Introducción a la programación concurrente

Programación concurrente

Docentes

- Juan Pablo Galeotti (jgaleotti@dc.uba.ar)
- Daniel Ciolek (dciolek@dc.uba.ar)

Lista de alumnos: tpi-est-pconc@listas.unq.edu.ar

Condiciones generales

- Clases teórico/prácticas
- Evaluaciones:
  - Dos parciales
  - Un trabajo práctico

Objetivos I

Introducir problemas comunes a muchas disciplinas

- Sistemas operativos
- Sistemas distribuidos
- Sistemas de tiempo real

Comprender problemas clásicos de la programación concurrente

- Problemas de sincronización

Objetivos II

- Comprender las principales primitivas para la programación concurrente
- Desarrollar habilidades para utilizar estas primitivas para resolver problemas de sincronización
- Conocer técnicas de programación concurrente de lenguajes modernos

Bibliografía

- Doug Lea, Concurrent Programmin in Java: Design Principles and Patterns
- M. Ben-Ari, Principles of Concurrent and Distributed Programming

Introducción

- ¿Qué es un programa?
- ¿Qué es un programa concurrente?
- ¿Qué diferencia tiene la concurrencia con el paralelismo?

Introducción

- ¿Qué es un programa?
  - Un algoritmo escrito en un lenguaje de programación
- ¿Qué es un programa concurrente?
- ¿Qué diferencia tiene la concurrencia con el paralelismo?

Introducción

- ¿Qué es un programa?
  - Un algoritmo escrito en un lenguaje de programación
  - La ejecución de un programa puede describirse a través de un orden total de estados
- ¿Qué es un programa concurrente?

- ¿Qué diferencia tiene la concurrencia con el paralelismo?

Introducción

- ¿Qué es un programa?
  - Un algoritmo escrito en un lenguaje de programación
  - La ejecución de un programa puede describirse a través de un orden total de estados
- ¿Qué es un programa concurrente?
  - Un conjunto de programas secuenciales que podrían ejecutar en paralelo
- ¿Qué diferencia tiene la concurrencia con el paralelismo?

Introducción

- ¿Qué es un programa?
  - Un algoritmo escrito en un lenguaje de programación
  - La ejecución de un programa puede describirse a través de un orden total de estados
- ¿Qué es un programa concurrente?
  - Un conjunto de programas secuenciales que podrían ejecutar en paralelo
- ¿Qué diferencia tiene la concurrencia con el paralelismo?
  - Paralelismo: la ejecución de varios programas al mismo tiempo (ejecutando en distintos procesadores)

Introducción

- ¿Qué es un programa?
  - Un algoritmo escrito en un lenguaje de programación
  - La ejecución de un programa puede describirse a través de un orden total de estados
- ¿Qué es un programa concurrente?
  - Un conjunto de programas secuenciales que podrían ejecutar en paralelo
- ¿Qué diferencia tiene la concurrencia con el paralelismo?
  - Paralelismo: la ejecución de varios programas al mismo tiempo (ejecutando en distintos procesadores)
  - Concurrencia: Paralelismo potencial

¿Por qué concurrencia?

1. Utilización eficiente del procesador ante la presencia de operaciones bloqueantes (ej. operaciones de entrada salida).
2. Modelar sistemas inherentemente concurrentes

¿Por qué concurrencia?

1. Utilización eficiente del procesador ante la presencia de operaciones bloqueantes (ej. operaciones de entrada salida).
2. Modelar sistemas inherentemente concurrentes

Concurrencia

Abstracción que permite comprender el comportamiento de conjuntos de programas que comparten recursos

- Evitando considerar detalles específicos de su ejecución (cantidad de procesadores sobre los que se ejecutará)
- Basta concentrarse en único modelo de ejecución (todos los programas se ejecutan sobre un único procesador)

Terminología

- Procesos: Programas secuencial que ejecuta en su propio espacio de direcciones
- Threads: Ejecuta dentro del espacio de direcciones de un único proceso
- Estado: Datos + Puntero de instrucción
- Scheduler: Un programa del sistema operativo que decide cual es el proceso que debe ejecutar en el próximo intervalo de tiempo

Ejemplo I

Escribamos un programa secuencial que muestre el mensajes "Hola" cada 3 segundos.

