SysV semaphores

Data structures

Reference: Stevens(APUE) 14.8

```
struct sem {
  ushort  semval;   // semaphore value
  ushort  semzcnt;  // # waiting for semval == 0
  ushort  semncnt;  // # waiting for increase in semval
  pid_t   sempid;   // process that did last op
};

struct semid_ds {
  ...
  struct sem *sem_base; // array of semaphores (kernel space)
  ushort sem_nsems;     // # of semaphores in the array
  /* timing information */
}
```

System calls

```
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
- 0 if existing semaphore is being used

`flag`

- specifies permissions, whether semaphore is to be created, etc.

System calls

```
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, SEM_UNDO, etc.
  }
  ```

- `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 (error)
  - otherwise, `semncnt` is incremented and process blocks until `semval >= |sem_op|`
- `sem_op == 0` wait till `semval == 0` (as above)

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`

`arg`

- specifies values used by various operations

```
union semun {
  int val;
  ushort *array;
  struct semid_ds *buf;
}
```

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 applications: 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"
}
```

Problems

- What happens if a process exits while holding a lock?
  - use `SEM_UNDO` with `semop()`
- What happens when all processes using a semaphore exit?
  - use `IPC_RMID` with `semctl()`
- 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*
- `sem_init(sem_t *sem, int pshared, unsigned int value);`
- `sem_wait(sem_t *sem)` : decrement if semaphore's value is greater than zero, otherwise block until it becomes possible to perform the decrement
- `sem_post` : increments semaphore and another process blocked in `sem_wait()` is woken up and proceeds to lock
- `sem_getvalue`
- `sem_destroy`