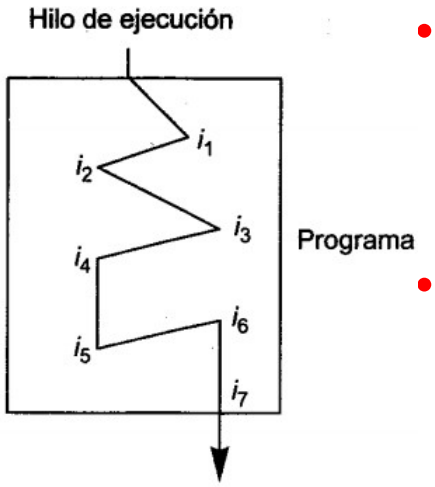
# Algoritmos y Programación Paralela

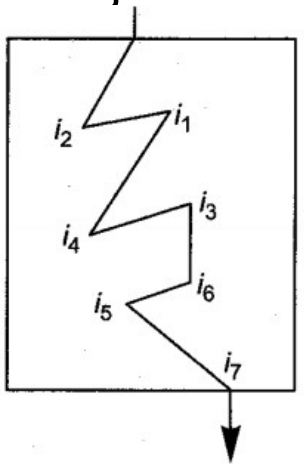
Dra. Ing. Ana Cori Morón

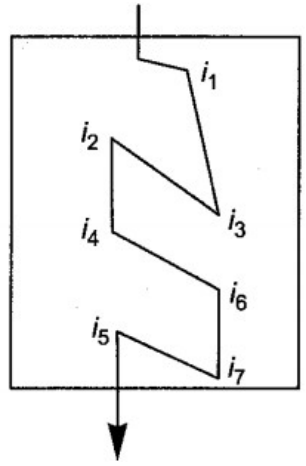
CLASE 3

# Características de los sistemas concurrentes




- **Orden de ejecución de las instrucciones**:
  - En los programas secuenciales hay un orden total, se sabe siempre por donde va a ir el programa, es decir cual va a ser su flujo de ejecución, el hilo de ejecución siempre tendrá el mismo recorrido, es decir las instrucciones siempre se ejecutarán en el mismo orden.
  - En los programas concurrentes hay un orden parcial, no se puede saber cual va a ser el flujo de ejecución, en cada ejecución del programa, el flujo puede ir por distinto sitio.



# Características de los sistemas concurrentes

- **Indeterminismo**, este orden parcial lleva a que los programas concurrentes puedan tener un **comportamiento indeterminista**, es decir, **puedan arrojar diferentes resultados cuando se ejecutan repetidamente sobre la misma entrada**.

```
Program Incognita;
Var x:integer;

Process P1;
  var i:integer;
Begin
  for i:=1 to 5 do x:=x+1;
End;

Process P2;
  var j:integer;
Begin
  for j:=1 to 5 do x:=x+1;
End;
```

- ¿Qué valor tendrá X al finalizar el programa?
- El error aquí es el acceso incontrolado a una variable compartida por parte de 2 procesos

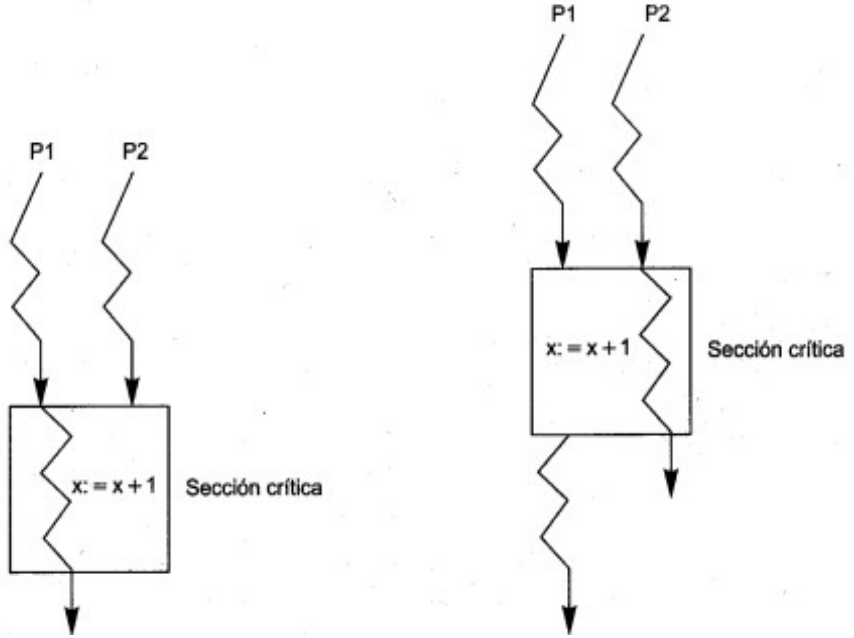
# Problemas inherentes a la programación concurrente

