Taller de Programación

REDICTADO

Clase 1 - Módulo - Programación Concurrente

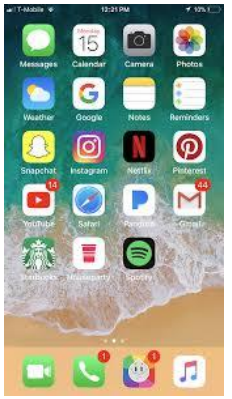
QUÉ ES LA CONCURRENCIA - DÓNDE LA ENCONTRAMOS

Navegadores

Concurrencia: capacidad de ejecutar varias actividades en paralelo o en simultáneo

Acceso a cuentas bancarias

Sistemas Operativos



Smartphones

EVOLUCIÓN DE ARQUITECTURAS

1 núcleo de procesamiento

2,4,8 núcleos de procesamiento

Supercomputadoras **con millones** de núcleos de procesamiento

1980

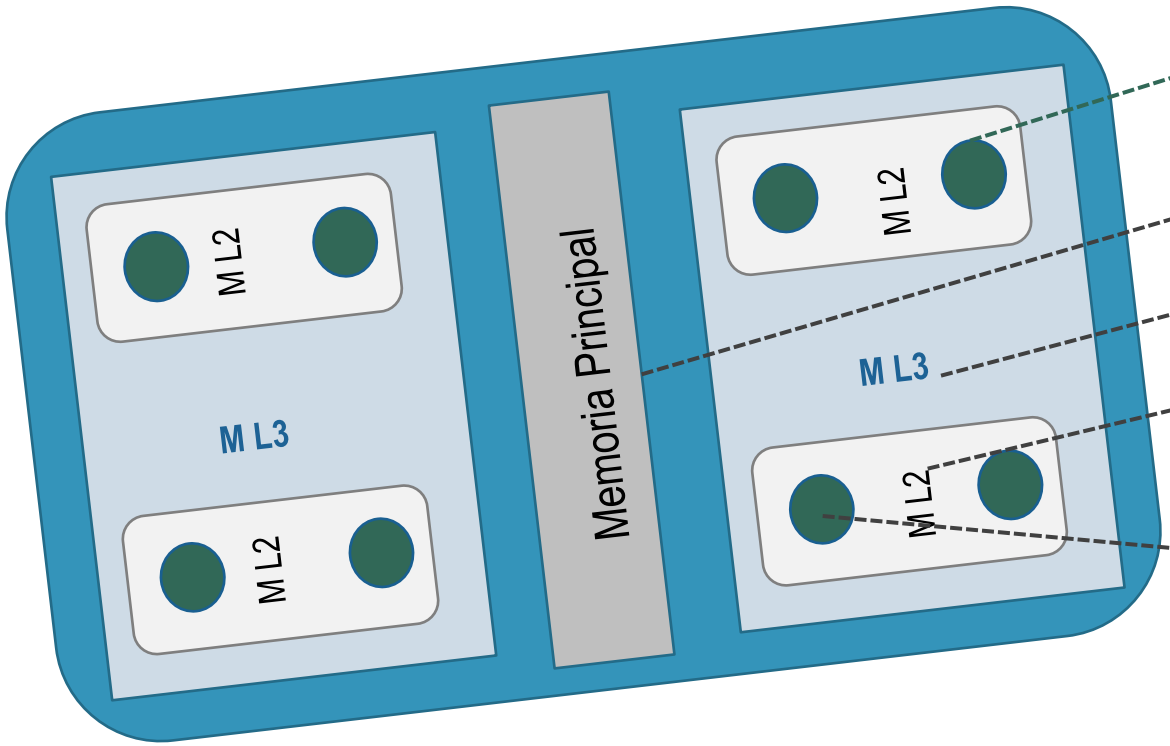
2000

2022

¿Cómo es una **máquina de 8 núcleos**?

EVOLUCIÓN DE ARQUITECTURAS

Ejemplo máquina multinúcleo



NUCLEO

RAM

MEMORIA CACHE (nivel 3)

MEMORIA CACHE (nivel 2)

REGISTROS (local al núcleo)

¿Qué ocurre con **los programas secuenciales** en las arquitecturas actuales?

Para **explotar** este hardware los programas deben ser **concurrentes**

PROGRAMA CONCURRENTE

Un programa concurrente se divide en tareas (2 ó mas), las cuales se ejecutan al mismo tiempo y realizan acciones para cumplir un objetivo común. Para esto pueden: compartir recursos, coordinarse y cooperar.

