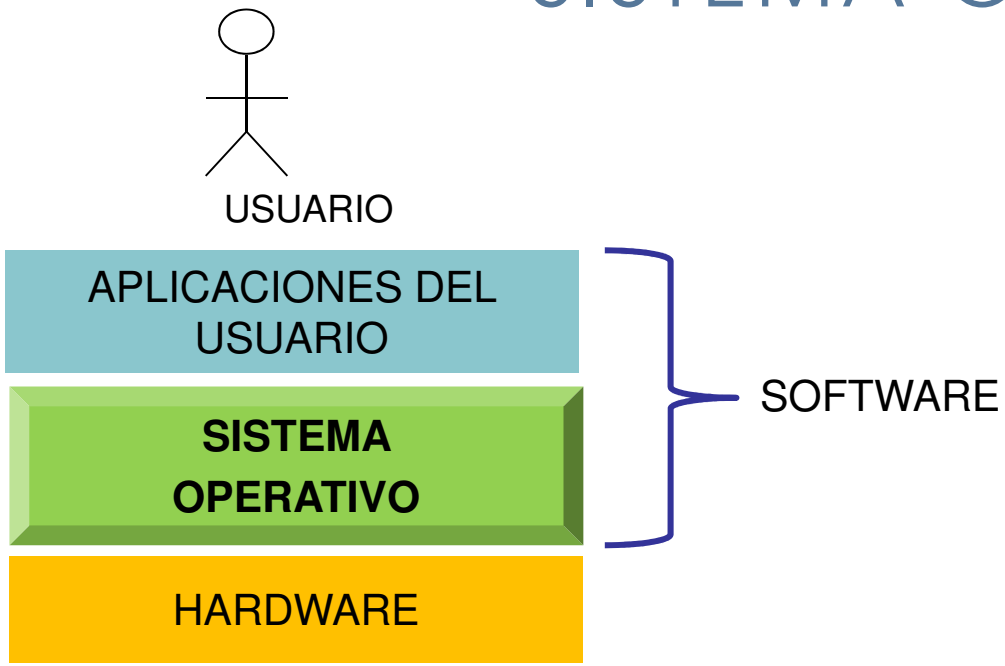
Universidad Nacional de Lanús
Departamento de Desarrollo Productivo y Tecnológico
Licenciatura en Sistemas

Unidad Nº 1:

