Concurrency: Mutual Exclusion and Synchronization

Chapter 5
Concurrency

- Multiple applications
- Structured applications
- Operating system structure
## Concurrency

### Table 5.1 Some Key Terms Related to Concurrency

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>critical section</strong></td>
<td>A section of code within a process that requires access to shared resources and which may not be executed while another process is in a corresponding section of code.</td>
</tr>
<tr>
<td><strong>deadlock</strong></td>
<td>A situation in which two or more processes are unable to proceed because each is waiting for one of the others to do something.</td>
</tr>
<tr>
<td><strong>livelock</strong></td>
<td>A situation in which two or more processes continuously change their state in response to changes in the other process(es) without doing any useful work.</td>
</tr>
<tr>
<td><strong>mutual exclusion</strong></td>
<td>The requirement that when one process is in a critical section that accesses shared resources, no other process may be in a critical section that accesses any of those shared resources.</td>
</tr>
<tr>
<td><strong>race condition</strong></td>
<td>A situation in which multiple threads or processes read and write a shared data item and the final result depends on the relative timing of their execution.</td>
</tr>
<tr>
<td><strong>starvation</strong></td>
<td>A situation in which a runnable process is overlooked indefinitely by the scheduler; although it is able to proceed, it is never chosen.</td>
</tr>
</tbody>
</table>
Difficulties of Concurrency

- Sharing of global resources
- Operating system managing the allocation of resources optimally
- Difficult to locate programming errors
Currency

- Communication among processes
- Sharing resources
- Synchronization of multiple processes
- Allocation of processor time
Concurrency

- Multiple applications
  - Multiprogramming

- Structured application
  - Application can be a set of concurrent processes

- Operating-system structure
  - Operating system is a set of processes or threads
A Simple Example

```c
void echo()
{
    chin = getchar();
    chout = chin;
    putchar(chout);
}
```
A Simple Example

- Assume
  - single processor
  - 2 processes execute echo
  - global variables

What are the possible outputs?

```c
void echo()
{
    chin = getchar();
    chout = chin;
    putchar(chout);
}
```
A Simple Example

Now assume 2 processors

Process P1

.  

chin = getchar(); 

.  

chout = chin; 

putchar(chout); 

.  

.

Process P2

.  

.

chin = getchar();

.

chout = chin;

.

putchar(chout);

.

.
Operating System Concerns

- Keep track of various processes
- Allocate and deallocate resources
  - Processor time
  - Memory
  - Files
  - I/O devices
- Protect data and resources
- Output of process must be independent of the speed of execution of other concurrent processes
Process Interaction

- Processes unaware of each other
- Processes indirectly aware of each other
- Process directly aware of each other
<table>
<thead>
<tr>
<th>Degree of Awareness</th>
<th>Relationship</th>
<th>Influence that one Process has on the Other</th>
<th>Potential Control Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processes unaware of each other</td>
<td>Competition</td>
<td>• Results of one process independent of the action of others</td>
<td>• Mutual exclusion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Timing of process may be affected</td>
<td>• Deadlock (renewable resource)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Starvation</td>
</tr>
<tr>
<td>Processes indirectly aware of each other</td>
<td>Cooperation by sharing</td>
<td>• Results of one process may depend on information obtained from others</td>
<td>• Mutual exclusion</td>
</tr>
<tr>
<td>(e.g., shared object)</td>
<td></td>
<td>• Timing of process may be affected</td>
<td>• Deadlock (renewable resource)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Starvation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Data coherence</td>
</tr>
<tr>
<td>Processes directly aware of each other</td>
<td>Cooperation by</td>
<td>• Results of one process may depend on information obtained from others</td>
<td>• Deadlock (consumable resource)</td>
</tr>
<tr>
<td>(have communication primitives available</td>
<td>communication</td>
<td>• Timing of process may be affected</td>
<td>• Starvation</td>
</tr>
<tr>
<td>to them)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Competition Among Processes for Resources

- **Mutual Exclusion**
  - Critical sections
    - Only one program at a time is allowed in its critical section
    - Example only one process at a time is allowed to send command to the printer

- **Deadlock**

- **Starvation**
Requirements for Mutual Exclusion

• Only one process at a time is allowed in the critical section for a resource

• A process that halts in its non-critical section must do so without interfering with other processes

• No deadlock or starvation
Requirements for Mutual Exclusion cont.

- A process must not be delayed access to a critical section when there is no other process using it
- No assumptions are made about relative process speeds or number of processes
- A process remains inside its critical section for a finite time only
Mutual Exclusion: Hardware Support

• Interrupt Disabling
  – In general: A process runs until it invokes an operating system service or until it is interrupted
  – Uni-processor: Disabling interrupts guarantees mutual exclusion
    • Processor is limited in its ability to interleave programs
  – Multiprocessing
    • disabling interrupts on one processor will not guarantee mutual exclusion
Mutual Exclusion: Hardware Support

• Special Machine Instructions
  – Performed in a single instruction cycle
  – Access to the memory location is blocked for any other instructions
Mutual Exclusion: Hardware Support

- Test and Set Instruction

```java
boolean testset (int i) {
    if (i == 0) {
        i = 1;
        return true;
    } else {
        return false;
    }
}
```
Mutual Exclusion: Hardware Support

- Exchange Instruction

```c
void exchange(int register,
              int memory) {
    int temp;
    temp = memory;
    memory = register;
    register = temp;
}
```
Mutual Exclusion

- **parbegin**: initiate all processes and resume program after all Pi’s have terminated

```c
/* program mutualexclusion */
const int n = /* number of processes */;
int bolt;
void P(int i)
{
    while (true)
    {
        while (!testset (bolt))
            /* do nothing */;
        /* critical section */;
        bolt = 0;
        /* remainder */
    }
}

void main()
{
    bolt = 0;
    parbegin (P(1), P(2), ... , P(n));
}
```

```c
/* program mutualexclusion */
int const n = /* number of processes*/;
int bolt;
void P(int i)
{
    int keyi;
    while (true)
    {
        keyi = 1;
        while (keyi != 0)
            exchange (keyi, bolt);
        /* critical section */;
        exchange (keyi, bolt);
        /* remainder */
    }
}

void main()
{
    bolt = 0;
    parbegin (P(1), P(2), ... , P(n));
}
```

(a) Test and set instruction  (b) Exchange instruction

**Figure 5.2** Hardware Support for Mutual Exclusion
Mutual Exclusion Machine Instructions

• Advantages
  – Applicable to any number of processes on either a single processor or multiple processors sharing main memory
  – It is simple and therefore easy to verify
  – It can be used to support multiple critical sections
Mutual Exclusion Machine Instructions

• Disadvantages
  – Busy-waiting consumes processor time
  – Starvation is possible when a process leaves a critical section and more than one process is waiting.
  – Deadlock
    • If a low priority process has the critical section and a higher priority process needs it, the higher priority process will obtain the processor to wait for the critical section (which will not be returned).