Características: comunicación - sincronización

¿Por qué comunicar procesos?

¿Por qué sincronizar procesos?

PROGRAMA CONCURRENTE –
Analogía

Automóviles =
Procesos

Carriles =
Recursos compartidos

Los automóviles deben sincronizarse para no chocar

Diseño: debemos examinar los tipos de autos (tipos de procesos) a usar, cuántos procesos, y la necesidad de comunicación y sincronización.

PROGRAMA CONCURRENTE –
Ejemplo 1

Suponga que una pareja comparte su cuenta bancaria (saldo).
En un mismo momento ambos integrantes van al cajero para extraer 1000 pesos.

¿Qué ocurre si ambos **operan** al **mismo tiempo** sobre el **saldo**?

Los procesos deben **sincronizarse** para operar sobre el saldo de a uno por vez


```
Integrante 1:
{
  accede a la cuenta
  saldo:= saldo - 1000;
}
```

50.000

49.000

```
Integrante 2:
{
  accede a la cuenta
  saldo:= saldo - 1000;
}
```

50.000

49.000

PROGRAMA CONCURRENTE –
Ejemplo 1

Suponga que una pareja comparte su cuenta bancaria (saldo).
En un mismo momento ambos integrantes van al cajero para extraer 1000 pesos.

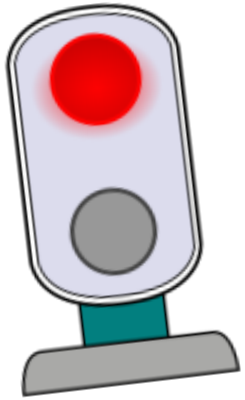
Cualquier lenguaje que brinde concurrencia debe proveer mecanismos para comunicar y sincronizar procesos.

En este caso quiero proteger el acceso a la variable compartida (dos procesos no accedan al mismo tiempo, sincronicen)

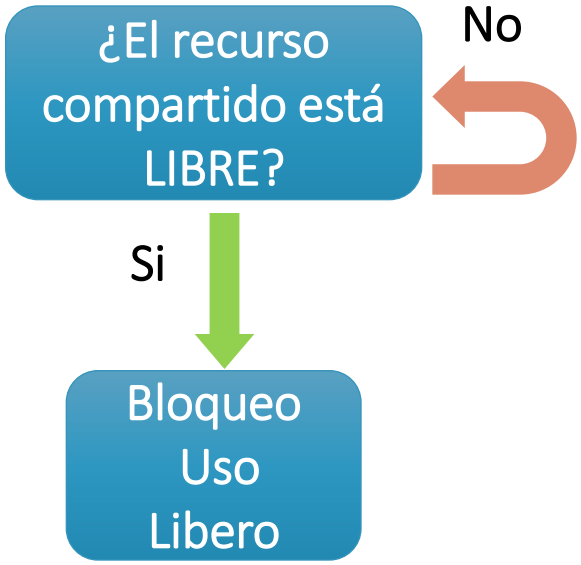
Semáforos
Pasaje de Mensajes
...

PROGRAMA CONCURRENTE –
Ejemplo 1

Suponga que una pareja comparte su cuenta bancaria (saldo).
En un mismo momento ambos integrantes van al cajero para extraer 1000 pesos.



Protocolo



Un **semáforo** asociado al recurso compartido nos indicará cuando el recurso está **LIBRE** u **OCUPADO**.

Admite dos operaciones

- P: demora al proceso hasta que el recurso está libre e inmediatamente lo bloquea
- V: libera el recurso

PROGRAMA CONCURRENTE –
Ejemplo 1

Suponga que una pareja comparte su cuenta bancaria (saldo).
En un mismo momento ambos integrantes van al cajero para extraer 1000 pesos.

Variable compartida **saldo**

¿El código es correcto?

¿Se puede mejorar?

```
Integrante 1:
{ P(saldo)
  accede a la cuenta
  saldo:= saldo - 1000;
  V(saldo)
}
```

```
Integrante 2:
{ P(saldo)
  accede a la cuenta
  saldo:= saldo - 1000;
  V(saldo)
}
```

PROGRAMA CONCURRENTE –
Ejemplo 1

Suponga que una pareja comparte su cuenta bancaria (saldo).
En un mismo momento ambos integrantes van al cajero para extraer 1000 pesos.

