CONCURRENCY: MUTUAL EXCLUSION AND SYNCHRONIZATION

CHAPTER 5

MULTIPLE PROCESSES

- Operating System design is concerned with the management of processes and threads:
  - Multiprogramming
  - Multiprocessing
  - Distributed Processing

- Fundamental to all of these areas, and fundamental to OS design, is **concurrency**.

CONCURRENCY AND DESIGN ISSUES

Concurrency encompasses a host of design issues, including;

- communication among processes,
- sharing of and competing for resources
- synchronization of the activities of multiple processes, and
- allocation of processor time to processes.

Concurrency
Arises in Three Different Contexts:

Multiple Applications

Multiprogramming was invented to allow processing

time to be dynamically shared among a number of active applications.

Structured Applications

As an extension of the principles of modular design

and structured programming, some applications can be effectively programmed

as a set of concurrent processes.

Operating System Structure

The same structuring advantages apply to systems

programs, and we have seen that operating systems are themselves

often implemented as a set of processes or threads.

Some Key Terms Related to Concurrency

Atomic operation	A function or action implemented as a sequence of one or more instructions that appears to be indivisible; that is, no other process can see an intermediate state or interrupt the operation. The sequence of instruction is guaranteed to execute as a group, or not execute at all, having no visible effect on system state. Atomicity guarantees isolation from concurrent processes.
Critical section	A section of code within a process that requires access to shared resources and that must 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 states 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.

PRINCIPLES OF CONCURRENCY

- Interleaving and Overlapping
  - can be viewed as examples of concurrent processing
  - **both present the same problems**

- In both multiprogramming and multiprocessing cases, the problems stem from… The relative speed of execution of processes cannot be predicted.
  - It depends on the activities of other processes, the way in which the OS handles interrupts, and the scheduling policies of the OS.

Extra Slide

- In a **single-processor multiprogramming system**, processes are ***interleaved*** in time to yield the appearance of simultaneous execution
  - Even though actual parallel processing is not achieved, and even though there is a certain amount of overhead involved in switching back and forth between processes, interleaved execution provides major benefits in processing efficiency and in program structuring.
- In a **multiple-processor system**, it is possible not only to interleave the execution of multiple processes but also to ***overlap*** them
- All of the foregoing difficulties present themselves in a multiprocessor system as well, because a multiprocessor system must also deal with problems arising from the simultaneous

Difficulties of Concurrency

1. **Sharing of global resources**
   - For example, if two processes both make use of the same global variable and both perform reads and writes on that variable, then the order in which the various reads and writes are executed is critical.

2. **Difficult for the OS to manage the allocation of resources optimally**
   - For example, process A may request use of, and be granted control of, a particular I/O channel and then be suspended before using that channel. It may be undesirable for the OS simply to lock the channel and prevent its use by other processes; indeed this may lead to a deadlock condition

3. **Difficult to locate programming errors**
   - because results are typically not deterministic and

- **3. It becomes very difficult to locate a programming error** because results are typically not deterministic and reproducible (e.g., see [LEBL87, CARR89, SHEN02] for a discussion of this point).

- All of the foregoing difficulties present themselves in a multiprocessor system as well, because here too the relative speed of execution of processes is unpredictable. A multiprocessor system must also deal with problems arising from the simultaneous execution of multiple processes. Fundamentally, however, the problems are the same as those for uniprocessor systems. This should become clear as the discussion proceeds.

A Simple Example

- Consider the following procedure:

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

- Any program can call this procedure repeatedly to accept user input and display it on the user’s screen.
- Now consider that we have a single-processor multiprogramming system supporting a single user. The user can jump from one application to another, and each application uses the same keyboard for input and the same screen for output.
- Because each application needs to use the procedure echo, it makes sense for it to be a shared procedure that is loaded into a portion of memory global to all applications. Thus, only a single copy of the echo procedure is used, saving space.
- The sharing of main memory among processes is useful to permit efficient and close interaction among processes. However, ***such sharing can lead to problems***.