Administración de Concurrencia entre Procesos

Sistemas Operativos

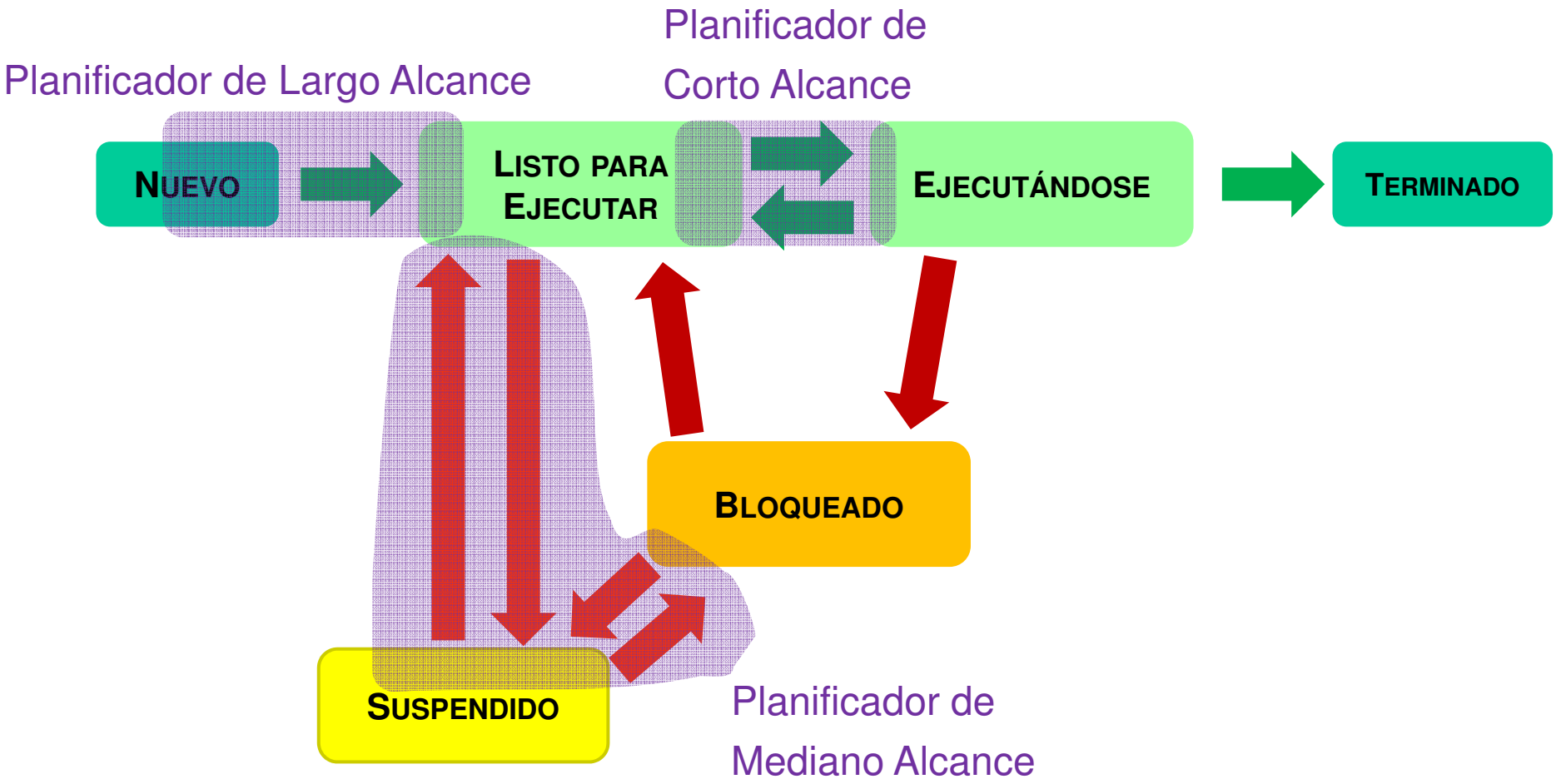
SISTEMA OPERATIVO



- **Objetivos del SO:**
  - Abstraer la complejidad del hardware al usuario y sus aplicaciones.
  - Administrar y proteger los recursos de la computadora.

CONCURRENCIA ENTRE PROCESOS

- Estados de los Procesos



CONCURRENCIA ENTRE PROCESOS

Planificador de
Corto Alcance
Apropiativo

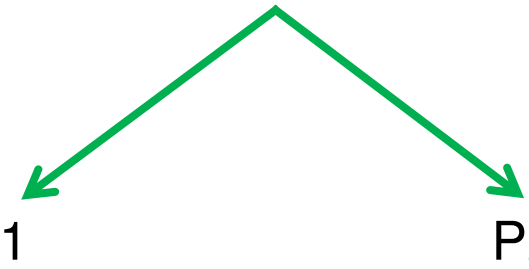
Proceso	Tiempo											
	1	2	3	4	5	6	7	8	9	10	11	12
P1	E					E			E			
P2			E				E				E	
P3				E					E			

Aumenta la
Concurrencia entre Proceso

Es útil
pero
puede generar inconvenientes…

CONCURRENCIA ENTRE PROCESOS


```
función Eco()
{
        car = LeerTeclado();
        pal = BuscarPalabra(car);
        MostrarPantalla(pal);
}
```



car = 'v'

car = 'b'

pal = 'burro'

pal = 'burro'

burro

burro

Aumenta la

Concurrencia entre Proceso

Es útil

pero

puede generar inconvenientes…

Se deben implementar mecanismos para asegurar la correcta sincronización entre procesos…

SINCRONIZACIÓN DE PROCESOS

- Busca que los Procesos no se interfieran entre sí al ejecutarse en forma concurrente.

- Controla *el acceso de los Recursos Compartidos* de los Procesos

Región Crítica

buscando garantizar la ***Exclusión Mutua***.

SINCRONIZACIÓN DE PROCESOS

Procesos Concurrentes sin Sincronización

Procesos Concurrentes Sincronizados

SINCRONIZACIÓN DE PROCESOS

- ***Exclusión Mutua***
  - Condiciones:
    - ✓ Si no hay ningún proceso dentro de la región crítica, un proceso que desee accederla podrá hacerlo.
    - ✓ Sólo un proceso por vez puede acceder a la región crítica.
    - ✓ Un proceso puede estar en la región crítica un tiempo finito.
    - ✓ En algún momento un proceso debe poder acceder a su región crítica.
    - ✓ Un fallo de un proceso fuera de la región crítica no debe afectar al resto.
    - ✓ No se debe asumir velocidad de procesamiento ni cantidad de procesos.

