More concurrency and deadlock

10/12/15
Announcements

• About to start networking. Please read from the TCP/IP text:
  – Chapter 1 for Wednesday
  – Chapter 2 for Friday
  – Section 3.1 for Monday
Recall: Readers and writers

- Processes share a common database
- Some want read access (readers) while others want ability to write (writers)
- Readers should be able to share the database, but all other processes must block if a writer gets access
Recall: Solving readers and writers

semaphore mutex = 1;  //control access to database
int numR = 0;         //number of active readers

void read() {
    numR++;
    if(numR == 1) down(mutex);
    //perform read
    numR--;
    if(numR == 0) up(mutex);
}

void write() {
    down(mutex);
    //perform write
    up(mutex);
}

Does this successfully implement readers and writers?

A. Yes.
B. I sure hope so
C. No. It allows deadlock
D. No. It creates some other problem
E. It’s Friday and I can’t think this hard anymore
Solving readers and writers

semaphore mutex = 1, num_mutex = 1;  //mutex protects database, num_mutex protects numR
int numR = 0;  //number of active readers

void read() {
    down(num_mutex);
    numR++;
    if(numR == 1) down(mutex);
    up(num_mutex);
    //perform read
    down(num_mutex);
    numR--;
    if(numR == 0) up(mutex);
    up(num_mutex);
}

void write() {
    down(mutex);
    //perform write
    up(mutex);
}

Does this successfully implement readers and writers?
A. Yes
B. I sure hope so
C. No. It allows deadlock
D. No. It creates some other problem
E. I give up. This problem can’t be solved
Solving readers and writers

Semaphore mutex = 1, num_mutex = 1; //mutex protects database, num_mutex protects numR
int numR = 0; //number of active readers

void read() {
    down(num_mutex);
    numR++;
    if(numR == 1) down(mutex);
    up(num_mutex);
    //perform read
    down(num_mutex);
    numR--;
    if(numR == 0) up(mutex);
    up(num_mutex);
}

void write() {
    down(mutex);
    //perform write
    up(mutex);
}

Does this successfully implement readers and writers?
A. Yes
B. I sure hope so
C. No. It allows deadlock
D. No. It creates some other problem
E. I give up. This problem can’t be solved

(but does privilege readers since they never have to give up the database)
Deadlock in detail

Next time: Networking with nuance!
Review: Deadlock

• Situation in which group of threads/processes all block forever
• Typically, each holds a resource that others are blocking on
Modeling resource contention deadlocks

- Focus on requesting and freeing resources
- Assume process blocks if requests something in use
- Vertices for each process and each resource
- Edge from resource to process holding it, from process to resource it is blocking on

Example:

<table>
<thead>
<tr>
<th>Process A</th>
<th>Process B</th>
<th>Process C</th>
</tr>
</thead>
<tbody>
<tr>
<td>request R</td>
<td>request S</td>
<td>request T</td>
</tr>
<tr>
<td>request S</td>
<td>request T</td>
<td>request R</td>
</tr>
<tr>
<td>free R</td>
<td>free S</td>
<td>free T</td>
</tr>
<tr>
<td>free S</td>
<td>free T</td>
<td>free R</td>
</tr>
</tbody>
</table>
Detecting deadlock

• Use DFS to look for cycle in graph
• Vertex colors: “unvisited”, “in progress”, “done”

all vertices initially “unvisited”
for each “unvisited” vertex v: visit(v);

visit(vertex v):
    color v “in progress”;
    for each neighbor u of v:
        if(u is “in progress”) print “found cycle”;
        if(u is “unvisited”) visit(u);
    color v “done”;
Detection strategy w/ multiple copies of resources

(using vectors and matrices...)

Is the following situation safe?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A = 2 1 1

---

A. Yes
B. Yes
C. No
D. No
E. As long as you don’t call on me to explain my answer
Is the following situation safe?

A. Yes
B. Yes
C. No
D. No
E. As long as you don’t call on me to explain my answer
Consider the “Ostrich Algorithm” for dealing with deadlocks, which is to ignore the possibility and reboot if they occur. Which of the following is true about this strategy?

A. It’s cheap and easy to implement
B. How common could deadlocks be anyway? Enjoy life and don’t be a worrywart!
C. Ostriches are cute so it must be a good strategy
D. It is far too irresponsible to use on production systems
E. Not exactly one of the above
Consider the “Ostrich Algorithm” for dealing with deadlocks, which is to ignore the possibility and reboot if they occur. Which of the following is true about this strategy?

A. It’s cheap and easy to implement
B. How common could deadlocks be anyway? Enjoy life and don’t be a worrywart!
C. Ostriches are cute so it must be a good strategy
D. It is far too irresponsible to use on production systems
E. Not exactly one of the above (A & B)