Semaphore

CIT 595
Spring 2010

Semaphore

- Restricts the number of simultaneous users of a shared resource up to a maximum number
- Threads can request access to the resource (decrementing the semaphore), and can signal that they have finished using the resource (incrementing the semaphore)
- Semaphore vs. Mutex
  - Mutex: a key to a room
    - One person can have the key, use it, when done gives up the key for next person to use
  - Semaphore: Number of free identical room keys
    - 4 keys – Semaphore count starts at 4, count decremented as people come for a room
    - if full then no keys left, count is 0, then the person has to wait
    - If one person gives up the key, semaphore is increased by 1 and waiting person is given the room

Semaphores

- A semaphore is an object with an integer value (having some initial value)
- It supports two operations:
  - P – If the semaphore's value is greater than zero, decrement. Otherwise, wait until the value is greater than zero and then decrement
  - V – Increment the value of the semaphore
- Semaphores were invented by Edsger Dijkstra
  - V stands for "verhoog" (increase)
  - P stands for "probeer te verlagen" (try and decrease)
- Both P and V have to be atomic operations

Semaphore API (semaphore.h)

- int sem_wait(sem_t *sem)
  - P action
  - blocks until the semaphore count pointed to by sem is greater than zero and then atomically decrements the count
- int sem_post(sem_t *sem)
  - V action
  - Atomically increments the count of the semaphore pointed to by sem. If there are any threads blocked on the semaphore, one will be unblocked
- int sem_init(sem_t *sem, int pshared, unsigned int value)
  - Initialize the semaphore to a value
  - If pshared is 0 then, semphamore is shared between threads of the process
  - else shared between processes
Producer/Consumer with semaphore

- Requirements
  - Consumer must wait for producer to fill buffer
  - Producer must wait for consumer to empty buffer (if filled)

- Requires 2 semaphores (type: sem_t)
  - emptyBuffer, fullBuffer (just like we need two con_t variables in thread3.c)

Producer

while (1) {
    sem_wait(&emptyBuffer);
    fill(&buffer);
    sem_post(&fullBuffer);
}

Consumer

while (1) {
    sem_wait(&fullBuffer);
    use(&buffer);
    sem_post(&emptyBuffer);
}

More Motivation for Semaphores

- Locks only provide mutual exclusion
  - Ensure only one thread is in critical section at a time
  - E.g. pthread_mutex_lock(&myMutex)
  - pthread_mutex_unlock(&myMutex)

- May want more: Place ordering on scheduling of threads
  - Readers/ Writers problem
  - E.g. Airline Database, many readers and writers
  - Ok to have many multiple readers at the same time
  - But if writer comes along, then no one should access the database
    - Writer must wait for readers currently reading to finish
    - Problem: Writer could wait indefinitely as while a reader is still present other readers could also come and read
    - Sol: Give writer priority
      - Any readers after the writer are blocked
      - This writer only waits for active readers

Readers/Writer with Readers having priority (Psuedo code)

<table>
<thead>
<tr>
<th>Reader</th>
<th>Writer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader</td>
<td></td>
</tr>
</tbody>
</table>
| while (1) {
  wait(rc_s);
  readcount++;
  if (readcount==1) {
    wait(db);
    signal(rc_s);
    doReading();
  } |
| | |
| Writer | |
| while (1) {
  wait(db);
  doWriting();
  signal(db);
} |

Comments on R/W w/ reader priority

- Reader:
  - Needs mutual exclusive access while manipulating readers variable
  - Does not need mutually exclusive while reading the database
  - If this reader is the first reader it has to wait if there is an active writer
    - Writer has exclusive access to the database
  - If other readers come along while the first one is waiting they wait at the statement wait(rc_s)
  - If other readers come along while the first is actively reading the database, they can also get in to read the database
  - When the last reader finishes, if there are any waiting writers, it must wake up one
Comments on R/W w/ reader priority

- If there is an active writer, this writer has to wait
  - Active writer has exclusive access to the database

- If there are active readers, this writer has to wait (reader have priority)
  - Because the first reader did wait(db)

- The write only gets in to write to the database when there are no other active readers or writers

- When the writer finishes, it wakes up someone (either a reader/write)

- If a reader gets to go next, then once it through statement wait(rc_s) and starts reading the database, then all other readers waiting also read