SINCRONIZACIÓN DE PROCESOS

- ***Exclusión Mutua***
  - Mecanismos de Implementación:
    - Software (puro)
    - Hardware (puro)
    - Semáforos
    - Monitores

SINCRONIZACIÓN DE PROCESOS

- Implementación de Exclusión Mutua mediante **Software**:
  - Se basa en la utilización de variables compartidas para controlar el acceso a la región crítica.
  - Ejemplo:

```
var entra = 1;

función Eco()
{
        mientras (entra == 0) { };
        entra = 0;
        car = LeerTeclado();
        pal = BuscarPalabra(car);
        MostrarPantalla(pal);

        entra = 1;
}
```

  - Problemas:
    - Espera activa.
    - Poco confiable.
    - Difícil de implementar y verificar.

SINCRONIZACIÓN DE PROCESOS

- Implementación de Exclusión Mutua mediante **Hardware**:
  - Se basa en la utilización de mecanismos especiales provistos por el hardware para controlar el acceso a la región crítica:
    - ✓ *Deshabilitar Interrupciones*
  - Ejemplo:

```
función Eco()
{
        Interrupciones(off);

        car = LeerTeclado();
        pal = BuscarPalabra(car);
        MostrarPantalla(pal);

        Interrupciones(on);
}
```

  - Problemas:
    - ❖ Reduce la eficiencia del sistema.
    - ❖ Puede generar inconvenientes ante errores.
    - ❖ No es útil para más de un procesador.

SINCRONIZACIÓN DE PROCESOS

- Implementación de Exclusión Mutua mediante **Hardware**:
  - Se basa en la utilización de mecanismos especiales provistos por el hardware para controlar el acceso a la región crítica:
    - ✓ *función atómica Leer-y-Setear(variable, valorComparación, nuevoValor)*
  - Ejemplo:

```
var entra = 1;

función Eco()
{
        mientras ( Leer-y-Setear( entra, 1, 0 ) == falso ) { };

        car = LeerTeclado();
        pal = BuscarPalabra(car);
        MostrarPantalla(pal);

        entra = 1;
}
```

  - Problemas:
    - ❖ Espera activa.
    - ❖ Un proceso puede esperar indefinidamente.

SINCRONIZACIÓN DE PROCESOS

- Implementación de Exclusión Mutua mediante **Semáforos**:
  - Se basa en la utilización de un mecanismo especial provistos por el Sistema Operativo para controlar el acceso a la región crítica: los *Semáforos*
  - Un *Semáforo* es una clase formada por:
    - ✓ *Contador Entero*
    - ✓ *Cola de Espera de Procesos (FIFO)*
    - ✓ *Funciones Atómicas: Up & Down*

```
función Down( semáforo S )
{
        S.valor = S.valor – 1;
        si ( S.valor < 0 )  {
            Bloquear( procesoActual );
            AgregarCola( S.cola, procesoActual );
        }
}
```

```
función Up( semáforo S )
{
        S.valor = S.valor + 1;
        si ( S.valor ≤ 0 )  {
            p = SacarCola( S.cola );
            Desbloquear( p );
        }
}
```

SINCRONIZACIÓN DE PROCESOS

- Implementación de Exclusión Mutua mediante **Semáforos**:

  - Ejemplo:

```
Semáforo S = 1;

función Eco()
{
        Down( S );
        car = LeerTeclado();
        pal = BuscarPalabra(car);
        MostrarPantalla(pal);
        Up( S );
}
```

  - Ventajas:
    - ✓ No presenta espera activa.
    - ✓ Es confiable.
  - Problemas:
    - ❖ Si se usan múltiples procesos y semáforos, puede ser difícil de implementar y verificar para evitar Deadlock y Starvation.