Variable compartida **saldo**

```
Integrante 1:
{ accede a la cuenta
  P(saldo)
  saldo:= saldo - 1000;
  V(saldo)
}
```

```
Integrante 2:
{ accede a la cuenta
  P(saldo)
  saldo:= saldo - 1000;
  V(saldo)
}
```

PROGRAMA CONCURRENTE –
Ejemplo 2

En un programa existen 3 procesos y un arreglo V de longitud M.
Se quiere calcular cuantas veces aparece el valor N en el arreglo.

¿El código es correcto?

Variable compartida **cont y V**

¿Se puede mejorar?

```
Proceso 1:
{inf:=…; sup:= …;
 P(cont)
 for i:= inf to sup do
  if v[i] = N then
     cont:= cont + 1;
 V(cont)
}
```

```
Proceso 2:
{inf:=…; sup:= …;
 P(cont)
 for i:= inf to sup do
  if v[i] = N then
     cont:= cont + 1;
 V(cont)
}
```

```
Proceso 3:
{inf:=…; sup:= …;
 P(cont)
 for i:= inf to sup do
  if v[i] = N then
     cont:= cont + 1;
 V(cont)
}
```

PROGRAMA CONCURRENTE –
Ejemplo 2

En un programa existen 3 procesos y un arreglo V de longitud M.
Se quiere calcular cuantas veces aparece el valor N en el arreglo.

Variable compartida **cont y V**

¿Se puede mejorar?

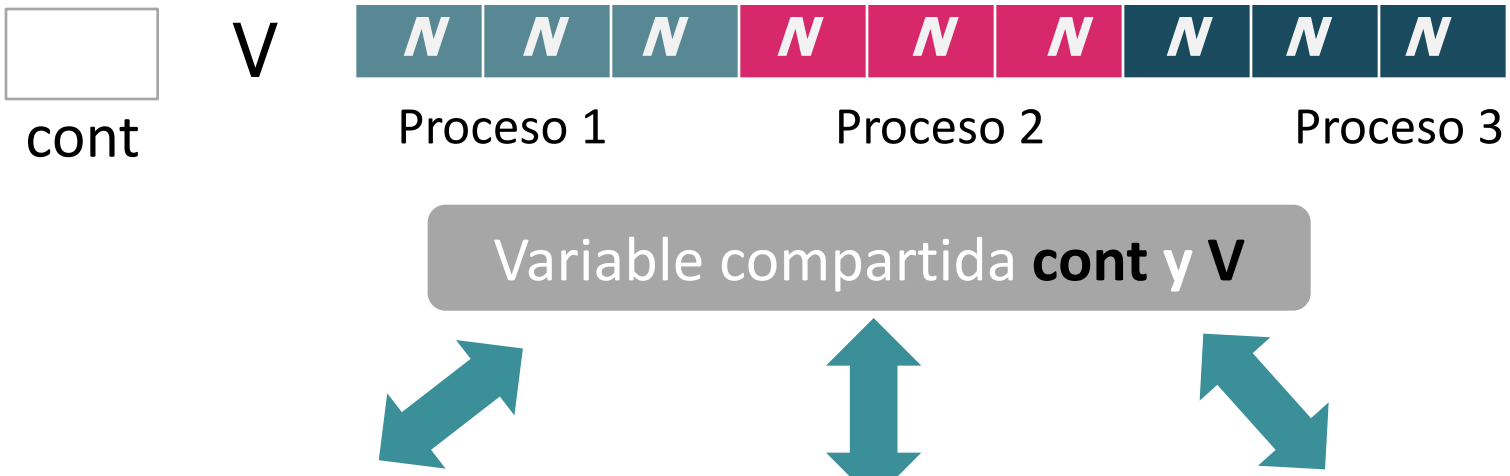
```
Proceso 1:
{inf:=…; sup:= …;
 for i:= inf to sup do
  if v[i] = N then
     P(cont)
     cont:= cont + 1;
     V(cont)
}
```

```
Proceso 2:
{inf:=…; sup:= …;
 for i:= inf to sup do
  if v[i] = N then
     P(cont)
     cont:= cont + 1;
     V(cont)
}
```

```
Proceso 3:
{inf:=…; sup:= …;
 for i:= inf to sup do
  if v[i] = N then
     P(cont)
     cont:= cont + 1;
     V(cont)
}
```