Ejemplo I

Escribamos un programa secuencial que muestre el mensajes "Hola" cada 3 segundos.

```
while (true) {
  print("Hola");
  sleep(3000);
}
```

Ejemplo II

Modifiquemos el programa anterior para que también muestre el mensaje “¿Cómo estás?” cada 5 segundos.

Ejemplo II

Modifiquemos el programa anterior para que también muestre el mensaje "¿Cómo estás?" cada 5 segundos.

```
time = 0;
while (true) {
  if (time % 3 == 0)
    print("Hola");
  if (time % 5 == 0)
    print("Como estas?");
  sleep(1000);
  time++;
}
```

Solución concurrente

Ejecutar dos programas concurrentemente

Solución concurrente

Ejecutar dos programas concurrentemente

```
thread T1:
  while (true) {
    print("hola");
    sleep(3000);
  }
```

```
thread T2:
  while (true) {
    print("Como estas?");
    sleep(5000);
  }
```

Scheduling de procesos

En una máquina de Von Nuemann los threads aparentan ser ejecutados al mismo tiempo, pero en realidad su ejecución es intercalada (interleaving).

Scheduling

Tarea de alternar la ejecución de múltiples threads

- Cooperativo: Un thread ejecuta hasta que libera voluntariamente el procesador.
- Expropiativo (pre-emptive): Se interrumpe la ejecución de un thread para dar lugar a la ejecución de otro.

Estados y trazas

- Un programa ejecuta una secuencia de acciones (atómicas)
- Un estado es el valor de las variables del programa en un momento dado
- Una traza es una secuencia de estados que pueden ser producidos por un conjunto de acciones de un programa

Memoria compartida

Considerar los threads

```
global int x;

thread T1:
  x = 0;

thread T2:
  x = 1;
```

Memoria compartida

Considerar los threads

```
global int x;

thread T1:
  x = 0;

thread T2:
  x = 1;
```

- ¿Cuál es el valor de x luego de ejecutar T1?

Memoria compartida

Considerar los threads

```
global int x;

thread T1:
  x = 0;

thread T2:
  x = 1;
```

- ¿Cuál es el valor de x luego de ejecutar T1?
- ¿Cuál es el valor de x luego de ejecutar T2?

Memoria compartida

Considerar los threads

```
global int x;

thread T1:
  x = 0;

thread T2:
  x = 1;
```

- ¿Cuál es el valor de x luego de ejecutar T1?
- ¿Cuál es el valor de x luego de ejecutar T2?
- ¿Cuál es el valor de x luego de ejecutar T1 $\|$ T2?

Memoria compartida

Considerar los threads

```
global int x;

thread T1:
  x = 0;

thread T2:
  x = 1;
```

- ¿Cuál es el valor de x luego de ejecutar T1?
- ¿Cuál es el valor de x luego de ejecutar T2?
- ¿Cuál es el valor de x luego de ejecutar T1 $\|$ T2?
  $\{x = 0, x = 1\}$ Más de un valor posible!

Ejemplo

Considerar los threads

```
global int x;

thread T1: {
  x = 1;
}

thread T2: {
  x = 0;
  x = x + 1;
}
```

Ejemplo

Considerar los threads

```
global int x;

thread T1: {
  x = 1;
}

thread T2: {
  x = 0;
  x = x + 1;
}
```

Notar que dejan en x el mismo valor

Ejemplo

Considerar los threads

```
global int x;

thread T1: {
  x = 1;
}

thread T2: {
  x = 0;
  x = x + 1;
}
```

Notar que dejan en x el mismo valor
¿Qué sucede con T1 || T1 y con T2 || T2?

Ejemplo

Considerar los threads

```
global int x;

thread T1: {
  x = 1;
}

thread T2: {
  x = 0;
  x = x + 1;
}
```

Notar que dejan en x el mismo valor
¿Qué sucede con T1 $\|$ T1 y con T2 $\|$ T2?

- Los resultados posibles de T1 $\|$ T1 son $\{x = 1\}$

Ejemplo

Considerar los threads

```
global int x;

thread T1: {
  x = 1;
}

thread T2: {
  x = 0;
  x = x + 1;
}
```

Notar que dejan en x el mismo valor
¿Qué sucede con T1 $\|$ T1 y con T2 $\|$ T2?

