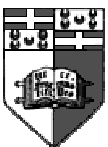




# Semaphores



- This is yet another SVR4 IPC structure following the usual protocol for creation and access.
- It is an overly complicated implementation of semaphores.
- A semaphore is a value which is used to indicate the availability of a resource.
- When a semaphore is positive, a process decrements the semaphore by 1, signifying it has used up one resource.
- When the semaphore is 0, it means the resource is not available and the process blocks until the resource is available.
- Testing and decrementing is done atomically in SVR4 implementations to guarantee functionality.

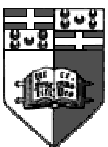




# Semaphore IPC

- Semaphores are created in sets with each set holding one or more semaphores.
- There are limits affecting semaphores:
  - SEMVMX: max value of semaphore.
  - SEMMNI: max number of semaphore sets, system wide.
  - SEMMNS: max number of semaphores, system wide.
  - SEMMSL: max number of semaphores per set.
  - SEMOPM: max number of operations per semop call.
- *semget()* is the usual function to create or obtain access to the IPC.
- *nsems* gives the number of semaphores to create in the set. When accessing an existing semaphore set, this value can be 0.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget(key_t key, int nsems, int flag);
    returns semaphore ID or -1 on error
```





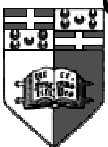
# Semaphore Control

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semctl(int semid, int semno,
            int cmd, union semun arg);
            return depends on cmd
```

```
union semun {
    int val; // SETVAL
    struct semid_ds *buf;
    ushort *array; // GETALL
                  // SETALL
}
```

```
cmd:
    GETVAL
    SETVAL
    GETALL
    SETALL
    IPC_RMID
```

- Value of *semno* varies from 0 to *nsems*-1
- Semaphore values are of type *ushort*.
- `IPC_RMID` removal is immediate and anyone using the semaphore set will return with *errno* equal to `EIDRM`.

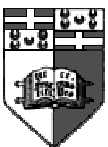


# Unix Semaphore Operations

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf semoparray[],
          size_t nops);
                                returns -1 on error
```

```
struct sembuf {
    ushort sem_num;
    short sem_op;
    short sem_flg;
};
```

- *nops* gives the number of operations in *semoparray*.
- If *sem\_op* is:
  - Positive: semaphore value is added by the value of *sem\_op* (*release resource*).
  - 0: Block until semaphore value becomes equal to zero.
  - Negative: If semaphore value is greater than or equal to absolute value of *sem\_op*, semaphore is subtracted. Otherwise block until we can do this subtraction (*wait to obtain resource*).



# Unix Semaphore Operations (Cont)

- All blocking operations will fail if semaphore is removed (ERMID) or signal is caught (EINTR).
- If flag is set to IPC\_NOWAIT, *semop()* never blocks and returns with errno equal to EAGAIN when an operation was not successful.
- Typical semaphore usage:

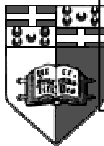
Create semaphore  
Set semaphore value to 1

**Process 1**

Access semaphore  
*semop()* with *sem\_op* = -1  
Use resource  
*semop()* with *sem\_op* = 1

**Process 2**

Access semaphore  
*semop()* with *sem\_op* = -1  
Use resource  
*semop()* with *sem\_op* = 1

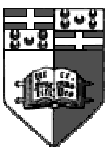




# Shared Memory



- This is an area of memory that can be used by one or more processes.
- It is another SVR4 IPC structure and thus follows the usual conventions for access and creation.
- One has to be careful in synchronizing access to a given region by several processes.
- Limits imposed by system:
  - SHMMAX: max number of bytes in shared memory segment.
  - SHMMIN: minimum number of bytes in shared memory segment.
  - SHMMNI: max number of shared memory segments system wide.
  - SHMSEG: max number of shared memory segments per process.

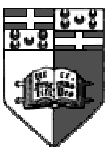


# Unix Shared Memory (cont)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, int size, int flag);
        return shm ID or -1 on error
```

- *size* is the minimum size of the shared memory segment desired.
- If accessing an existing shared memory segment, *size* can be 0.
- *flag* is the usual access permission bits with option of `IPC_CREAT` and `IPC_EXCL`.
- `shmctl()` is used to remove the structure created by the same *effective* user. Removal is immediate and anyone using the structure will get an error of `EIDRM`.

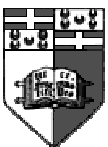
```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl(int shmid, IPC_RMID, NULL);
        return shm ID or -1 on error
```



# Unix Attaching and Detaching

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
void *shmat(int shmid, void *addr, int flag);
int shmdt(void *addr);
                                return -1 on error
```

- *shmat()* returns a pointer to the shared memory segment. This pointer resides in the process's user space.
- If *addr* is 0, system decides the return address, otherwise *addr* indicates desired address to attach shared memory segment.
- *flag* can be set to SHM\_RDONLY to set the shared memory segment as read only.
- *shmdt()* detaches the memory segment from user space. This does not delete the shared memory segment: *shmctl()* is used to delete resource.





# Exercises

- Use a semaphore to allow access to a common file between several processes.
- Create a shared memory segment and protect access to it using a semaphore.
- Use a shared memory segment to pass semaphore identifiers between different processes.

