Concurrency & Semaphores

Multiple Processes

- Central to the design of modern Operating Systems is managing multiple processes
  - Multiprogramming
  - Multiprocessing
  - Distributed Processing
- Big Issue is Concurrency
  - Managing the interaction of all of these processes

Concurrency

Concurrency arises in:

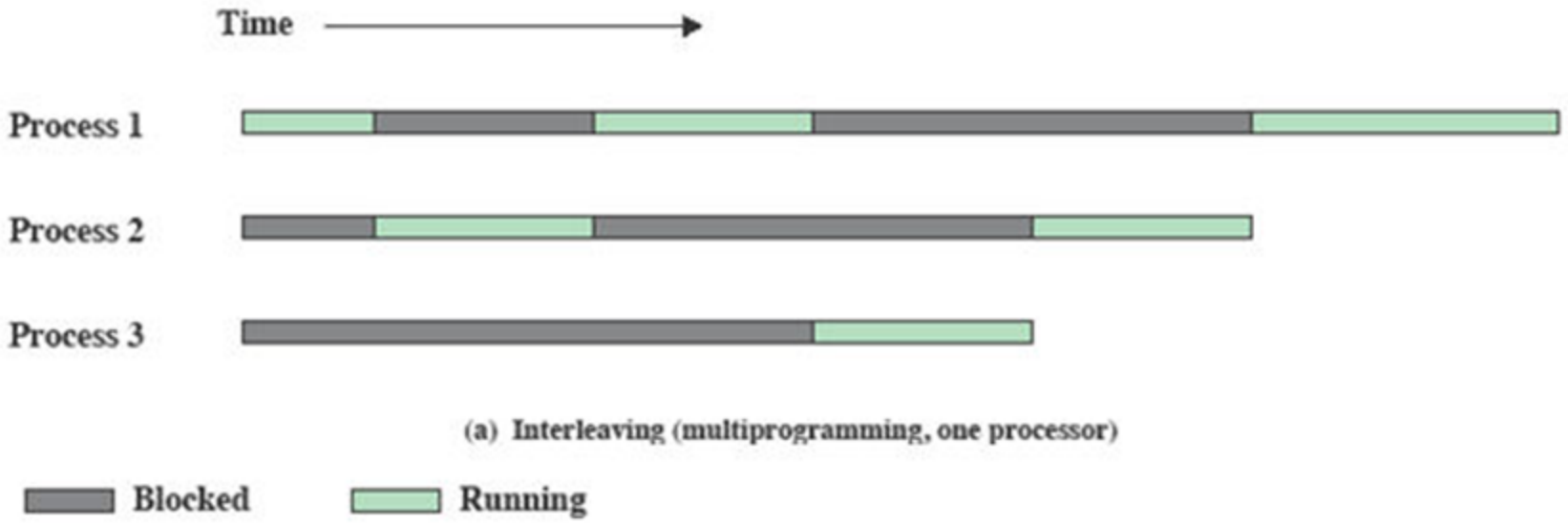
- Multiple applications
  - Sharing time
- Structured applications
  - Extension of modular design
- Operating system structure
  - OS themselves implemented as a set of processes or threads

Key Terms

atomic operation	A sequence of one or more statements that appears to be indivisible; that is, no other process can see an intermediate state or interrupt the operation.
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.

Interleaving and Overlapping Processes

- Earlier we saw that processes may be interleaved on uniprocessors



Interleaving and Overlapping Processes

- And not only interleaved but overlapped on multi-processors

(b) Interleaving and overlapping (multiprocessing; two processors)

Difficulties of Concurrency

- Sharing of global resources- shared read and write by more than one processes
- Optimally managing the allocation of resources- by os. Ex process suspended which is granted i/o channel access
- Difficult to locate programming errors as results are not deterministic and reproducible.

A Simple Example

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

A Simple Example: On a Multiprocessor

Process P1

```
        .
chin = getchar();
        .
chout = chin;
putchar(chout);
        .
        .
```

Process P2

```
        .
        .
chin = getchar();
chout = chin;
        .
putchar(chout);
        .
```

Enforce Single Access

- If we enforce a rule that only one process may enter the function at a time then:
- P1 & P2 run on separate processors
- P1 enters echo first,
  - P2 tries to enter but is blocked – P2 suspends
- P1 completes execution
  - P2 resumes and executes echo

Race Condition

- A race condition occurs when
  - Multiple processes or threads read and write data items
  - They do so in a way where the final result depends on the order of execution of the processes.
- The output depends on who finishes the race last.

Operating System Concerns

- What design and management issues are raised by the existence of concurrency?
- The OS must
  - Keep track of various processes
  - Allocate and de-allocate resources
  - Protect the data and resources against interference by other processes.
  - Ensure that the processes and outputs are independent of the processing speed

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

Competition among Processes for Resources

Need for mutual exclusion.

- Suppose two or more processes require access to a single nonsharable resource, such as a printer.
- During the course of execution, each process will be sending commands to the I/O device, receiving status information, sending data, and/or receiving data.
    - We will refer to such a resource as a **critical resource**, and the portion of the program that uses it a **critical section** of the program.
- It is important that only one program at a time be allowed in its critical section.
- We cannot simply rely on the OS to understand and enforce this restriction because the detailed requirements may not be obvious.
- In the case of the printer, for example, we want any individual process to have control of the printer while it prints an entire file.
    - Otherwise, lines from competing processes will be interleaved.

The enforcement of mutual exclusion creates two additional control problems:

- **deadlock.**
  - Two processes is waiting for the same resources (or each waiting for a resource that the other has exclusive use to)
  - Neither will release the resource that it already owns until it has acquired the other resource and performed the function requiring both resources.
  - The two processes are deadlocked.
- **starvation.**
  - The OS may grant access to resources to a number of processes while neglecting another

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 noncritical section must do so without interfering with other processes
- No deadlock or starvation

Requirements for Mutual Exclusion

- 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

In Uniprocessors

- Uniprocessors only allow interleaving but no overlapping
- Interrupt Disabling – sufficient for mutual excluion
  - A process runs until it invokes an operating system service or until it is interrupted
  - Disabling interrupts guarantees mutual exclusion
  - Will not work in multiprocessor architecture

Pseudo-Code

```
while (true) {
   /* disable interrupts */;
   /* critical section */;
   /* enable interrupts */;
   /* remainder */;
}
```