A Simple Example

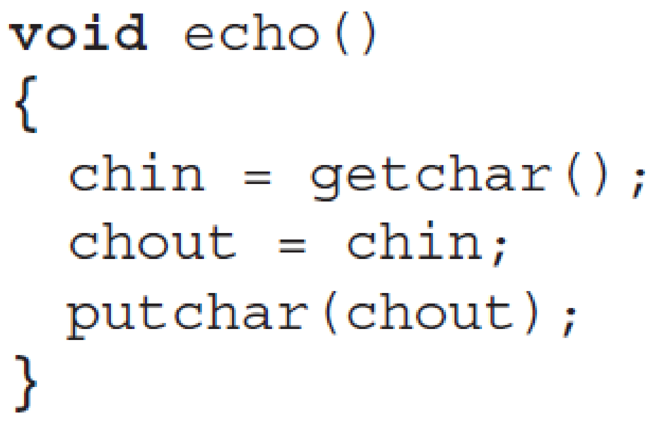
- Consider the following procedure:

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

- Process P1 invokes the echo procedure and is interrupted immediately after getchar returns its value and stores it in chin. At this point, the most recently entered character, x, is stored in variable chin.
- **2.** Process P2 is activated and invokes the echo procedure, which runs to conclusion, inputting and then displaying a single character, y, on the screen.
- **3.** Process P1 is resumed. By this time, the value x has been overwritten in chin and therefore lost. Instead, chin contains y, which is transferred to chout and displayed.
- **The essence of this problem is the *shared global variable*, chin.**

A Simple Example

- Consider the following procedure:



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

Suppose, however, that **we permit only one process at a time to be in that procedure**. Then the foregoing sequence would result in the following:

1. Process P1 invokes the echo procedure and is interrupted immediately after the conclusion of the input function. At this point, the most recently entered character, x, is stored in variable chin.
2. Process P2 is activated and invokes the echo procedure. However, because P1 is still inside the echo procedure, although currently suspended, P2 is blocked from entering the procedure. Therefore, P2 is suspended awaiting the availability of the echo procedure.
3. At some later time, process P1 is resumed and completes execution of echo. The proper character, x, is displayed.
4. When P1 exits echo, this removes the block on P2. When P2 is later resumed, the echo procedure is successfully invoked.

- **This example shows that it is necessary to protect shared global variables (and other shared global resources) and that the only way to do that is to control the code that accesses the variable.**
- **How that discipline may be imposed is a major topic of this chapter.**

An other Example

First, suppose that there is no mechanism for controlling access to the shared global variable:

1. Processes P1 and P2 are both executing, each on a separate processor. Both processes invoke the echo procedure.
2. The following events occur; events on the same line take place in parallel:

```
        Process P1                 Process P2
    •                          •
    chin = getchar();          •
    •                          chin = getchar();
    chout = chin;              chout = chin;
    putchar(chout);            •
    •                          putchar(chout);
    •                          •
```

Solution

- In the case of a uniprocessor system, the reason we have a problem is that an *interrupt can stop instruction execution anywhere* in a process.

- In the case of a multiprocessor system, we have that same condition and, in addition, a problem can be caused because *two processes may be executing simultaneously* and both trying to access the same global variable.

- However, the solution to both types of problem is the same: **control access to the shared resource**.

Race Condition

- A race condition occurs when multiple processes or threads read and write data items so that the **final result depends on the order of execution of instructions in the multiple processes**.

<u>Example 1:</u>

- suppose that two processes, P1 and P2, share the global variable a.
- At some point in its execution, P1 updates a to the value 1, and at some point in its execution
- P2 updates a to the value 2.
- Thus, the two tasks are in a race to write variable a.
- The "loser" of the race (the process that updates last) determines the final value of a.

<u>Example 2:</u>

- consider two processes, P3 and P4, that share global variables b and c, with initial values b = 1 and c = 2.
- At some point in its execution, P3 executes the assignment b = b + c,
- At some point in its execution, P4 executes the assignment c = b + c.
- Note that the two processes update different variables. However, the final values of the two variables depend on the order in which the two processes execute these two assignments.

Operating System Concerns

- Design and management **issues raised by the existence of concurrency**:
  - The OS must:

- be able to keep track of various processes
- allocate and de-allocate resources for each active process
- protect the data and physical resources of each process against interference by other processes
- ensure that the processes and outputs are independent of the processing speed