Synchronization

POSIX Semaphores

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Semaphore implementations

- There are several semaphores implementations
  - **Semaphores by means of a pipe**
  - **POSIX Pthread**
    - Condition variables
    - **Semaphores**
      - The most important
    - Mutex (for mutual exclusion)
    - ...
  - Linux semaphores
- Notice that semaphores are
  - Global share objects (see **sem_init**)
  - They are allocated by a thread, but they are kernel objects

System call: semget, semop, semctl (in sys/sem.h) they are complex to use

POSIX semaphores

- Kernel and OS independent system calls (POSIX)
  - Header file
    - #include <semaphore.h>
- A semaphore is a type sem_t variable
  - sem_t *sem1, *sem2, ...;
- All semaphore system calls
  - Have name sem_*
  - On error, they return the value -1

System calls:
sem_init
sem_wait
sem_trywait
sem_post
sem_getvalue
sem_destroy

sem_init ()

```
int sem_init (
   sem_t *sem,
   int pshared,
   unsigned int value
);
```

- Initializes the semaphore counter at value **value**
- The value **pshared** identifies the semaphore type
  - If equal to `0`, the semaphore is local to the **threads of the current process**
  - Otherwise, the semaphore can be **shared between different processes** (parent that initializes the semaphore and its children)

Linux does not currently support shared semaphores

sem_wait ()

```
int sem_wait (
   sem_t *sem
);
```

- Standard wait
  - If the semaphore is equal to 0, it blocks the caller until it can decrease the value of the semaphore

sem_trywait ()

```
int sem_trywait (
   sem_t *sem
);
```

- Non-blocking wait
  - If the semaphore counter has a value greater than 0, perform the decrement, and returns 0
  - If the semaphore is equal to 0, returns -1 (instead of blocking the caller as **sem_wait** does)

sem_post ()

```
int sem_post (
   sem_t *sem
);
```

- Standard signal
  - Increments the semaphore counter, or wakes up a blocked thread if present

sem_getvalue ()

```
int sem_getvalue (
   sem_t *sem,
   int *valP
);
```

Better not use this function. From Linux manual: "The value of the semaphore may already have changed by the time sem_getvalue() returns"

- Allows obtaining the value of the semaphore counter
  - The value is assigned to *valP
  - If there are waiting threads
    - 0 is assigned to *valP (Linux)
    - or a negative number whose absolute value is equal to the number of processes waiting (POSIX)

sem_destroy ()

```
int sem_destroy (
   sem_t *sem
);
```

- Destroys the semaphore at the address pointed by sem
  - Destroying a semaphore that other threads are currently blocked on produces undefined behavior (on error, -1 is returned)
  - Using a semaphore that has been destroyed produces undefined results, until the semaphore has been reinitialized

Example

Use of sem_* primitives to synchRonize threads

```
...
#include "semaphore.h"
...
sem_t *sem;
...
sem = (sem_t *) malloc(sizeof(sem_t));
sem_init (sem, 0, 1);
...
... create threads ...
...
sem_wait (sem);
... CS ...
sem_post (sem);
...
sem_destroy (sem);
```

The semaphore variable is allocated **dynamically**

Example

Use of sem_* primitives to synchRonize threads

```
...
#include "semaphore.h"
...
sem_t sem;
...
sem_init (&sem, 0, 1);
...
... create threads ...
...
sem_wait (&sem);
... CS ...
sem_post (&sem);
...
sem_destroy (&sem);
```

The semaphore variable is Allocated **statically**