COS 318: Operating Systems

Deadlocks

Kai Li and Andy Bavier
Computer Science Department
Princeton University

http://www.cs.princeton.edu/courses/archive/fall13/cos318/
Today’s Topics

- Finish CPU scheduling algorithms
- Conditions for a deadlock
- Strategies to deal with deadlocks
4.3 BSD Scheduling with Multi-Queue

- "1 sec" preemption
  - Preempt if a process doesn’t block or complete within 1 second

- Priority is recomputed every second
  - $P_i = \text{base} + \frac{\text{CPU}_i - 1}{2} + \text{nice}$, where $\text{CPU}_i = \frac{\text{U}_i + \text{CPU}_{i-1}}{2}$
  - Base is the base priority of the process
  - $\text{U}_i$ is process utilization in interval $i$

- Priorities
  - Swapper
  - Block I/O device control
  - File operations
  - Character I/O device control
  - User processes
Linux Scheduling

- **Time-sharing scheduling**
  - Two priority arrays: active and expired
  - 40 priority levels, lower number = higher priority
  - Priority = base (user-set) priority + “bonus”
    - Bonus between -5 and +5, derived from `sleep_avg`
    - Bonus decremented when task sleeps, incremented when it runs
    - Higher priority gets longer timeslice
  - Move process with expired quantum from active to expired
  - When active array empty, swap active and expired arrays

- **Real-time scheduling**
  - 100 static priorities, higher than time sharing priorities
  - Soft real-time
Windows Scheduling

- Classes and priorities
  - Real time: 16 static priorities
  - User: 16 variable priorities, start at a base priority
    - If a process has used up its quantum, lower its priority
    - If a process waits for an I/O event, raise its priority

- Priority-driven scheduler
  - For real-time class, do round robin within each priority
  - For user class, do multiple queue

- Multiprocessor scheduling
  - For N processors, normally run N highest priority threads
  - Threads have hard or soft affinity for specific processors
  - A thread will wait for processors in its affinity set, if there are other threads available (for variable priorities)
Today’s Topics

- Finish CPU scheduling algorithms
- Conditions for a deadlock
- Strategies to deal with deadlocks
Definitions

- Use processes and threads interchangeably
- Resources
  - Preemptable: CPU (can be taken away)
  - Non-preemptable: Disk, files, mutex, ... (can’t be taken away)
- Use a resource
  - Request, Use, Release
- Starvation
  - Processes wait indefinitely
- Deadlocks
  - A set of processes have a deadlock if each process is waiting for an event that only another process in the set can cause
Resource Allocation Graph

- Process A is holding resource R
- Process B requests resource S
- A cycle in resource allocation graph $\Rightarrow$ deadlock
- If A requests for S while holding R, and B requests for R while holding S, then

How do you deal with multiple instances of a resource?
Non-Resource Deadlock

Guns don’t cause deadlocks – people do
An Example

- A utility program
  - Copy a file from tape to disk
  - Print the file to printer

- Resources
  - Tape
  - Disk
  - Printer

- A deadlock
  - A holds tape and disk, then requests for a printer
  - B holds printer, then requests for tape and disk
Conditions for Deadlock

- **Mutual exclusion condition**
  - Each resource is assigned to exactly one process
- **Hold and Wait**
  - Processes holding resources can request new resources
- **No preemption**
  - Resources cannot be taken away
- **Circular chain of requests**
  - One process waits for another in a circular fashion

**Question**
- Are all conditions necessary?
Eliminate Competition for Resources?

- If running A to completion and then running B, there will be no deadlock.

- Generalize this idea for all processes?

- Is it a good idea to develop a CPU scheduling algorithm that causes no deadlock?

Previous example
Strategies

- Ignore the problem
  - It is user’s fault
- Detection and recovery
  - Fix the problem afterwards
- Dynamic avoidance
  - Careful allocation
- Prevention
  - Negate one of the four conditions
Ignore the Problem

- The OS kernel locks up
  - Reboot

- Device driver locks up
  - Remove the device
  - Restart

- An application hangs ("not responding")
  - Kill the application and restart
  - Familiar with this?

- An application ran for a while and then hang
  - Checkpoint the application
  - Change the environment (reboot OS)
  - Restart from the previous checkpoint
Detection and Recovery

- **Detection**
  - Scan resource graph
  - Detect cycles

- **Recovery (difficult)**
  - Kill process/threads (can you always do this?)
  - Roll back actions of deadlocked threads

- What about the tape-disk-printer example?
Avoidance

Safety Condition:
- It is not deadlocked
- There is some scheduling order in which every process can run to completion (even if all request their max resources)

Banker’s algorithm (Dijkstra 65)
- Single resource
  - Each process has a credit
  - Total resources may not satisfy all credits
  - Track resources assigned and needed
  - Check on each allocation for safety
- Multiple resources
  - Two matrices: allocated and needed
  - See textbook for details
Examples (Single Resource)

