Semaphore Use In Synchronization

Ch 6 [BenA 06]

Consumer Producer Revisited
Readers and Writers
Baton Passing
Private Semaphores
Resource Management

Semaphore Use In Synchronization

Process Q(i = 1 to 4)

V(gate); # signal Q

Process R(i = 1 to 4)

V(gate); # signal Q

Baton Passing

Private Semaphores

Resource Management

Barrier Synchronization with Semaphores

Barrier is implemented as separate process
– This is just one possibility to implement the barrier
– Cost of process switches?
– How many process switches?

Barrier Synchronization with Barrier OS-Primitive

Specific synchronization primitive in OS
– Implemented with semaphores…
– No need for extra process – less process switches

Communication with Semaphores

Sem mutex = 1, data_ready = 0;
Int buffer: # one data item buffer

Process W

P(mutex)

write_buffer(data)

V(mutex)

V(data_ready); # signal Q

Process R

P(mutex)

read_buffer(data)

V(mutex)

V(data_ready); # signal Q

• What is wrong?

W might rewrite data buffer before R reads it
– Might have extra knowledge that avoids the problem

Communication with Semaphores Correctly

Sem mutex = 1, data_ready = 0, buffer_empty = 1;
Int buffer

Process W

P(buffer_empty);
write_buffer(data)
V(mutex)

Process R

P(data_ready);
write_buffer(data)
V(mutex)

• Fast W can not overtake R now
• One reader R, one writer W, binary semaphores
• Actual communication with buffer in shared memory
– Use model: 1 producer – 1 consumer – size 1 buffer
Producer-Consumer with Binary Semaphores

- Binary semaphore has values 0 and 1
  - OS or programming language library
- Semaphore does not keep count
  - Must have own variable count (nr of elements in buffer)
    - Protect it with critical section
- Two important state changes
  - Empty buffer becomes not empty
    - Consumer may need to be awakened
  - Full buffer becomes not full
    - Producer may need to be awakened

Simple Solution #1

```c
typeT buf[n]; /* n element buffer */
int front=0; /* read from here */
rear=0; /* write to this one */
count=0; /* nr of items in buf */
sem space=1; /* need this to write */
items=0; /* need this to read */
mutex=1; /* need this to update count */
```

```c
process Producer [i=1 to M] {
    while(true) {
        ... produce data …
        P(space); /* wait until space to write*/
        P(mutex);
        buf[rear] = data; rear = (rear+1) %n; /* outside mutex, ok? */
        count++;
        if (count == 1) V(items); /* first item to empty buffer */
        if (count < n) V(space); /* still room for next producer */
        V(mutex);
    }
}

process Consumer [i=1 to N] {
    while(true) {
        P(items); /* wait until items to consume */
        data=buf[front]; front = (front+1) %n;
        P(mutex);
        count--; /* all of this must be in mutex */
        if (count == n-1) V(space); /* buffer was full */
        if (count > 0) V(items); /* still items for next consumer */
        V(mutex);
        ... consume data …
    }
}
```

Evaluate Solution #1

- Simple solution
  - Mutex and synchronization ok
  - Mutex inside space or items
  - Get space first and then mutex
  - Buffer reserved for one producer/consumer at a time
    - Does not allow for simultaneous buffer use
      - Producer inserts item to “rear”
      - Consumer removes item from “front”
  - First waiting producer/consumer advances when signalled
    - Queued in semaphores

Not good
Simultaneously?