SINCRONIZACIÓN DE PROCESOS

- Implementación de Exclusión Mutua mediante **Semáforos**:
  - Ejemplo 2: Productor / Consumidor

```
Vector vec[];
Int pos = 0;
Semáforo S = 1;
Semáforo N = 0;
```

```
función Productor()
{
    mientras (verdadero) {
        Item i = producir();

        Down( S );
        pos = pos + 1;
        vec[pos] = i;
        Up( S );
        Up( N );
    }
}
```

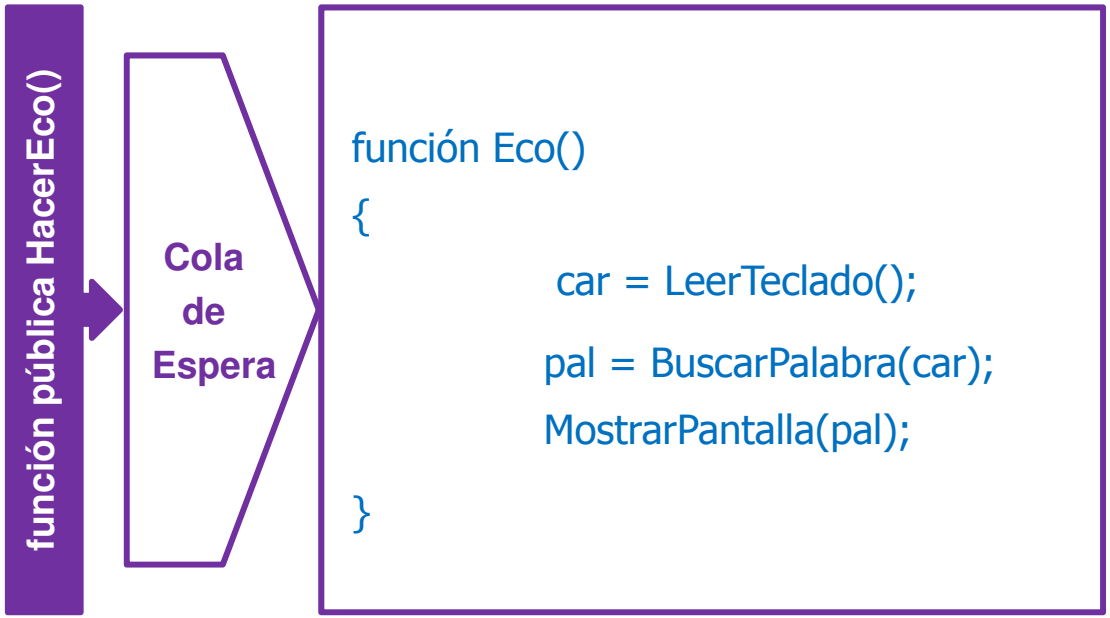
```
función Consumidor()
{
    mientras (verdadero) {
        Down( N );
        Down( S );
        Item j = vec[pos];
        pos = pos - 1;
        Up( S );

        consumir( j );
    }
}
```

SINCRONIZACIÓN DE PROCESOS

- Implementación de Exclusión Mutua mediante **Monitores**:
  - Se basa en la utilización de otro mecanismo especial provistos por el Sistema Operativo para controlar el acceso a la región crítica: los *Monitores*
    - ✓ Un *Monitor* es una clase que encapsula todos los recursos compartidos.
    - ✓ Para acceder a los recursos se deben usar los métodos públicos del *Monitor*.
    - ✓ Se garantiza que sólo un proceso ejecutará cada uno de sus métodos por vez (el resto de los procesos se bloquean hasta que el que ejecuta finalice).

SINCRONIZACIÓN DE PROCESOS

- Implementación de Exclusión Mutua mediante **Monitores**:
  - Ejemplo:



- Ventajas:
  - ✓ No presenta espera activa.
  - ✓ Es confiable.
  - ✓ Es fácil de implementar y verificar.
  - ✓ Reduce la posibilidad de Deadlock y Starvation.
- Problemas:
  - ❖ No es tan flexible como los semáforos.

Bibliografía

- Guía de Estudio Nº 1: *Administración de Concurrencia entre Procesos en Sistemas Operativos* http://sistemas.unla.edu.ar/sistemas/sls/ls-4-sistemas-operativos/pdf/SO-GE1-Concurrencia-de-Procesos.pdf

- Stallings, W. (2011). *Sistemas Operativos - Aspectos Internos y Principios de Diseño*, 7ma Edición Prentice Hall. Capítulo 5.

- Tanenbaum, A.S. (2009). *Sistemas Operativos Modernos*, 3ra Edición Prentice Hall. Capítulo 2 (secciones 2.3 y 2.5).