- Los resultados posibles de T1 $\|$ T1 son $\{x = 1\}$
- Los resultados posibles de T2 $\|$ T2 son $\{x = 1, x = 2\}$

Ejemplo

Considerar los threads

```
global int x;

thread T1: {
  x = 1;
}

thread T2: {
  x = 0;
  x = x + 1;
}
```

Notar que dejan en x el mismo valor
¿Qué sucede con T1 $\|$ T1 y con T2 $\|$ T2?

- Los resultados posibles de T1 $\|$ T1 son $\{x = 1\}$
- Los resultados posibles de T2 $\|$ T2 son $\{x = 1, x = 2\}$
- No son equivalentes! Los threads se comunican a través de x de formas distintas

Concurrencia

- No determinismo
- Describir la interacción/comunicación
- El resultado no siempre es interesante
- Hay interés en programas que no terminan

Propiedades

Una propiedad es una fórmula lógica verdadera para toda traza posible.

- Safety: Una traza nunca alcanza un estado “malo”
- Liveness: Tarde o temprano, toda traza alcanza un estado “bueno”

Propiedades

Una propiedad es una fórmula lógica verdadera para toda traza posible.

- Safety: Una traza nunca alcanza un estado "malo"
- Liveness: Tarde o temprano, toda traza alcanza un estado "bueno"

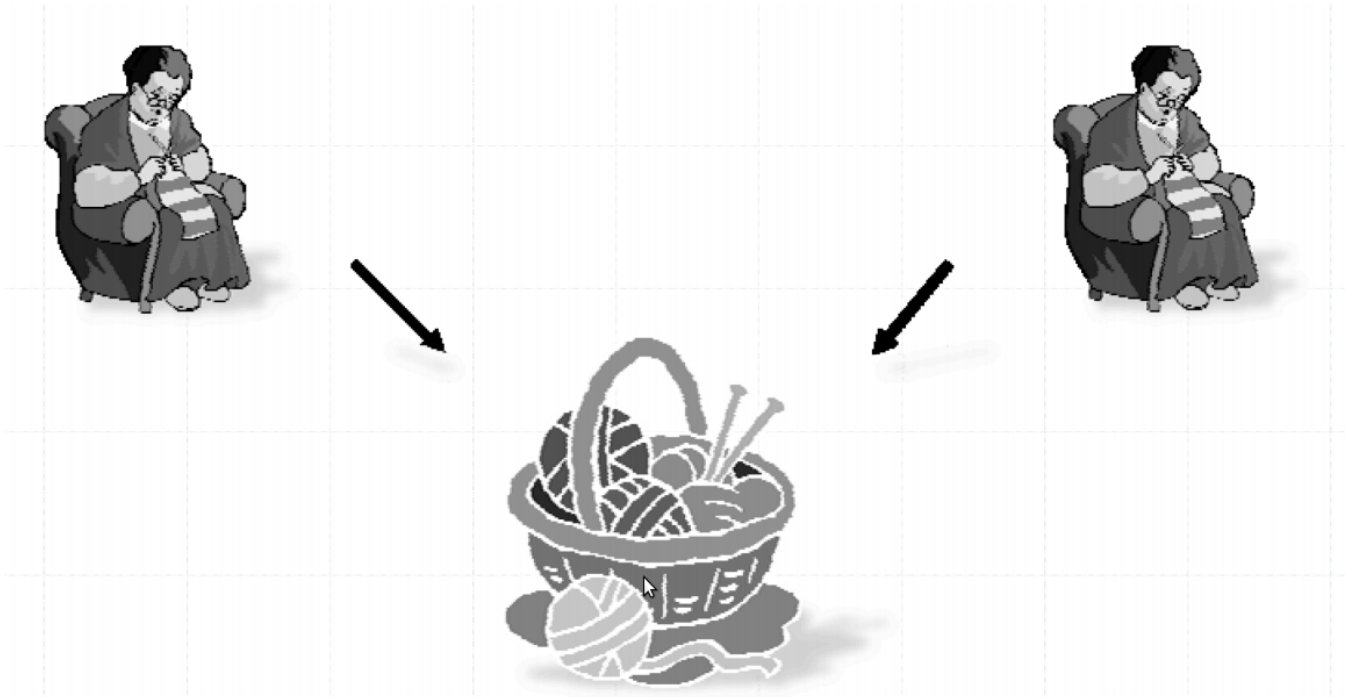
Sincronización

Mecanismo para restringir las posibles trazas de un programa concurrente con el fin de asegurar propiedades de safety y liveness.