Better Solution #2

```c
typeT buf[n]; /* n element buffer */
int front=0; /* read from here */
rear=0; /* write to this one */
count=0; /* nr of items in buf */
sem space=1; /* need this to write */
items=0; /* need this to read */
mutex=1; /* need this to update count */
```

```c
process Producer [i=1 to M] {
    while(true) {
        ... produce data …
        P(space); /* wait until space to write*/
        buf[rear] = data; rear = (rear+1) %n; /* outside mutex, ok? */
        P(mutex);
        count++;
        if (count == 1) V(items); /* first item to empty buffer */
        if (count < n) V(space); /* still room for next producer */
        V(mutex);
    }
}

process Consumer [i=1 to N] {
    while(true) {
        P(items); /* wait until items to consume */
        data=buf[front]; front = (front+1) %n;
        P(mutex);
        count--; /* all of this must be in mutex */
        if (count == n-1) V(space); /* buffer was full */
        if (count > 0) V(items); /* still items for next consumer */
        V(mutex);
        ... consume data …
    }
}
```
Evaluate Solution #2
- Relatively simple solution
  - Data copying (insert, remove) outside critical section
  - Simultaneous insert and remove ops
    - Producer inserts item to “rear”
    - Consumer removes item from “front”
  - First waiting producer/consumer advances when signalled
    - Queued in semaphores

Another Solution #3
- Use condition synchronization
  - Do P(space) or P(items) only when needed
  - Expensive op?
  - Requires execution state change (kernel/user)?

Solutions #1, #2, and #3
- Which one is best? Why? When?
- How to maximise concurrency?
  - Separate data transfer (insert, remove) from permission
to do it
  - Allow obtaining permission
    (e.g., code with P(space) and updating count)
    for one process run concurrently with data transfer
    for another process
  - Need new mutexes to protect data transfers and
    index (rear, front) manipulation
- Problem: signalling to other producers/consumers
  should happen in same critical section with updating
count, but should happen only after data transfer is
completed (i.e., in different critical section …)
Readers and Writers Problem
- Shared data structure or data base
- Two types of users: readers and writers
  - Readers
    - Many can read at the same time
    - Can not write when someone reads
    - Can not read when someone writes
  - Writers
    - Read and modify data
    - Only one can be active at the same time
    - Can be active only when there are no readers

Readers and Writers with Baton
Passing Split Binary Semaphores
- Component semaphores e, r, w
  - Mutex wait in P(e), initially 1
  - Readers wait in P(r) if needed, initially 0
  - Writers wait in P(w) if needed, initially 0
- In critical control areas only one process advances at a time
  - Wait in e, r, or w
  - One advances, others wait in e, r or w
  - When done, pass the baton (turn) to next one

Baton passing
- When done your own mutex zone, wake up next ...
  (one or more semaphores control the same mutex)

- If reader waiting and no writers: V(r)
  - Do not release mutex (currently reserved e, r, or w)
  - New reader will continue with mutex already locked
- If writer waiting and no readers: V(w)
  - Do not release mutex, pass baton to writer
- Else (let new process to compete with old ones): V(e)
  - Release mutex to let new process in the game
  - New process gets in mutex only when no old one can be advance

- If writer waiting and no readers: V(w)
  - Do not release mutex, pass baton to writer
  - Else (let new process to compete with old ones): V(e)
  - Release mutex to let new process in the game
  - New process gets in mutex only when no old one can be advance
  - Can happen concurrently when reading database

- Else if writer waiting and no readers: V(w)
  - Do not release mutex, pass baton to writer
  - Else (let new process to compete with old ones): V(e)
  - Release mutex to let new process in the game
  - New process gets in mutex only when no old one can be advance
  - Can happen concurrently when reading database

Readers and Writers with Baton
Passing Split Binary Semaphores
- Component semaphores e, r, w
  - Mutex wait in P(e), initially 1
  - Readers wait in P(r) if needed, initially 0
  - Writers wait in P(w) if needed, initially 0
- In critical control areas only one process advances at a time
  - Wait in e, r, or w
  - One advances, others wait in e, r or w
  - When done, pass the baton (turn) to next one

Baton passing
- When done your own mutex zone, wake up next ...
  (one or more semaphores control the same mutex)

- If reader waiting and no writers: V(r)
  - Do not release mutex (currently reserved e, r, or w)
  - New reader will continue with mutex already locked
- If writer waiting and no readers: V(w)
  - Do not release mutex, pass baton to writer
- Else (let new process to compete with old ones): V(e)
  - Release mutex to let new process in the game
  - New process gets in mutex only when no old one can be advance

