Operating Systems Concepts

Implementing semaphores in xv6

CS 4375, Fall *2025*
Instructor: MD Armanuzzaman (***Arman***)
marmanuzzaman@utep.edu
November 24, 2025

Summery

- File Control Block
- Directories
  - Structure
  - Pathname Translation
  - Implementation
- Inode
- Soft Links and Hard Links

Agenda

- Revisit semaphores
- Overview of Homework 5
  - What we need to do
    - Semaphore system calls
    - One user progress that uses semaphores
  - Code walk through

Semaphores

- Synchronization tool for critical section problem
- Semaphore *S* - Integer variable
- Can only be accessed through two standard operations:
  - wait() and signal()
  - P() and V()
    - Proberen/test ; Verhogen/increase (in Dutch)
- Classical implementation (using busy-waiting): (Without interruption)

```
wait (S) {
  S--;
  while (S <= 0); // loop
}
```

```
signal (S) {
 S++;
}
```

Producer consumer Problem - Solution 4

- **Implementation 4** - 3 semaphore

Producer Process

```
int empty = N, full = 0;
while (true) {
     /* produce an item */
     wait(empty);
     wait(mutex);
     // add the item to the buffer
     signal(mutex);
     signal(full);
}
```

Consumer Process

```
while (true) {
     wait(full);
     wait(mutex);
     // remove item from buffer
     signal(mutex);
     signal(empty);
     /* consume the item */
}
```

Pull from given branch

- **Branch**: hw5-init
  - mmap system call
    - creates a new mapping in the virtual address space of the calling process
  - munmap system call
    - deletes the mappings for the specified address range
  - private user program
    - `$ private`

      `total = 55`

Implementation of producer consumer problem

- `prodcons-sem.c`
  - semaphore occupied to synchronize number of ***full*** slots
  - semaphore free to synchronize number of ***empty*** slots
  - mutex lock to ensure ***mutual exclusion*** of the critical region
  - mmap to allocate virtual memory
  - munmap to destroy the allocated memory
  - producer() executed by new process with fork
  - consumer() executed by new process with fork

Implementing semaphores

- `sem_init()`
  - Initialize the `semtab` array for the kernel
- `sem_destroy()`
  - Free semaphore slot from `semtab`
- `sem_wait()`
  - P() operation of semaphores
  - Put the process to sleep *(AVOID BUSY WAITING)*: *sleep(s, s->lock)*
- `sem_post()`
  - V() operation of semaphores *(WAKE UP SLEEPING P)*: *wakeup(s)*

Data Structures

```
// Counting semaphore
struct semaphore {
        struct spinlock lock; // semaphore lock
        int count; // semaphore value
        int valid; // 1 if this entry is in use
};
// OS semaphore table type
struct semtab {
        struct spinlock lock;
        struct semaphore sem[NSEM];
};
extern struct semtab semtable;
```

Helper functions

- `semaphore.c`
  - `semalloc()`
    - *allocate a free semaphore slot; return index or -1 if none*
  - `semdealloc(indx)`
    - *free a semaphore slot (index from semalloc / sem_t)*

Readers and Writers Problem

Reader process

```
while (true) {
     wait(mutex);
     readercount++;
     if (readercount == 1)
          wait (wrt_mutex);
     signal(mutex);
     /* read from database */
     wait(mutex);
     readercount--;
     if (readercount == 0)
          signal (wrt_mutex);
     signal(mutex);
}
```

Writer Process

```
while (true) {
     wait(wrt_mutex);
     /* write to database */
     signal(wrt_mutex);
}
```

Implementing Readers–Writers user program

- reader()
  - Each reader should read -> `READER_ITERS 50`
  - Critical section
    - `printf("reader %d sees value %d\n", getpid(), v);`
- writer()
  - Each writer should write -> `WRITER_ITERS 100`
  - Critical section:
    - `rw->value++;`
    - `printf("writer %d updated value to %d\n", getpid(), rw->v);`

Announcement

- Course evaluation
  - Timeline: **11/24/25 - 12/07/25**
  - **20 MARKS BONUS** IF WE HAVE **90%** OF THE CLASS SUBMISSION
- Next week review classes
  - Questions highly encouraged
  - Which topics should I revisit?