CSCI 340
Lecturer: Dr. Simina Fluture

Lecture # 8

Process Synchronization (Introduction)
Principles of Concurrency

Read: Class Notes
Web Lecture
Textbook

Topics:
Degrees of interaction between processes (threads) & Potential Control Problems
Critical Section & the Critical Section Problem
Constraints on acceptable solutions to CS Problem
The general structure of a typical process P

Between processes there might be interactions. These interactions can be classified on the basis of the degree to which processes are aware of each other's existence.

Processes unaware of each other: independent processes that are not intended to work together. The O.S. needs to be concerned about the competition for resources.
The best example is the multiprogramming of multiple independent processes.
Potential Control Problems: Mutual Exclusion, Deadlock, Starvation.

Processes indirectly aware of each other: processes those are not necessarily aware of each other by name but share access to some object, such as an I/O buffer. Such processes need cooperation in sharing the common object. Multiple processes may have access to shared variables, files or databases.
Potential Control Problems: Mutual Exclusion, Deadlock, Starvation, Data Coherence. (Producer/Consumer, Reader/Writer problem)

Processes directly aware of each other: processes able to communicate with each other by name. The communication provides a way to synchronize or coordinate the various activities.
Potential Control Problems: Deadlock, Starvation.

CSCI 340
Lecturer: Dr. Simina Fluture

Critical Section & the Critical Section Problem

Let's consider the following situation.
The registration period is over, but for the next 2 weeks students can drop or add courses. Students can drop/add classes by calling one of the two operators. One of the operators is responsible for the 'add' cases, the other one for the 'drop' cases.

***A*: the number of students currently registered.**
A is a shared variable.

```
P0
add ( ) {
   while (true) {
      .........
      A++;
      .........
   }
}
```

```
P1
drop ( ) {
   while (true) {
      .........
      A--;
      …….
   }
}
```

Suppose that we don't know the order of students' requests, but if we consider the case by which one student called to drop a course and another called to add the course, after both students finished their transactions, the number of registered students remains unchanged.

The corresponding lower level instructions are: (given in class)

Let's consider the following sequences of instructions (up - down):

P0-0	P0-0	P1-0
P0-1	P1-0	P1-1
P1-0	P0-1	P1-2
P1-1	P1-1	P0-0
P1-2	P0-2	P0-1
P0-2	P1-2	P0-2

Initially: A = 20

What will be the value of A, after the execution of each sequence of instructions?

The final result depends on the order in which the processes (threads) access the shared value.

A++ and **A—** represent the **Critical Sections** (CS).

Critical Section: is a segment of code in which the process may be access common variables, shared data, etc.

In order to solve the previous problem, we must require only one process to be allowed in its CS at a given time (**Mutual Exclusion condition**).

CSCI 340
Lecturer: Dr. Simina Fluture

Data coherence represents a problem not only for the lower-level instructions.
We can have the following situation. Suppose two items of data A and B are to be maintained in the relation A=B. Any process that updates one value, must also update the other one.

```
P0:   A = A++;
      B = B++;

P1:   B = 2*B;
      A = 2*A;
```

Even if we keep mutual exclusion on the execution of each individual instruction, we might still have a data coherence problem.

Race Condition: When several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place.

Critical Section Problem: design a protocol that processes can use to cooperate and properly use the data of the critical section.

Constraints on acceptable solutions to CS problem
Mutual Exclusion with respect to CS. At a given time, only one process can be in the CS.
Progress Condition: if no process is in the CS and there are processes trying to enter their CS, then only processes competing for the CS should participate in the decision of which will enter the CS next and the decision must be made in finite time.
No Starvation: no process should be postponed for an indefinite period of time.

Assumptions:
Write and Read memory cell are atomic operations (indivisible).
Processes do not have priorities.
The relative process speeds are unknown.
Processes are assumed to be sequential and cyclic.

The general structure of a typical process Pi:

```
while(true) {
      entry section
        CS
      exit section
      remainder section
}
```