- Else if writer waiting and no readers: V(w)
  - Do not release mutex, pass baton to writer
  - Else (let new process to compete with old ones): V(e)
  - Release mutex to let new process in the game
  - New process gets in mutex only when no old one can be advance
  - Can happen concurrently when reading database
Concurrent Programming (RIO) 7.2.2011

Lecture 7: Readers and Writers

Baton Passing with SIGNAL

SIGNAL – CS baton passing, priority on readers

- if (nw == 0 and dr > 0) {
  dr = dr -1;
  V(r); # wake up waiting reader
}
- else if (nr == 0 and nw == 0 and dw > 0) {
  dw = dw -1;
  V(w); # wake up waiting writer
}
- else
  V(e); # let new process to mix

Still readers first

Unnecessary parts of SIGNAL code was removed

Modify to give writers priority?

Resource Management

- Problem
  - Many types of resources
  - N units of given resource
  - Request allocation: K units
    - Wait suspended until resource available

- Solution
  - Semaphore mutex (init 1)
  - Semaphore Xavail
    - init N – wait for available resource
    - init 0 – wait for permission to continue

Simple Very Bad Solution

- What is wrong?
  - everything
  - Mutex?
  - Deadlock?
  - Unnecessary delays?

- Each P() may result in (long) delay?
- Hold mutex while waiting for resource
- Others can not get mutex to release resources…

Another Not So Good Solution

- What is wrong?
  - Works only for resources allocated and freed
  - Mutex?
  - Mutex of control data?
  - Mutex of resource allocation data structures?

Resource Management with Baton Passing Split Semaphore

- BATON PASSING – CS
- (short wait) (long wait) = If ok?
  yes.
Problems with Resource Management

- Need strong semaphores
- Strong semaphores are FIFO
  - What if 1\textsuperscript{st} in line want 6 units, 2\textsuperscript{nd} wants 3 units, and there are 4 units left?
  - What about priorities?
    - Each priority class has its own semaphore
    - Baton passing within each priority class?
  - How to release just some specific process?
    - Strong semaphore releases 1\textsuperscript{st} in line
    - Answer: private semaphores

Private Semaphore

- Semaphore, to which only one process can ever make a P-operation
  - Initialized to 0, belongs to that process
- Usually part of PCB (process control block) for the process
  - Can create own semaphore array for this purpose
- Process makes demands, and then waits in private semaphore for turn
- Most often just one process at a time
  - Usually P(mutex) does not lead to process switches
- Usually still need to wait in private semaphore

Private Semaphore Use Example

- Common resource allocation method
  - Here: time = amount of resource requested
  - Here: just select next job (with shortest time)
  - Here: just one job (at most) holding the resource at a time
- Use private semaphores

  \[
  \text{P(e):} \quad \text{if} \, \text{free} = \text{false} \quad \text{DELAY; } \text{free} = \text{false; } \text{V(e);}
  \]

  \[
  \text{release():} \quad \text{P(e); } \text{free} = \text{true; } \text{SIGNAL;}
  \]

PAIRS:

<table>
<thead>
<tr>
<th>ID</th>
<th>P2</th>
<th>P15</th>
<th>P3</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
<td>17</td>
<td>64</td>
</tr>
</tbody>
</table>

Queue can be ordered according to requested cpu-time
(requested cpu-time is the resource in this example)

b(n)

Private semaphore b[ID] for each process ID: 0..n-1

Process release is dependent on its location in PAIRS.
When resource becomes free, the 1\textsuperscript{st} process in line may advance.
Semaphore Feature Summary

- Many implementations and semantics
  - Be careful to use
  - E.g., is the (process) scheduler called after each V()?
    - Which one continues with processor, the process executing V() or the process just woken up?
    - Can critical section continue after V()?
  - Busy wait vs. suspend state?

- Hand coded synchronization solutions
  - Can solve almost any synchronization problem
  - Baton passing is useful and tricky
    - Explicit mutex handover of some type of resource use
  - Private semaphores
    - Explicit signal to some specific process
  - Be careful to use
    - Do not leave mutex'es open, do not suspend inside mutex
    - Avoid deadlocks, do (multiple) P's and V's in correct order

Copyright Teemu Kerola 2011