CS 4410
Operating Systems

Synchronization
Classic Problems

Summer 2016
Cornell University
Today

- Producer-Consumer Problem
- Bounded-Buffer Problem
Restrictions on accessing shared data

• For a multithreaded process to be correct, some restrictions should be applied to when thread access shared data.
• Semaphores can model these restrictions.
• We see how semaphores can model different kinds of restrictions in two different problems.
Producer-Consumer Problem

- One bounded buffer with N entries.
- Multiple producer-threads: fill buffer’s entries.
  - Pointer $\text{In}$ shows the next entry to be filled.
  - Each producer fills the entry pointed by $\text{In}$, advances $\text{In}$ to point to the next entry.
- Multiple consumer-threads: empty buffer’s entries.
  - Pointer $\text{Out}$ shows the next entry to be emptied.
  - Each consumer empties the entry pointed by $\text{Out}$, advances $\text{out}$ to point to the next entry.
Producer-Consumer Problem

- Shared data between all threads: buffer.
- Shared data between producers: In.
- Shared data between consumers: Out.
- Requirements on shared data:
  - Only one thread should modify the buffer at any time.
  - No production when all N entries are full.
  - No consumption when no entry is full.
Selecting semaphores for satisfying restrictions

• Requirements on shared data:
  1. Only one thread should modify the buffer at any time.
  2. No production when all N entries are full.
  3. No consumption when no entry is full.

• Semaphores on shared data:
  1. Mutex
  2. Counter semaphore initialized at N.
  3. Counter semaphore initialized at 0.
Producer-Consumer Problem

Shared data: buffer, “In”, “Out”
Shared Semaphores: mutex, empty, full;

mutex = 1; /* for mutual exclusion*/
empty = N; /* number empty buf entries */
full = 0; /* number full buf entries */

**Producer**
do {
   //produce item
   //update “In”
} while (true);  

**Consumer**
do {
   //consume item
   //update “Out”
} while (true);
Producer-Consumer Problem

Shared data: buffer, “In”, “Out”

Shared Semaphores: mutex, empty, full;

mutex = 1; /* for mutual exclusion*/
empty = N; /* number empty buf entries */
full = 0; /* number full buf entries */

**Producer**
do {
P(empty);
//produce item
//update “In”
V(full);
} while (true);

**Consumer**
do {
P(full);
//consume item
//update “Out”
V(empty);
} while (true);
Producer-Consumer Problem

Shared data: buffer, “In”, “Out”

Shared Semaphores: mutex, empty, full;

mutex = 1; /* for mutual exclusion*/
empty = N; /* number empty buf entries */
full = 0; /* number full buf entries */

**Producer**
do {
P(empty);
P(mutex);
//produce item
//update “In”
V(mutex);
V(full);
} while (true);

**Consumer**
do {
P(full);
P(mutex);
//consume item
//update “Out”
V(mutex);
V(Empty);
} while (true);
Readers-Writers Problem

- One file.
- Many reader-threads: read data from the file.
- Many writer-threads: write data to the file.
Readers-Writers Problem

- Shared data between all threads: file.
- Requirement:
  - At any point of time, the file may be accessed only by one writer or by multiple readers.

Need some additional state to count the active readers.
Readers-Writers Problem

- Shared data between all threads: file.
- Shared data between readers: readcount.
- Requirements:
  - At any time, the file may be accessed only by one writer or by multiple readers.
  - At any time, readcount may be accessed by one reader.
Selecting semaphores for satisfying restrictions

• Requirements on shared data:
  1. At any time, the file may be accessed only by one writer or by multiple readers.
  2. At any time, readcount may be accessed by one reader.

• Semaphores on shared data:
  1. Mutex
  2. Mutex
Readers-Writers Problem

mutex = Semaphore(1)
wrt = Semaphore(1)
readcount = 0;

**Writer**
do{
    /*writing is performed*/
}
}while(true)

**Reader**
do{
    /*reading is performed*/
    readcount = readcount + 1;
}
}while(true)
Readers-Writers Problem

mutex = Semaphore(1)

wrt = Semaphore(1)

readcount = 0;

**Writer**
do{
  P(wrt);
  /*writing is performed*/
  V(wrt);
}while(true)

**Reader**
do{
  P(wrt);
  /*reading is performed*/
  V(wrt);
}while(true)
Readers-Writers Problem

mutex = Semaphore(1)
wrt = Semaphore(1)
readcount = 0;

**Writer**
do{
    P(wrt);
    /*writing is performed*/
    V(wrt);
}while(true)

**Reader**
do{
P(mutex);
    readcount++;
    if (readcount == 1)
P(wrt);
V(mutex);
    /*reading is performed*/
P(mutex);
    readcount--;
    if (readcount == 0)
    V(wrt);
    V(mutex);
}while(true)
Today

- Producer-Consumer Problem
- Bounded-Buffer Problem
Coming up...

- Next lecture: monitors
- HW2: all exercises except for repair.py can be solved