SysV semaphores

semget - I

```
int semid = semget(key_t key, int nsems, int flag);
```

key

- if key is set to IPC_PRIVATE, new semaphore is created
- if no semaphore corresponding to this key value exists and IPC_CREAT is asserted in flag, new semaphore is created
- otherwise, integer identifier for the existing semaphore corresponding to key is returned

nsems

- # of semaphores to be created
- ignored(?) if existing semaphore is being used

flag

- specifies permissions, whether semaphore is to be created, etc.
  Example: IPC_CREAT (create if needed), IPC_EXCL (ensure does not exist)
  Flags: S_IRUSR, S_IWGRP, S_IWOTH, S_IRWXU, etc.

semget – II

Programming issues:

- How to avoid using an existing semaphore that may be used by other processes?

  Ans: use `IPC_PRIVATE` as `key`
- How to ensure that the same semaphore is used by all cooperating processes within an application?

  Ans: create semaphore before forking all other processes required for the application
- Initialisation: use `semctl`

  From the man page: Although Linux, like many other implementations, initializes the semaphore values to 0, a portable application cannot rely on this.

Example use:

```
semid = semget(IPC_PRIVATE, n, IPC_CREAT|0600);
```

(`IPC_EXCL` not needed if `IPC_PRIVATE` specified)

semop – I

```
int semop(int semid, struct sembuf *ops, size_t nops);
```

`ops`

- array of semaphore operations

  ```
  struct sembuf {
    ushort sem_num; // which semaphore (0 .. nsems-1)
    short  sem_op;
    short  sem_flg; // specifies IPC_NOWAIT or SEM_UNDO
  }
  ```

- `sem_op > 0` add this value to `semval` (≡ unlocking / returning resources)
- `sem_op < 0` subtract this value from `semval` (≡ obtaining resources)
  - `IPC_NOWAIT` (no blocking) ⇒ return -1 with error code `EAGAIN`
  - otherwise, `semncnt` is incremented and process blocks until `semval >= |sem_op|`
- `sem_op == 0` wait till `semval == 0` (as above)

`semop` – II

- `sem_flg = SEM_UNDO` operation is undone when process terminates

`nops`

- # of operations in `ops` array

Important property

> The set of operations contained in `ops` is performed ***in array order***, and ***atomically***, that is, ***the operations are performed either as a complete unit, or not at all***.

System calls

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

semnum

- specifies which semaphore is to be operated on (`0 .. nsems-1`)

cmd

- `GETVAL` gets the value of `sem_base[semnum].semval`
- `SETVAL` sets the value of `sem_base[semnum].semval` to `arg.val`
- `GETALL`, `SETALL` operates on all semaphores using `arg.array`
- `IPC_STAT`, `IPC_RMID` (see man page for details)

arg

- specifies values used by various operations

```
union semun {
  int val;
  ushort *array;
  struct semid_ds *buf;     /* Buffer for IPC_STAT, IPC_SET */
}
```

has to be defined in calling program

Example application: file locking

Reference: Stevens(UNP) 3.10

- Locking operation: wait till semaphore is free (0); increment by 1
- Unlocking operation: decrement by 1

```
#define SEMKEY 123456L /* arbitrary key value for semget() */

static struct sembuf op_lock[2] = {
  /* semnum, sem_op, sem_flg */
       0,       0,      0,    /* wait for sem#0 to become 0 */
       0,       1,      0     /* then increment sem#0 by 1 */
};

static struct sembuf op_unlock[1] = {
  0, -1, IPC_NOWAIT           /* decrement sem#0 by 1 */
};
```

Example application: file locking

```
my_lock()
{
  if (semid < 0) {
  if ((semid = semget(SEMKEY, 1, IPC_CREAT | PERMS)) < 0)
      err_sys("semget error");
  }
  if (semop(semid, &op_lock[0], 2) < 0)
     err_sys("semop lock error");
}

my_unlock()
{
  if (semop(semid, &op_unlock[0], 1) < 0)
     err_sys("semop unlock error"); // "impossible"
}
```

Shared memory

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

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

void *shmat(int shmid, const void *shmaddr, int shmflg);
     /* shmaddr generally set to NULL */

int shmdt(const void *shmaddr); // DETACH (not delete)

int shmctl(int shmid, int cmd, struct shmid_ds *buf);
    /* cmd = IPC_STAT, IPC_RMID, etc. */
```

- Contents of shared memory are initialised to zero

Problems

- What happens if a process exits while holding a lock?
    - use `SEM_UNDO` with `semop()`
- How are semaphores initialized?
    - use `SETVAL` with `semctl()`
    - race conditions are possible

POSIX semaphores

Overview

