Concurrency: Mutual Exclusion and Synchronization

Chapter 5

1

Concurrency

- Multiple applications
- Structured applications
- Operating system structure

2

Concurrency

Table 5.1 Some Key Terms Related to Concurrency

critical section	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.
deadlock	A situation in which two or more processes are unable to proceed because each is waiting for one of the others to do something.
livelock	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.
mutual exclusion	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.
race condition	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.
starvation	A situation in which a runnable process is overlooked indefinitely by the scheduler; although it is able to proceed, it is never chosen.

3

Difficulties of Concurrency

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

4

Currency

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

5

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

6

A Simple Example

```
void echo()
{
  chin = getchar();
  chout = chin;
  putchar(chout);
}
```


A Simple Example

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

```
void echo()
{
  chin = getchar();
  chout = chin;
  putchar(chout);
}
```

- What are the possible outputs?

A Simple Example

Now assume 2 processors

Process P1	Process P2
.	.
chin = getchar();	.
.	chin = getchar();
chout = chin;	chout = chin;
putchar(chout);	.
.	putchar(chout);
.	.

9

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

10

Process Interaction

- Processes unaware of each other
- Processes indirectly aware of each other
- Process directly aware of each other

11

Table 5.2 Process Interaction

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

12

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

13

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

14

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

15

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

16

Mutual Exclusion: Hardware Support

- Special Machine Instructions
  - Performed in a single instruction cycle
  - Access to the memory location is blocked for any other instructions

17

Mutual Exclusion: Hardware Support

- Test and Set Instruction

```
boolean testset (int i) {
    if (i == 0) {
        i = 1;
        return true;
    }
    else {
        return false;
    }
}
```

18

Mutual Exclusion: Hardware Support

- Exchange Instruction

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

19

Mutual Exclusion

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

```
/* 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));

}
```

(a) Test and set instruction

```
/* 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));
}
```

(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).