POSIX Semaphores

UNX511 Week 12 Class 1

John Sellens

July 29, 2025

Seneca Polytechnic

Outline

POSIX Semaphores

POSIX Semaphores

- A semaphore is an integer counter that is not allowed to go below 0
- There are atomic system calls that adjust the counter
- If a decrement would make the counter negative, the operation blocks or fails
- Consider it a counter of available (or unassigned) resources that consumers could take and work with
- Confession: I had trouble coming up with a good programming example that couldn't be handled in other ways
    - Semaphores were created in early operating system research

Conceptual Overview

- System starts up, controls access to a resource with a semaphore
    - e.g. Coordinate write access to a file
- Create the semaphore, set it to 1 (available)
- Process or thread wants to write to the file, so requests that the semaphore be decremented
    - If available, decrements to 0, carries on, then increments semaphore when finished
    - If not available, the decrement blocks until the process or thread that has access finishes, and increments the semaphore
- Multiple processes or threads can use the semaphore, and can be waiting for access (decrement to 0)

Semaphore Use Cases

- Control (lock) access to a file
    - Use a binary (0 or 1) semaphore to allow access by one at a time
- Indicate work or tasks to be done, consumers/workers can decrement to indicate they will take work
    - Needs some safe place to store the work and hand it out
    - Like a message queue maybe?
- I had trouble coming up with good simple practical examples

Unnamed vs Named Semaphores

- Unnamed semaphores are saved in memory locations
    - e.g. global variables, shared by threads or forked processes
    - or shared memory available to different processes
- Named semaphores appear in a global namespace
    - So unrelated processes could access the same semaphore
    - On Linux, appear under `/dev/shm`
- e.g. somewhat analogous to pipes and named pipes

Creation and Destruction

- Unnamed semaphores
  - declare a variable of type `sem_t`
  - call `sem_init(3)` with the variable and initial value
  - later call `sem_destroy(3)`
- Named semaphores
  - call `sem_open(3)` with flags and initial value, which returns a `sem_t`
  - later call `sem_close(3)`
  - and call `sem_unlink(3)` to remove the name

Semaphore Operations

- `sem_post(3)` increments the counter – indicates availability
- `sem_wait(3)` decrements the counter – requests access
    - also `sem_trywait(3)` and `sem_timedwait(3)`
- `sem_getvalue(3)` gets current semaphore value

Semaphores Code Samples

- Let's have a look in unx511_samples
    - `https://github.com/jsellens/unx511_samples`
- `week12_1/1_unnamed` – unnamed semaphore example
- `week12_1/2_named` – named semaphore example
- `week12_1/3_database` – network database, 3 clients
    - Possibly a little contrived – the clients prevent contention, normally a database service would handle that

References

- On Ubuntu, for the posix man pages:
  `sudo apt install manpages-posix-dev`
- `https://github-pages.senecapolytechnic.ca/unx511/Week12/Week12.html`
- The Linux Programming Interface book, chapter 53 “POSIX Semaphores”
- Wikipedia (which I found a little theoretical for my tastes):
  `https://en.wikipedia.org/wiki/Semaphore_(programming)`

Summary

- Semaphores provide another way to control access to resources
- Perhaps a little like a more general mutex
    - You can have multiple consumers if the semaphore count is greater than 1
- Are semaphores conceptually simpler?