Operating Systems: Practice: Lesson 3

Sevak Amirkhanian

What is errno?

errno is defined by the ISO C standard to be a modifiable lvalue of type *int*, and must not be explicitly declared; *errno* may be a macro. *errno* is thread-local; setting it in one thread does not affect its value in any other thread.

How is errno changed and how can we use it?

errno is always set by the last system call. It is the error code of this last system call if any error occurred. To access errno-associated message we use perror(..):

```
void perror(const char *s);
```

Sync primitives can be shared.

Synchronization primitives in POSIX like mutexes, semaphores and conditional variables can be shared across different processes.

How to make POSIX mutex shared?

Whether mutex is shared or not, should be determined with the mutex attribute. In order to make mutex shareable, we need to use the following POSIX interface:

```
int pthread_mutexattr_setpshared(
pthread_mutexattr_t *attr,
int pshared);
```

`PTHREAD_PROCESS_SHARED` should be passed as pshared value if we want the mutex to be accessible from other processes.

Semaphores

Semaphore is a another synchronization primitive which have similarities with conditional variables.

Semaphore is basically an integer counter which has 2 operations:
P() // increments the counter
V() // decrements the counter

The value of the semaphore is the number of units of the resource that are currently available

POSIX semaphores

POSIX semaphores have 4 main operations:

sem_init(...)

sem_post(..)

sem_wait(..)

sem_destroy(...)

https://man7.org/linux/man-pages/man7/sem_overview.7.html

Binary semaphores

If the semaphore count is 1, semaphore is essentially a mutex which is why in literature mutex is sometimes called *binary semaphore*.

sem_wait() -> pthread_mutex_lock()

sem_post() -> pthread_mutex_unlock()

Real-life examples of semaphores

Semaphores may be used in networking operating systems in routers for connection throttling.

Semaphores may also be used database drivers to restrict the amount of parallel connections, hence read-write operations.

Producer-Consumer Problem

The simple of description of the problem is:

We have a storage with limited storage with size N.

We have M producers who periodically produce a new item and insert into the storage if and only if there is a free space in the storage.

We have K consumers who periodically consume a new item from the storage if and only if there the storage is not empty.

Solution with semaphores

We will use 3 semaphores:

1 binary semaphore for lock

1 counting semaphore for full state

1 counting semaphore for empty state

Homework 3: Multiprocess sum calculation

Calculate the sum using multiple processes by forking the parent process.

The detailed description can be seen in: description.txt

The homework folder is called homework_3 in the zip, where you can see all the necessary files.

Thank you.