PROGRAMA CONCURRENTE –
Ejemplo 2

En un programa existen 3 procesos y un arreglo V de longitud M.
Se quiere calcular cuantas veces aparece el valor N en el arreglo.




```
Proceso 1:
{inf:=…; sup:=…; tot:=0;
 for i:= inf to sup do
  if v[i] = N then
    tot:= tot + 1;
 P(cont)
 cont:= cont + tot;
 V(cont)
}
```

```
Proceso 2:
{inf:=…; sup:=…; tot:=0;
 for i:= inf to sup do
  if v[i] = N then
    tot:= tot + 1;
 P(cont)
 cont:= cont + tot;
 V(cont)
}
```

```
Proceso 3:
{inf:=…; sup:= …; tot:=0;
 for i:= inf to sup do
  if v[i] = N then
    tot:= tot + 1;
 P(cont)
 cont:= cont + tot;
 V(cont)
}
```

PROGRAMA CONCURRENTE –
MECANISMOS DE COMUNICACIÓN Y SINCRONIZACIÓN

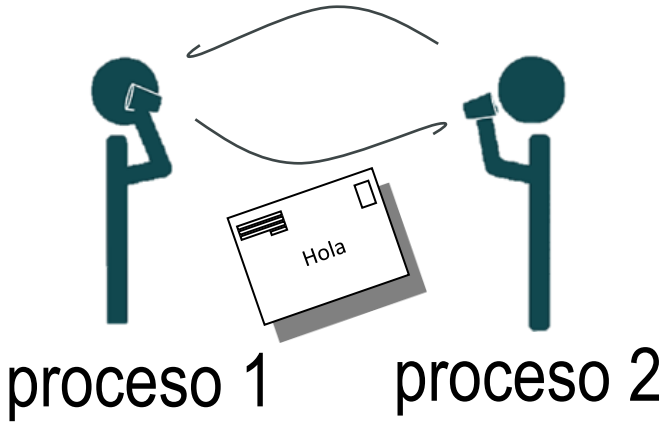
proceso 1

proceso 2

PASO DE MENSAJES

MEMORIA COMPARTIDA

PROGRAMA CONCURRENTE –
PASO DE MENSAJES

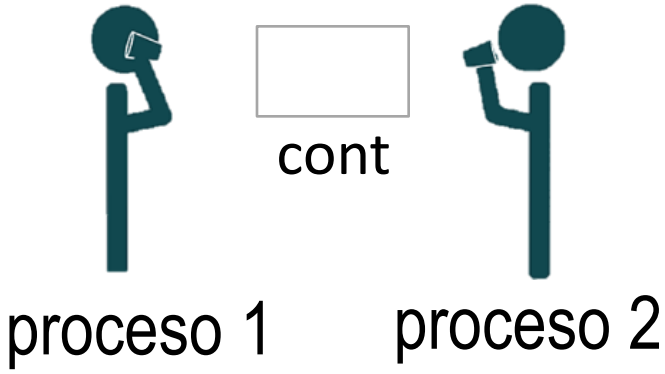


- Los procesos comparten canales (lógico o físico) para transmitirse información.
- El lenguaje debe proveer un protocolo adecuado.

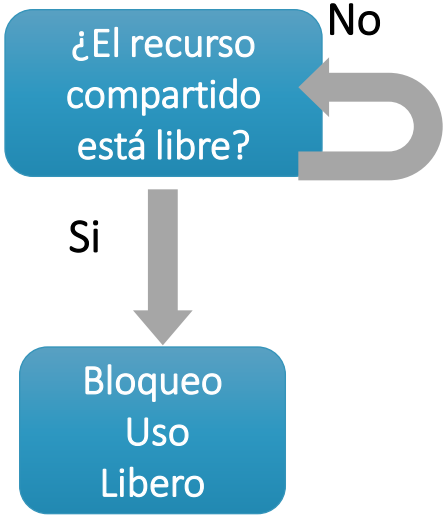
ENVIAR
RECIBIR

- Para que la comunicación sea efectiva los procesos deben “saber” cuándo tienen mensajes para leer y cuando deben “transmitir” mensajes.

PROGRAMA CONCURRENTE – MEMORIA COMPARTIDA



Protocolo:



- Los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre datos residentes en ella.
- Lógicamente NO pueden operar simultáneamente sobre la memoria compartida, lo que obliga a bloquear y liberar el acceso a la memoria.
- La solución más elemental es una “variable de control” que habilite o no el acceso de un proceso a la memoria compartida.

BLOQUEAR
LIBERAR