- **Exclusión mutua**, veamos el ejemplo:
- Sea $S_1$: x=x+1 y $S_2$= x=x+1
- El proceso P1 y P2 , no se pueden ejecutar concurrentemente.
- **El problema es que dos procesos distintos están accediendo al mismo tiempo a una variable compartida entre los dos para actualizarla.**
- A la porción de código que queremos que se ejecute de forma indivisible y atómica se le denomina sección crítica.
- Es importante que las secciones críticas se ejecuten en exclusión mutua, es decir solo uno de los procesos debe estar en la sección critica en un instante del tiempo.

Suponer que $S_1$ se ejecuta en $P_1$ y $S_2$ se ejecuta en $P_2$

# Problemas inherentes a la programación concurrente

- **<u>Exclusión mutua,</u>**

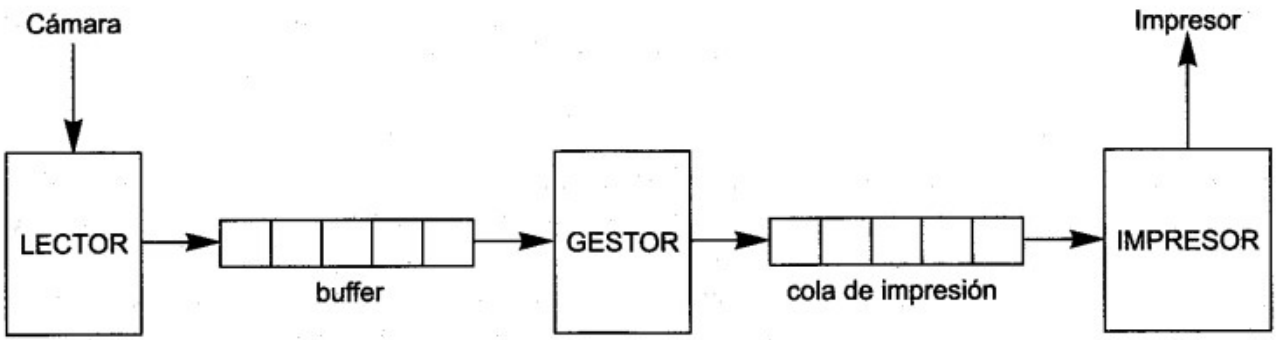



Ejecución de las Instrucciones es atómica

- La programación concurrente nos ofrece mecanismos para especificar que parte del código ha de ejecutarse en exclusión mutua.

# Problemas inherentes a la programación concurrente

- **<u>Condición de la sincronización</u>**, veamos el ejemplo de lector-gestor-impresor.




- ¿Qué ocurre cuando el proceso lector trata de poner una imagen y el buffer está lleno?
- ¿Qué ocurre cuando el proceso gestor trata de poner una imagen y la cola de impresión está llena?
- ¿Qué ocurre cuando el proceso gestor trata de coger una imagen y el buffer está vacío?
- ¿Qué ocurre cuando el proceso impresor trata de coger una imagen y la cola está vacía?

# Problemas inherentes a la programación concurrente

- En el **problema de Condición de la sincronización**, hay situaciones en las que un recurso compartido por varios procesos, se encuentra en un estado en el que el proceso no puede hacer una determinada acción con el recurso compartido hasta que no cambie su estado.
- El recurso compartido para este ejemplo es el buffer y la cola.
- La programación concurrente nos ofrece mecanismos para bloquear procesos que no puedan hacer algo en un momento determinado a la espera de algún evento, pero que también permita desbloquearlos cuando el evento haya ocurrido.

