# *Sistemas Operativos*

## *Cursada 2022*

**Comisión S21 y S22**

# *Sincronización*

- **Herramientas especificas para esto implementadas a nivel Sistema Operativo y Lenguajes de Alto Nivel**

- **Semaforos**

- **Monitores**

# *Semáforos*

- ***Wait = down = P(S)***

La función P(S) es conocida como la que pide el recurso, y si no está disponible queda en un ciclo infinito, ahora si está disponible no entra al while y lo que hace es decrementar la variable **semáforo S**, esto quiere decir que toma el recurso, o variable, etc. O sea cuando accede a lo que llamamos la "**Seccion Critica",** una vez terminada esta porción de código **se** libra con la función V(S).

- ***Signal = Up = V(S)***

Esta función libera el recurso e incrementa el valor del semáforo S o despierta el proceso que esta esperando acceder a la sección critica, si es que lo usamos para el acceso a recursos compartidos, en el caso de que sea tipo Mutex la variable será modificada como 0 o 1 según la implementación

# *Semáforos*

**Definición:** Un semáforo es un **tipo de variable especial** (Tipo de dato Abstracto), que pertenece al dominio de los enteros, esta variable es de tipo protegido y solo la maneja el "**Sistema Operativo**"

## Cuales son las operaciones:

- Inicialización del semáforo (Ej. S=0, 1 o n)
- Pide el recurso ***Wait = down = P(S)***
- Libera el recurso ***Signal = Up = V(S)***

# *Semáforos*

Inicializar_semaforo ( S, valor )

Wait = down = P(S) :

```
while S <= 0 do
    Begin
        Recurso no disponible
    End
S=S-1; -----> Toma el recurso
```

Signal = Up = V(S) : S=S+1; ----- > Libera el recurso

# *Semáforos*

**Tipos:**

- **Binarios (mutex)**
  - Toman valores 0 o 1
- **Contadores**
  - Toman valores mayores a 1
  - Para administrar cantidad de recursos

# *Semáforos*

**Tipos:**

- **Binarios (mutex)**
  - Toman valores 0 o 1
- **Contadores**
  - Toman valores mayores a 1
  - Para administrar cantidad de recursos

# Semáforos

## Espera Activa

```
wait(s)
{
        s = s - 1;
        if (s < 0) {
            <Bloquear al proceso>
        }
}

signal(s)
 {
        s = s + 1;
        if (s <= 0)
            <Desbloquear a un proceso bloqueado por la
        operacion wait>
        }
}
```

# *Ejemplos Semaforos*

- Vamos a ver un ejemplo donde yo quiero sincronizar dos procesos, para que si o si uno deba ejecutarse primero que el otro, Por ejemplo quiero grabar en un archivo la palabra **Hola** y **Adiós**, donde un proceso (P1) graba **Hola**, y el proceso (P2) graba **Adiós**. Es evidente que primero se tiene que ejecutar el P1 y luego P2.

- Como debería inicializar el semáforo para esta problemática?

# *Ejemplos Semaforos*

**Inicializar_semaforo ( S,  )**

**P1**

**{**

Printf(“ Hola\n”);

**}**

**P2**

**{**

Printf(“ Adios\n”);

**}**

# *Ejemplos Semáforos*

**Inicializar_semaforo ( S,  )**

**P1**

{

Function  retirar_dinero( int, dinero)

Aux = Saldo;

Aux = Aux – dinero

Saldo = Aux

}

**P2**

{

Function depositar_dinero( int, dinero)

Aux = Saldo

Aux = Saldo  + dinero

Saldo = Aux

}

# *Ejemplos Semaforos*

**Inicializar_semaforo ( S,  )**

**P1**
**{**
Function  retirar_dinero( int, dinero)
Down(s)←
Aux = Saldo;
Aux = Aux – dinero
Saldo = Aux
Up(s)

**}**

**P2**
**{**
Function depositar_dinero( int, dinero)
Down(s)
Aux = Saldo  ←
Aux = Saldo  + dinero
Saldo = Aux
Up(s)

**}**

# *Monitores*

**Cantidad de procesos en modo concurrente**

- **Antes había pocos procesos**
- **Que pasa si son muchos procesos?**
- **Los semáforos son una buena solución?**
- **Confiamos en los programadores?**