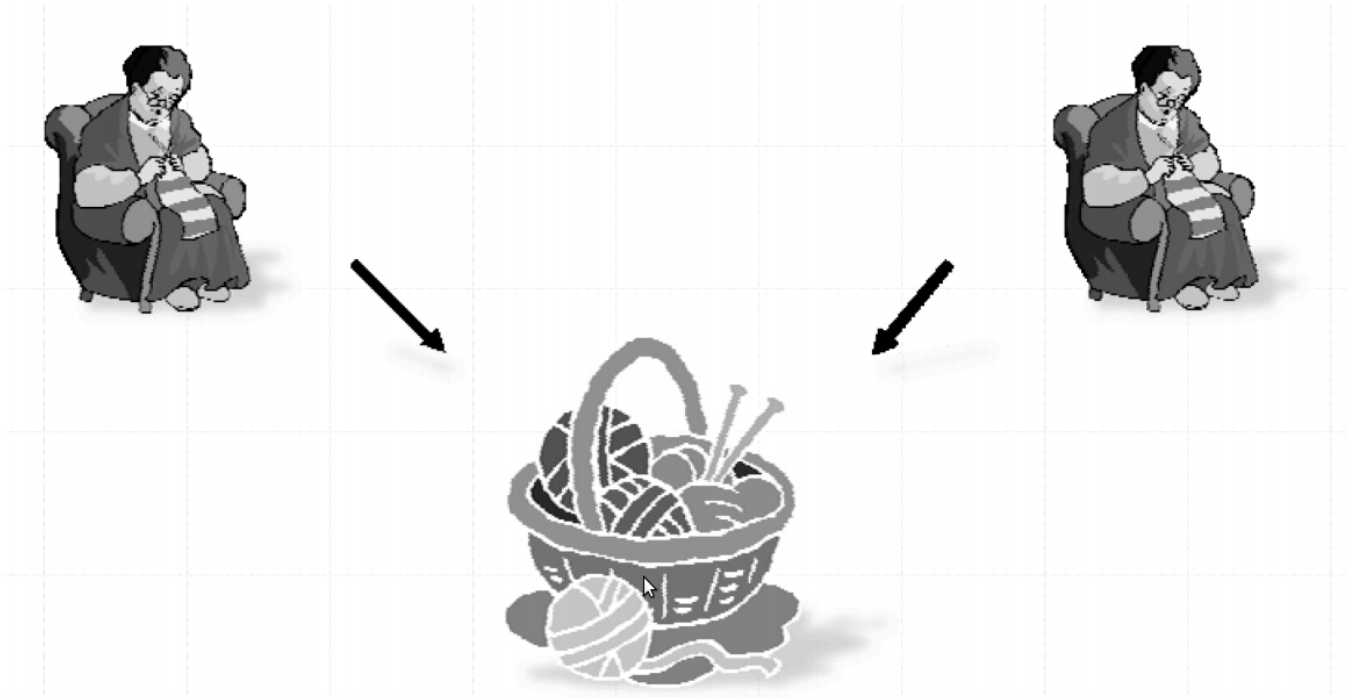
Procesos independientes

No hay recursos compartidos ni comunicación con lo cual no hay problemas ni cooperación.