```
#include <semaphore.h>

sem_t mutex;
```

- *Semaphores can only have non-negative values*
- Semaphores may be *named* or *unnamed*
  - *named semaphores* — like files (see under `/dev/shm`)
  - *unnamed semaphores* — like variables

System calls: semaphores (compiler flag -pthread)

```c
#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>

/* Named semaphores */
sem_t *sem_open(char *name, int oflag); // open existing semaphore
sem_t *sem_open(char *name, int oflag,  // create new semaphore
                mode_t mode, unsigned int initial_value);
int sem_close(sem_t *sem);
int sem_unlink(char *name);

/* For unnamed semaphores */
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_destroy(sem_t *sem);

int sem_wait(sem_t *sem);
int sem_trywait(sem_t *sem);
int sem_post(sem_t *sem);
int sem_getvalue(sem_t *sem, int *sval);
```

System calls: shared memory (compiler flag `-lrt`)

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>

int shm_open(char *name, int flag, mode_t mode);
int shm_unlink(char *name);
```

- `flag` : `O_CREAT`, `O_RDWR`, `O_RDONLY`, `O_EXCL`, etc.

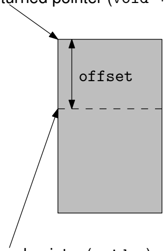
```
int ftruncate(int fd, off_t length);
```

- new segments are zero length
- use `ftruncate` to set length

Shared memory: attaching and detaching

```
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
/* addr - NULL
 * length - length of segment
 * prot - PROT_EXEC, PROT_READ, PROT_WRITE, PROT_NONE (no access!)
 * flags - MAP_SHARED, MAP_ANONYMOUS, ...
 * fd - file descriptor returned by shm_open()   (or an open file)
 * offset - usually 0 (see figure below)
 */
```

The returned pointer (`void *`) points here if `offset` is 0.



Chunk of shared memory obtained using `fd = shm_open(...)`

OR

Contents of a file opened using `fd = open(...)`
This use of `mmap()` was not discussed in class, so you may ignore it for now.

The returned pointer (`void *`) points here if `offset > 0`.

```
int munmap(void *addr, size_t length);
```

Example I

```c
#include "common.h"
#include <fcntl.h>           /* For O_* constants */
#include <sys/stat.h>        /* For mode constants */
#include <sys/types.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <time.h>

#define NUM 5

sem_t *lock; // Being lazy
```

Example II

```
int race_condition(int pid, char *filename)
{
    int count, i;
    FILE *fp;
    if (NULL == (fp = fopen(filename, "r+")))
        ERR_MESG("semaphores: error opening file for second time");
    for (i = 0; i < NUM; i++) {
        if (ERROR == sem_wait(lock))
            ERR_MESG("sem_wait failed");
        fscanf(fp, "%d", &count);
        rewind(fp);
        fprintf(stderr, "Value read by %d: %d\n", pid, count);
#ifndef DEBUG
        sleep(1);
#endif
```

Example III

```
        (pid) ? count++ : count-- ;

        fprintf(fp, "%d\n", count);
        rewind(fp);
        fprintf(stderr, "Value written by %d: %d\n", pid, count);
        if (ERROR == sem_post(lock))
            ERR_MESG("sem_post failed");
#ifndef DEBUG
        sleep(rand() % 2*NUM);
#endif
    }
    fclose(fp);

    return 0;
}
```

Example IV

```
int main(int ac, char *av[])
{
    char tmpfilename[BUF_LEN];
    int pid, status;
    FILE *fp;

    srand((int) time(NULL));

    strncpy(tmpfilename, "rcXXXXXX", BUF_LEN - 1);
    close(mkstemp(tmpfilename)); // V. lazy, don't do this!

    if (NULL == (lock = sem_open(tmpfilename, O_RDWR|O_CREAT, 0600, 1)))
        ERR_MESG("semaphores: error creating semaphore");

    if (NULL == (fp = fopen(av[1], "w")))
        ERR_MESG("semaphores: error opening file");
    fprintf(fp, "0\n");
```

Example V

```
    fclose(fp);

    if (ERROR == (pid = fork()))
        ERR_MESG("semaphores: fork failed");
    if (ERROR == race_condition(pid, av[1]))
        ERR_MESG("semaphores: error in race-condition() function");
    if (pid && // parent
        (ERROR == wait(&status) ||
         ERROR == sem_unlink(tmpfilename) ||
         ERROR == unlink(tmpfilename)))
        ERR_MESG("semaphores: error cleaning up");

    return 0;
}
```