# Condiciones de Bernstein

- ¿Qué se puede ejecutar concurrentemente?
  - No todas las partes de un programa se pueden ejecutar concurrentemente.
- Considere el siguiente fragmento de programa:
  - X=X+1;
  - Y=X+2;
- Se tiene que ejecutar la primera instrucción o acción antes que la segunda.
- Ahora veamos el siguiente ejemplo:
  - X=1;
  - Y=2;
  - Z=3;
- El orden en que se ejecuten no altera el resultado final.

# Condiciones de Bernstein

- El sentido común ayudó a resolver ambos ejemplos.
- Permiten determinar si dos conjuntos de instrucciones se pueden ejecutar concurrentemente.
- $L(S_k)=\{a_1,a_2,\ldots,a_n\}$ es el conjunto de lectura del conjunto de instrucciones $S_k$.
- $E(S_k)=\{b_1,b_2,\ldots,b_n\}$ es el conjunto de escritura del conjunto de instrucciones $S_k$.
- Bernstein definió tres condiciones para determinar si dos conjuntos de instrucciones $S_i$ y $S_j$, se pueden ejecutar concurrentemente.

$$L(S_i) \cap E(S_j) = \Phi$$
$$E(S_i) \cap L(S_j) = \Phi$$
$$E(S_i) \cap E(S_j) = \Phi$$

# Condiciones de Bernstein

- Conjunto de condiciones formales que determinan si dos procesos (segmentos de programa) pueden ser ejecutados en paralelo.
- L(S1) es el conjunto de variables cuyo valor es accedido para consultar durante la ejecución de la instrucción.
- E(S1) es el conjunto de variables cuyo valor cambia durante la ejecución de la instrucción.
- Si se verifican las tres condiciones, P1 y P2 pueden ejecutarse en paralelo, esto lo denotamos como : P1 || P2

# Ejemplo

- Sea el siguiente grupo de instrucciones y se desea averiguar que instrucciones se pueden eiecutar concurrentemente:

$$S1 \rightarrow a := x + y;$$
$$S2 \rightarrow b := z - 1;$$
$$S3 \rightarrow c := a - b;$$
$$S4 \rightarrow w := c + 1;$$

- Identificamos el conjunto de variables de lectura y escritura para las instrucciones S1, S2, S3, S4

$$L(S_1) = \{x, y\} \qquad E(S_1) = \{a\}$$
$$L(S_2) = \{z\} \qquad E(S_2) = \{b\}$$
$$L(S_3) = \{a, b\} \qquad E(S_3) = \{c\}$$
$$L(S_4) = \{c\} \qquad E(S_4) = \{w\}$$

# Ejemplo

- Aplicamos las condiciones de Bernstein:

$$L(S_1) = \{x, y\} \qquad E(S_1) = \{a\}$$
$$L(S_2) = \{z\} \qquad E(S_2) = \{b\}$$
$$L(S_3) = \{a, b\} \qquad E(S_3) = \{c\}$$
$$L(S_4) = \{c\} \qquad E(S_4) = \{w\}$$

Entre $S_1$ y $S_2$:

$$L(S_1) \cap E(S_2) = \varnothing$$
$$E(S_1) \cap L(S_2) = \varnothing$$
$$E(S_1) \cap E(S_2) = \varnothing$$

Entre $S_1$ y $S_3$:

$$L(S_1) \cap E(S_3) = \varnothing$$
$$E(S_1) \cap L(S_3) = a \neq \varnothing$$
$$E(S_1) \cap E(S_3) = \varnothing$$

Entre $S_1$ y $S_4$:

$$L(S_1) \cap E(S_4) = \varnothing$$
$$E(S_1) \cap L(S_4) = \varnothing$$
$$E(S_1) \cap E(S_4) = \varnothing$$

Entre $S_2$ y $S_4$:

$$L(S_2) \cap E(S_4) = \varnothing$$
$$E(S_2) \cap L(S_4) = \varnothing$$
$$E(S_2) \cap E(S_4) = \varnothing$$

Entre $S_2$ y $S_3$:

$$L(S_2) \cap E(S_3) = \varnothing$$
$$E(S_2) \cap L(S_3) = b \neq \varnothing$$
$$E(S_2) \cap E(S_3) = \varnothing$$