Procesos competitivos

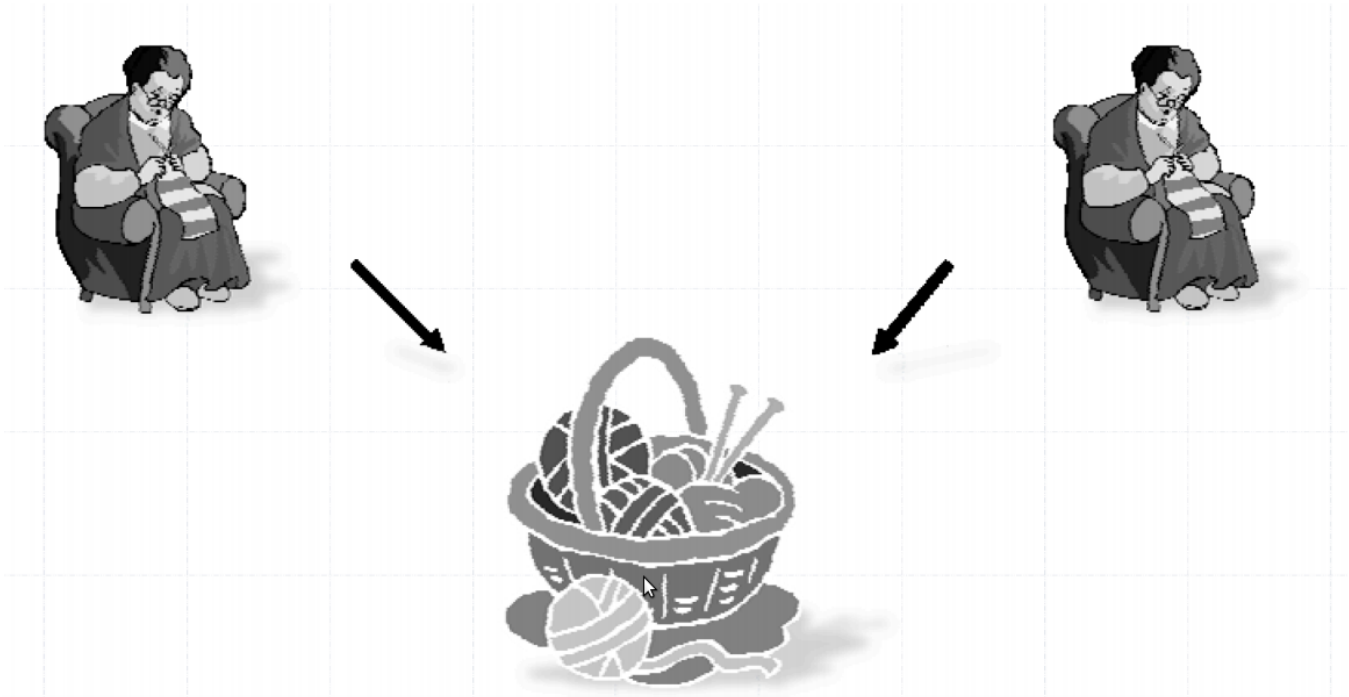


Procesos competitivos



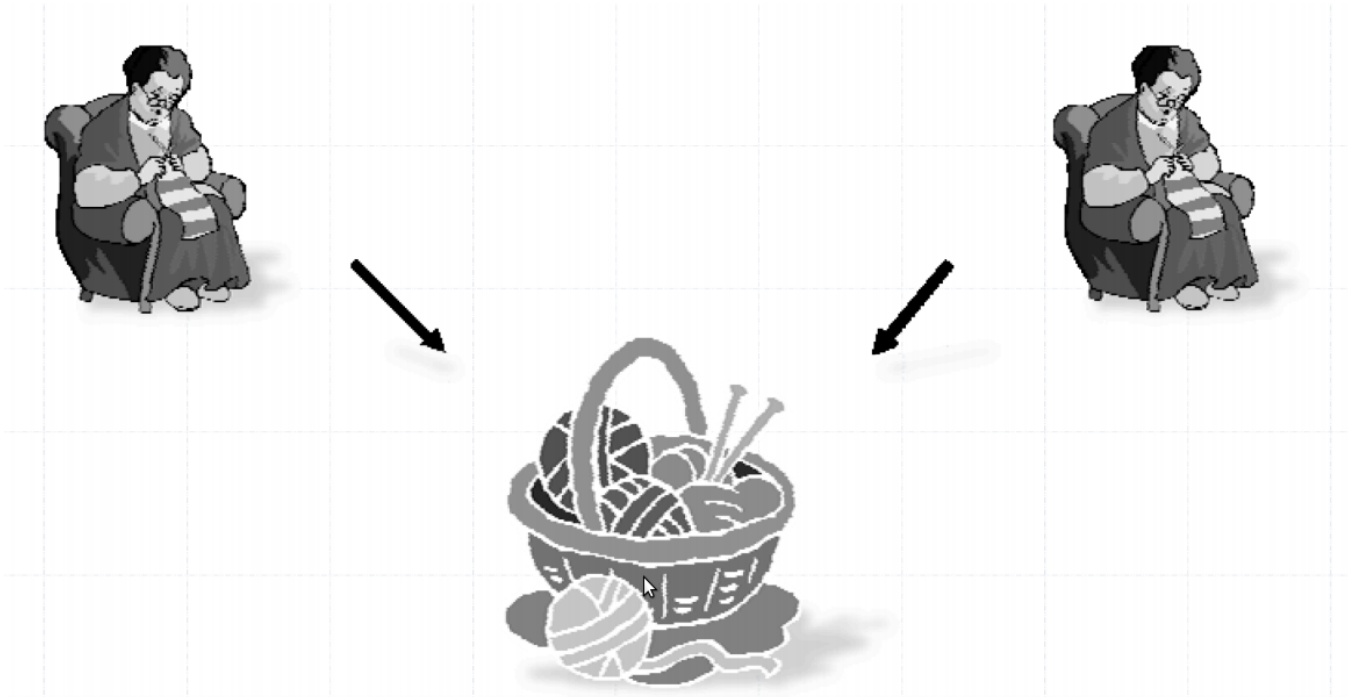
- Deadlock: cada abuela toma una aguja y espera indefinidamente hasta que la otra libere la que tiene.