**Para solucionar estos problemas se desarrollo una solución a nivel de los lenguajes de alto nivel.**

**Llamada Monitor**

# *Monitores*

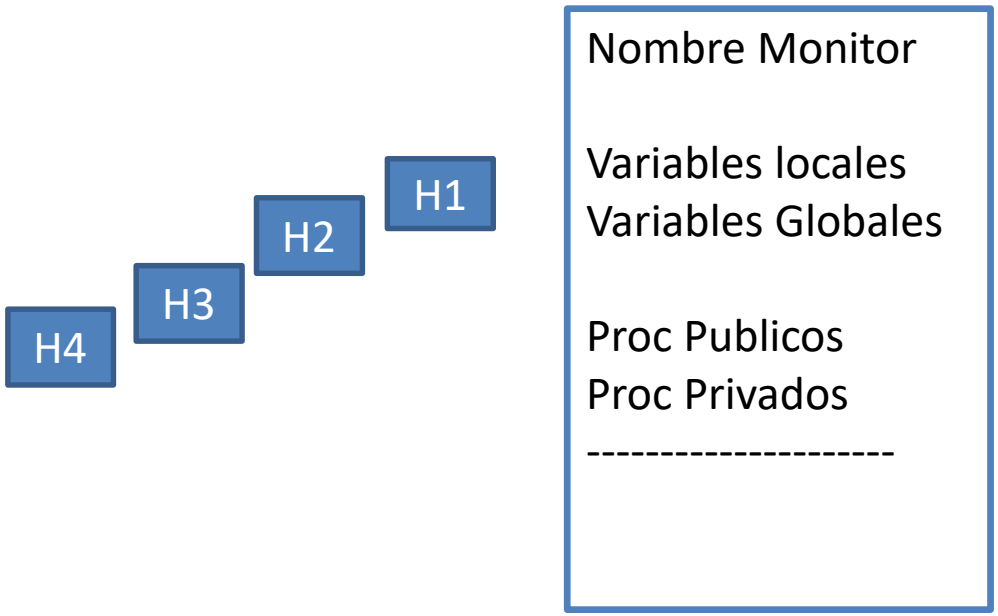
- **Los sistemas operativos proveen una solucion llamada *semáforos***
- **Los compiladores proveen una solucion llamada *Monitor***
- **Los compiladores son parte del sistema operativo?**

**Un monitor es un tipo abstracto de datos, es un método que se invoca, lo que hace es encapsular todas las operaciones que se hacen sobre un recurso compartido. (variables, punteros de buffers, etc)**

# *Monitores*

- **Le pido al monitor que haga algo**
- **El buffers esta disponible**
- ***Donde esta el puntero del buffers***
- **Corre 5 lugares al puntero del buffers**

**La operación la hace el monitor, lo invoco**



# *Monitores*

- **En definitiva encapsula todos los procedimientos**
  - ***Consulta***
  - ***Invocacion***
  - ***Desplazamiento***
  - ***Incremento de una variable (seccion critica)***
  - ***Grabar sobre el recurso***
- **Los hilos van a invocar al método monitor**
- **Solo uno va a poder estar**
- **La única garantía es que el monitor este bien *Programado***

# *Monitores*

- **Existe un tipo de variable exclusiva de los monitores, se llama *"condition"***
- ***Solo se declaran dentro de monitor***
- ***Garantiza la sincronización* plena del monitor**
- **Asociadas algunas a operaciones wait y signal**
- **Administran la cola de entrada al Monitor**

**Por lo tanto decimos que los Sistemas Operativos modernos proveen herramientas como los *Semaforos* y los lenguajes herramientas que me permiten definir *Monitores***

*Fin del Tema*