = shmget(IPC_PRIVATE,sizeof(int),IPC_CREAT | 0660);
   if (shm_id == -1) {
       perror("Shared memory creation"); exit(EXIT_FAILURE); }

   /* Create a new semaphore id */
   sem_id = semget(IPC_PRIVATE,1,IPC_CREAT | 0660);
   if (sem_id == -1) {
       perror("Semaphore creation ");
       shmctl(shm_id,IPC_RMID,(struct shmid_ds *)NULL);
       exit(EXIT_FAILURE);
   }

   /* Set the value of the semaphore to 1 */
   if (sem_Init(sem_id, 1) == -1) {
       free_resources(shm_id,sem_id);
       exit(EXIT_FAILURE);
   }

   sh = (int *)shmat(shm_id,NULL,0);  /* Attach the shared memory segment */
   if (sh == NULL) {
       perror("Shared memory attach ");
       free_resources(shm_id,sem_id);
       exit(EXIT_FAILURE);
   }
   /* Setting shared memory to 0 */
   *sh = 0;
```

Access to Critical Section

```
/* New process */
if ((pid = fork()) == -1) {
    perror("fork");
    free_resources(shm_id,sem_id);
    exit(EXIT_FAILURE);
}

if (pid == 0) {
   /* Child process */
   printf("# I am the child process with process id: %d\n", getpid());
} else {
   /* Parent process */
   printf("# I am the parent process with process id: %d\n", getpid());
   sleep(2);
}

printf("(%d): trying to access the critical section\n", getpid());
sem_P(sem_id);
printf("(%d): accessed the critical section\n", getpid());

(*sh)++;
printf("(%d): value of shared memory is now: %d\n", getpid(), *sh);

printf("(%d): getting out of the critical section\n", getpid());
sem_V(sem_id);

printf("(%d): got out of the critical section\n", getpid());
```

Access to Critical Section

```
   /* Child process */
   if (!pid)
     exit(EXIT_SUCCESS);

   /* Wait for child process */
   wait(NULL);

   /* Clear recourses */
   free_resources(shm_id,sem_id);
   return 0;
}
```

→ outcome of execution:

```
antoulas@sazerac:~/src/V-Sems$ ./access-criticalsection
# I am the parent process with process id: 9256
# I am the child process with process id: 9257
(9257): trying to access the critical section
(9257): accessed the critical section
(9257): value of shared memory is now: 1
(9257): getting out of the critical section
(9257): got out of the critical section
(9256): trying to access the critical section
(9256): accessed the critical section
(9256): value of shared memory is now: 2
(9256): getting out of the critical section
(9256): got out of the critical section
antoulas@sazerac:~/src/V-Sems$
```