Entre $S_3$ y $S_4$:

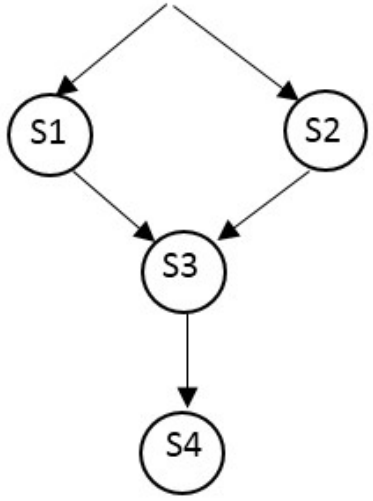
$$L(S_3) \cap E(S_4) = \varnothing$$
$$E(S_3) \cap L(S_4) = c \neq \varnothing$$
$$E(S_3) \cap E(S_4) = \varnothing$$

# Ejemplo

$$S1 \rightarrow a := x + y;$$
$$S2 \rightarrow b := z - 1;$$
$$S3 \rightarrow c := a - b;$$
$$S4 \rightarrow w := c + 1;$$

- Resumiendo los siguientes pares de instrucciones se pueden ejecutar concurrentemente:

| | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| $S_1$ | — | Sí | No | Sí |
| $S_2$ | — | — | No | Sí |
| $S_3$ | — | — | — | No |
| $S_4$ | — | — | — | — |



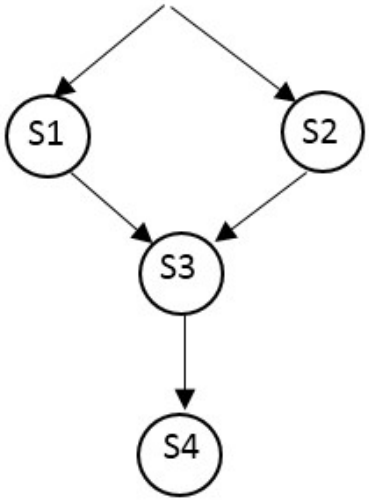
# Grafos de precedencia

- Es un grafo acíclico orientado cuyos nodos corresponden a sentencias individuales.
- Un arco de un nodo $S_i$ al nodo $S_j$ significa que la sentencia $S_j$ puede ejecutarse sólo cuando ha acabado $S_i$.
- Grafo de precedencia para el ejemplo anterior.

# Sentencias COBEGIN-COEND en PASCAL FC

```
Begin
        Cobegin
          S1, S2;
        Coend;
        S3;
        S4;
End;
```

```
Begin
        Cobegin
            a:=x+y;
            b:=z-1;
        Coend;
                c:=a-b;
                w:=c+1;
        End;
```



# ¿Cómo escribir Programas concurrentes correctos?

Para que un programa concurrente sea correcto debe cumplir con las propiedades  inherentes a la concurrencia.

- Propiedad de seguridad, son aquellas que aseguran que nada malo va a pasar durante la ejecución del programa. Las propiedades de seguridad son **exclusión mutua, condición de sincronización e interbloqueo pasivo o deadlock.**
- Propiedad de viveza, son aquellas que aseguran que algo bueno eventualmente pasará durante la ejecución del programa. Las propiedades de vivacidad son; **interbloqueo activo o livelock, inanición**.

# ¿Como escribir Programas concurrentes correctos?

- **Inanición:** Es un problema relacionado con los sistemas multitarea, donde a un proceso o un hilo de ejecución se le deniega siempre el acceso a un recurso compartido. Sin este recurso, la tarea a ejecutar no puede ser nunca finalizada.
- En el **interbloqueo**, dos procesos o dos hilos de ejecución llegan a un punto muerto cuando cada uno de ellos necesita un recurso que es ocupado por el otro.

https://www.youtube.com/watch?v=P18lFP8hT-U
Deadlock
https://www.youtube.com/watch?v=g0K3-IcjRtQ
Livelock 1.16

# TRABAJO

- ELABORAR EJEMPLOS DE EXCLUSION MUTUA Y CONDICION DE SINCRONIZACION

# PARA REVISAR

- http://ferestrepoca.github.io/paradigmas-de-programacion/paralela/paralela_teoria/index.html#eight