Total: 8

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>P_2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>P_3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Free: 1

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>P_2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>P_3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Free: 0

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>P_2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Free: 3

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>P_2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_3</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Free: 1

<table>
<thead>
<tr>
<th></th>
<th>Has</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>P_2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Free: 6

Free: 1

?
Prevention: Avoid Mutual Exclusion

- Some resources are not physically sharable
  - Printer, tape, etc
- Some can be made sharable
  - Read-only files, memory, etc
  - Read/write locks
- Some can be virtualized by spooling
  - Use storage to virtualize a resource into multiple resources
  - Use a queue to schedule
  - Does this apply to all resources?
- What about the tape-disk-printer example?
Prevention: Avoid Hold and Wait

- **Two-phase locking**
  - **Phase I:**
    - Try to lock all resources at the beginning
  - **Phase II:**
    - If successful, use the resources and release them
    - Otherwise, release all resources and start over

- **Application**
  - Telephone company’s circuit switching

- What about the tape-disk-printer example?
Prevention: No Preemption

- Make the scheduler be aware of resource allocation

Method

- If the system cannot satisfy a request from a process holding resources, preempt the process and release all resources
- Schedule it only if the system satisfies all resources

Alternative

- Preempt the process holding the requested resource

- What about the tape-disk-printer example?
Prevention: No Circular Wait

- Impose an order of requests for all resources
- Method
  - Assign a unique id to each resource
  - All requests must be in an ascending order of the ids
- A variation
  - Assign a unique id to each resource
  - No process requests a resource lower than what it is holding
- What about the tape-disk-printer example?
- Can we prove that this method has no circular wait?
Tradeoffs and Applications

- Ignore the problem for applications
  - It is application developers’ job to deal with their deadlocks
  - OS provides mechanisms to break applications’ deadlocks

- Kernel should not have any deadlocks
  - Use prevention methods
  - Most popular is to apply no-circular-wait principle everywhere
OpenLDAP deadlock, bug #3494

```c
{
  lock(A)
  ...
  lock(B)
  ...
  unlock(A)
  ...
  if ( cursize > maxsize) {
    ...
    for (...) ...
    ...
      lock(A)
      ...
      unlock(A)
      ...
  }
}
... unlock(B)
```
OpenLDAP deadlock, fix #1

Changes the algorithm, but maybe that’s OK
OpenLDAP deadlock, fix #2

```c
{  
    lock(A)  
    ...
    lock(B)  
    ...
    unlock(A)  
    ...
    if ( cursize > maxsize) {
        ...
        for (...)  
            ...
            lock(A)  
            ...
            unlock(A)  
            ...
    }  
}  
```
Apache bug #42031

http://issues.apache.org/bugzilla/show_bug.cgi?id=42031
Summary: EventMPM child process freeze
Product: Apache httpd-2 Version: 2.3-HEAD
Platform: PC
OS/Version: Linux
Status: NEW
Severity: critical
Priority: P2
Component: Event MPM
AssignedTo: bugs@httpd.apache.org
ReportedBy: serai@lans-tv.com

Child process freezes with many downloading against MaxClients.

How to reproduce:
(1) configuration to httpd.conf StartServers 1 MaxClients 3 MinSpareThreads 1 MaxSpareThreads 3 ThreadsPerChild 3 MaxRequestsPerChild 0 Timeout 10 KeepAlive On MaxKeepAliveRequests 0 KeepAliveTimeout 5
(2) put a large file "test.mpg" (about 200MB) on DocumentRoot
(3) apachectl start
(4) execute many downloading simultaneously. e.g. bash and wget:
   $ for (( i=0 ; i<20 ; i++ )); do wget -b http://localhost/test.mpg; done;
   Then the child process often freezes. If not, try to download more.
(5) terminate downloading e.g. bash and wget: $ killall wget
(6) access to any file from web browser. However long you wait, server won't response.
Apache deadlock, bug #42031

listener_thread(...) {
    lock(timeout)
    ...
    lock(idlers)
    ...
    cond_wait(wait_for_idler, idlers)
    ...
    unlock(idlers)
    ...
    unlock(timeout)
}

worker_thread(...) {
    lock(timeout)
    ...
    unlock(timeout)
    ...
    lock(idlers)
    ...
    signal(wait_for_idler)
    ...
    unlock(idler)
    ...
}
Principle of Least Astonishment

- People are part of the system. The design should match the user’s experience, expectations, and mental models.
- With this, system works intuitively
- Without this, users get disoriented, confused, angry, …

Example: original iPad (2010)

- Precursors: Newton, PalmPilot, Pocket PC, Tablet PC, etc.
- Less capable than a PC yet more expensive
- But it took off… why?
Summary

- **Deadlock conditions**
  - Mutual exclusion
  - Hold and wait
  - No preemption
  - Circular chain of requests

- **Strategies to deal with deadlocks**
  - Simpler ways are to negate one of the four conditions