Procesos competitivos



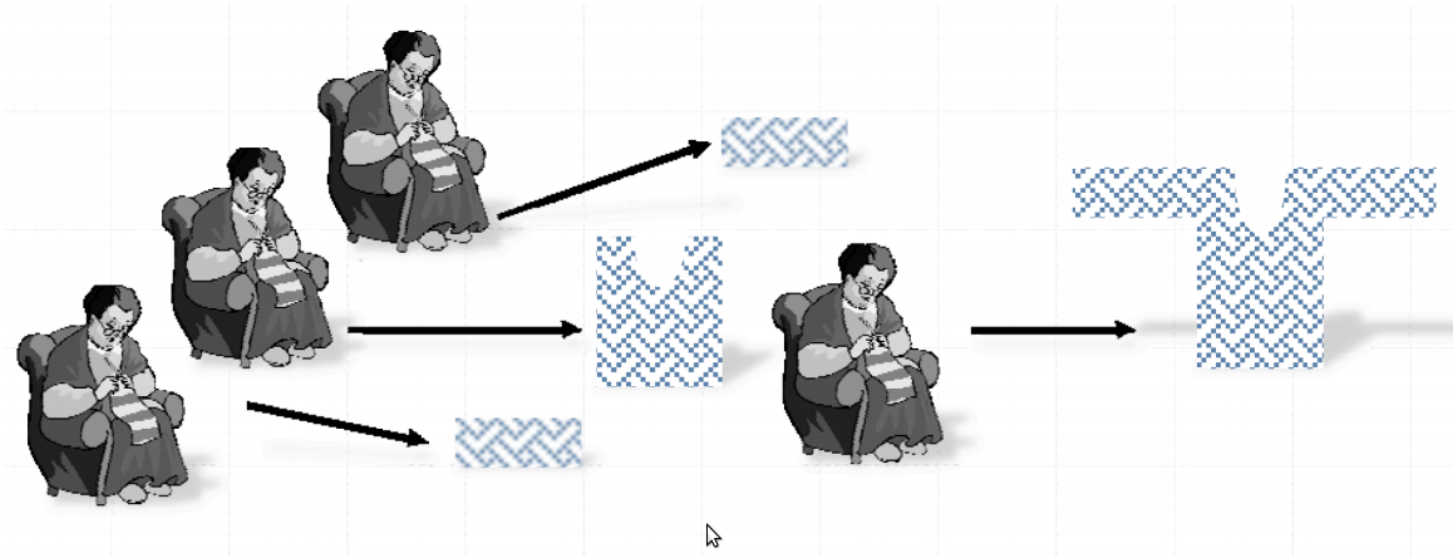
- Deadlock: cada abuela toma una aguja y espera indefinidamente hasta que la otra libere la que tiene.
- Livelock: cada abuela toma una aguja, ve que la otra tiene la otra y la deja (se repite indefinidamente).

