

System calls for signals.

Signals are called “software interrupts.” One process can send signals to another process.

- Signal names: SIGINT, SIGKILL, SIGALRM, ...

- What to do with a signal system call?
 - Catch a signal: `signal(sig, func)`.
Provide a function that is called whenever a specific type of signal occurs. Need to re-enable signal catching.

 - Ignore a signal: `signal(sig, SIG_IGN)`.
All signal, other than SIGKILL can be ignored.

 - Allow the default to happen: `signal(sig, SIG_DEF)`.
Normally, a process is terminated when receive a signal.

- Send a signal: `kill(pid, sig)`.

Semaphore and shared memory system calls

1. Semaphore system calls

- Get a set of semaphores

- `semid = semget(key, nsems, permflags)`

key: a user defined name for the semaphore set

nsems: number of semaphores in the set

permflags: permission state (read, write)

semid: semaphore set identifier associated with key, used by other semaphore operations.

- Four ways to use it

- (a) Create a private semaphore set

`semid = semget(IPC_PRIVATE, nsems, 0600|IPC_CREAT|IPC_EXCL)`

Return a unique semid system wide, private to the process.

(b) Find key if already defined.

```
semid = semget(key, nsems, 0)
key ≠ IPC_PRIVATE, e.g. 0X200.
```

(c) Create only if key is not already defined

```
semid = semget(key, nsems,
0600|IPC_CREAT|IPC_EXCL)
If other processes specify the same
key, they will get the same semid.
```

(d) Find key if already defines, otherwise create

```
semid = semget(key, nsems, 0600|IPC_CREAT)
```

– SHELL commands for IPC:

`ipcs`: check ipc state

`ipcrm -s semid`: remove a semaphore.

- Semaphore control operations

`semval= semctl(semid, index, GETVAL, val)`

Get the value of the semaphore
index: index in the set, e.g. 0 means the first semaphore in the set.

`semval= semctl(semid, index, SETVAL, val)`

Set the semaphore value to val.

`pid= semctl(semid, 0, GETPID, val)`

Return the process id of the last process that performs an operation on the semaphore.

- Semaphore operations (up and down)

`semop(semid, op_array, somevalue)`

`op_array`: an array of semaphore operations to perform

`somevalue`: the number of semaphore operation records

`struct sembuf op_array[somevalue]` has three fields:

`sem_num`: index to semaphore in the set

`sem_op`: -1: down; +1: up

`sem_flag`: usually set to `SEM_UNDO`, automatically “undo” all operations after process exits.

Example:

```
sem_num= 0;
```

```
sem_op = -1;
```

```
sem_flag = SEM_UNDO
```

2. Shared memory system calls

- Create shared memory segment

```
shmid = shmget(key, size,  
0600|IPC_CREAT|IPC_EXCL)
```

size: number of bytes.

- Shared memory operations

```
shmat(shmid, dataptr, flag)
```

Attach the memory segment identified by shmid to process's logical data space

dataptr = 0: the segment is attached to the first available address selected by the system

dataptr nonzero: attach to user specified address, depending on flag:

- flag & SHM_RND is true. shmat will round dataptr to a page boundary
- flag & SHM_RND is false. attach to the exact values of dataptr
- flag & SHM_RDONLY is true. Read only.

Example:

```
struct databuf *pp;  
pp=(struct databuf *) shmat(shmid, 0,  
0);
```

- **Shared memory control operations**

`shmctl(shmid, command, &shm_stat)`

After you are done, remove the shared memory identifier specified by `shmid` from the system and destroy the shared memory segment and data structures associated to it:

`shmctl(shmid, IPC_RMID, (struct shm_id *)0)`