CSE 6431

- Traditional Operating Systems deal with typical system software designed to be:
  - general purpose
  - running on single processor machines
- Advanced Operating Systems are designed for either a special purpose, or for more complex architectures
  - Special purpose o.s. open new design dimensions
    - examples: real-time systems, fault tolerant design, database systems
  - Complex architectures
    - Multiprocessor machines, distributed architectures (i.e. over a SAN, over the Internet, …)
- In this course we will be dealing with some of the conceptual issues that arise in the design of system software for distributed and special purpose systems.
Course content and objectives

“CSE 6431 is a course on advanced concepts and mechanisms in operating systems”

Course topics

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5</td>
<td>Process Synch/Mutual Exclusion</td>
</tr>
<tr>
<td>1.5</td>
<td>Distributed System Basics</td>
</tr>
<tr>
<td>1</td>
<td>Distributed Mutual Exclusion</td>
</tr>
<tr>
<td>1.5</td>
<td>OS for Databases</td>
</tr>
<tr>
<td>1.5</td>
<td>Deadlock Analysis</td>
</tr>
<tr>
<td>2</td>
<td>Fault-Tolerance</td>
</tr>
<tr>
<td>2</td>
<td>Coherence/Race Detec.</td>
</tr>
<tr>
<td>1</td>
<td>Virtualization</td>
</tr>
<tr>
<td>1</td>
<td>Misc</td>
</tr>
</tbody>
</table>

Course objectives

- to become aware of issues found in the design of distributed systems in general and the design of advanced operating systems in particular
- to become familiar with techniques that have been proposed for dealing with these issues.
Definition of a Distributed System (1)

- A distributed system is:

A collection of independent computers that appears to its users as a single coherent system.
Distributed systems issues

- Distributed systems introduce a whole new set of design issues w.r.t traditional system design
- Scalability
- Transparency
- On multicomputers:
  - Lack of common address space
  - Lack of common clock
Different basic organizations and memories in distributed computer systems
## Transparency in a Distributed System

<table>
<thead>
<tr>
<th>Transparency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Hide differences in data representation and how a resource is accessed</td>
</tr>
<tr>
<td>Location</td>
<td>Hide where a resource is located</td>
</tr>
<tr>
<td>Migration</td>
<td>Hide that a resource may move to another location</td>
</tr>
<tr>
<td>Relocation</td>
<td>Hide that a resource may be moved to another location while in use</td>
</tr>
<tr>
<td>Replication</td>
<td>Hide that there may be multiple copies of a resource</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Hide that a resource may be shared by several competitive users</td>
</tr>
<tr>
<td>Failure</td>
<td>Hide the failure and recovery of a resource</td>
</tr>
<tr>
<td>Persistence</td>
<td>Hide whether a (software) resource is in memory or on disk</td>
</tr>
</tbody>
</table>

Different forms of transparency in a distributed system.
### Scalability Problems

<table>
<thead>
<tr>
<th><strong>Concept</strong></th>
<th><strong>Example</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized services</td>
<td>A single server for all users</td>
</tr>
<tr>
<td>Centralized data</td>
<td>A single on-line telephone book</td>
</tr>
<tr>
<td>Centralized algorithms</td>
<td>Doing routing based on complete information</td>
</tr>
</tbody>
</table>

Examples of scalability limitations.
Concept of a process

- In the context of this course a process is a program whose execution is in progress.
- States of a process: running, ready, blocked
Concurrent processes

• In a multiprocessor system two or more processes can be in execution at the same time
  – physical concurrency - as opposed to logical concurrency achieved by interleaving process execution

• Concurrent processor interaction:
  – shared variables
  – message passing

• If they don’t interact their execution is functionally the same as their serial execution
The critical section problem

- A critical section is a code segment of a concurrent process in which a shared resource is accessed
- Concurrent access to a shared variable is potentially dangerous
  - example: if $a=0$, what is the result of the command $a=a+1$ executed simultaneously by processes A and B?
  - a common solution is the mutual exclusion i.e. serialization of accesses
Early Solutions

• Busy Waiting
  – Wastes cycles

• Disabling Interrupts
  – Only applicable to uniprocessors

• A special test-and-set instruction
Example of busy waiting on a lock (1/2)

- One could think of using a variable as a flag to be checked upon entering a critical section ...
- … but the lock itself is a critical section!

```c
Shared integer lock = 0;
Process i
  .
  .
  while lock == 1;
  lock = 1;
  execute CS;
  lock = 0;
  .

Process A
  .
  .
  while lock == 1;
  lock = 1;
  .
  .

Process B
  .
  .
  while lock == 1;
  lock = 1;
  .
  .
```

Possible race condition
Example of busy waiting on a lock (2/2)

- The correct implementation uses a test-and-set instruction to avoid race conditions

Semantic of test-and-set instruction

```c
int test-and-set (int a) {
    int rv = a;
    a = 1;
    return rv;
}
```

Correct lock implementation

```c
Process A

Shared integer lock = 0;
.
.
While( test-and-set(lock) ==1)
    ;
.
.
```
Locks: pros and cons

• Pros:
  – simple and fast
  – ubiquitous: every processor has a test-and-set or equivalent operation

• Cons:
  – busy waiting is wasteful of resources (CPU cycles, memory bandwidth)
Semaphores - definition

- Proposed by Dijkstra, it was the first high level constructs used to synchronize concurrent processes.
- A semaphore $S$ is an integer variable on which two atomic operations are defined, $P(S)$ and $V(S)$, and with an associated queue.
- $P$ and $V$ semantic:

  $$P(S): \text{if } S \geq 1 \text{ then } S := S - 1$$
  $$\text{else } <\text{block and enqueue the process}>;$$

  $$V(S): \text{if } <\text{some process is blocked on the queue}> \text{ then }$$
  $$<\text{unblock a process}>$$
  $$\text{else } S := S + 1;$$
Semaphores - properties

- The P operation may block a process, but V does not
- Two type of semaphores
  - binary: initial value is 1
  - resource counting: any initial value
- P and V are atomic operations

\[
P(S): \text{if } S \geq 1 \text{ then } S := S - 1 \\
\text{else } <\text{block and enqueue the process}>; \\
\]

\[
V(S): \text{if } <\text{some process is blocked on the queue}> \text{ then} \\
<\text{unblock a process}> \\
\text{else } S := S + 1; \\
\]
Example of use

Shared var mutex: semaphore = 1;

Process $i$

    begin
    .
    .
    P(mutex);
    execute CS;
    V(mutex);
    .
    .
    End;
Other synchronization problems

• Semaphore can be used in other synchronization problems besides Mutual Exclusion

• The Producer-Consumer problem
  – a finite buffer pool is used to exchange messages between producer and consumer processes

• The Readers-Writers Problem
  – reader and writer processes accessing the same file

• The Dining Philosophers Problem
  – five philosophers competing for a pair of forks