Procesos competitivos

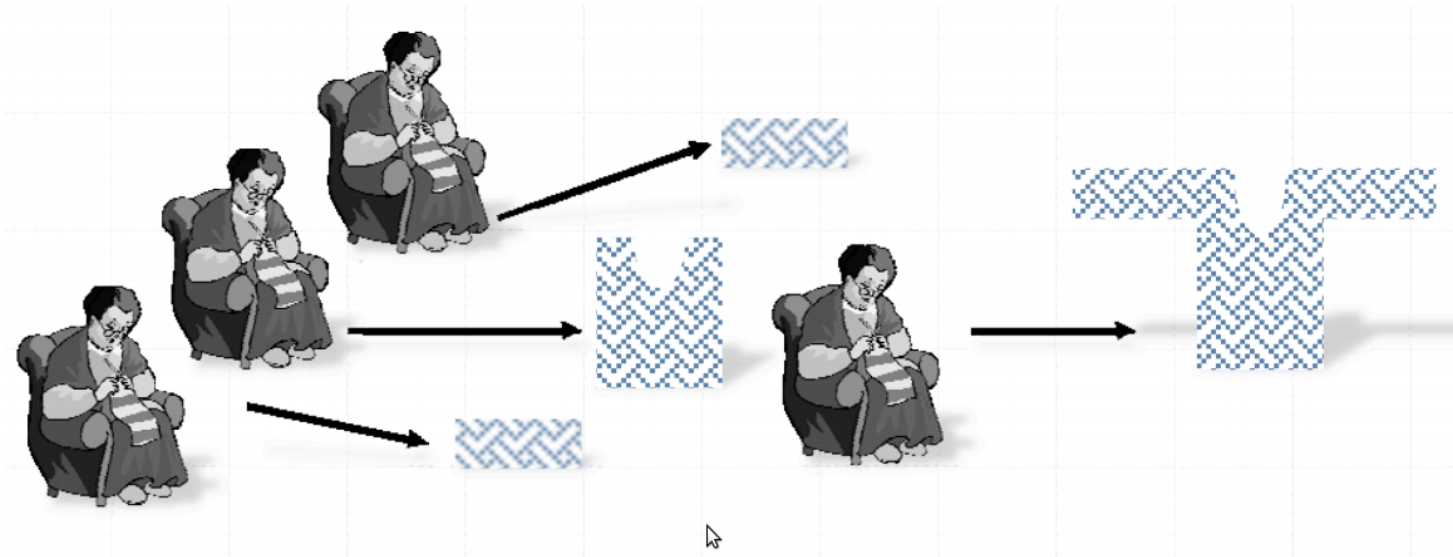


- Deadlock: cada abuela toma una aguja y espera indefinidamente hasta que la otra libere la que tiene.
- Livelock: cada abuela toma una aguja, ve que la otra tiene la otra y la deja (se repite indefinidamente).
- Starvation: una de las abuelas siempre toma las agujas antes que la otra.

Procesos cooperativos



Procesos cooperativos



- Son necesarios mecanismos de comunicación para que pueda lograrse una cooperación.

Aclaración

No se puede asumir la velocidad de ejecución como mecanismo de sincronización.

```
thread T1: {
  sleep(1000);
  x = 1;
}

thread T2: {
  x = 2;
}
```

Sigue teniendo dos resultados posibles.

Resumen

- Necesitamos la concurrencia para aprovechar el procesador durante los tiempos muertos de la ejecución de un thread.

Resumen

- Necesitamos la concurrencia para aprovechar el procesador durante los tiempos muertos de la ejecución de un thread.
- Los programas concurrentes son no-deterministicos.

Resumen

- Necesitamos la concurrencia para aprovechar el procesador durante los tiempos muertos de la ejecución de un thread.
- Los programas concurrentes son no-determinísticos.
- En la materia estudiaremos distintos mecanismos de sincronización que nos permitirán controlar el comportamiento de los programas concurrentes.

Resumen

- Necesitamos la concurrencia para aprovechar el procesador durante los tiempos muertos de la ejecución de un thread.
- Los programas concurrentes son no-determinísticos.
- En la materia estudiaremos distintos mecanismos de sincronización que nos permitirán controlar el comportamiento de los programas concurrentes.
- En particular utilizaremos los mecanismos de sincronización para garantizar que nuestros programas cumplen con propiedades deseables.