Introducción a la programación concurrente

Programación concurrente

Información General

Clases teórico-prácticas: Martes de 18hs a 22hs.

Docentes:

- Pablo Terlisky (Teóricas)
- Nicolás Mastropasqua (Prácticas)

Medios de contacto:

- Página de la materia en el Campus Virtual
- Lista de alumnos: tpi-est-pconc@listas.unq.edu.ar (para consultas envíennos mail directamente o ponganlas en Discord)
- Página Web de la materia: `https://sites.google.com/view/pconctpiunq/`
- Server de Discord

Objetivos

Introducir problemas comunes a muchas disciplinas

- Sistemas operativos
- Sistemas distribuidos
- Sistemas de tiempo real

Comprender problemas clásicos de la programación concurrente

- Problemas de sincronización
- Primitivas para la programación concurrente

Bibliografía:

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

Resumen y evaluación

La materia tiene 4 temas principales:

- Exclusión Mutua
- Semáforos
- Monitores
- Mensajes

Se evalúan en 2 parciales, dos temas por parcial. Los parciales se toman en horario de clase. (3 horas)

La calificación y recuperación es por tema.

Luego de los parciales se debe realizar un TP con nota aprobado/desaprobado.

Se puede promocionar con promedio 7 (con mínimo 6 en cada tema) y el TP aprobado.

Entornos de programación

En general los ejercicios son en pseudocódigo, pero haremos algunos experimentos en un lenguaje de la materia llamado **Hydra**.

Basado en Java, provee algunas herramientas y *syntactic sugar* para facilitar la escritura de programas concurrentes.

No es perfecto pero se puede utilizar para hacer pequeñas pruebas.

Ya está en la página de la materia (ejecutar con una máquina virtual Java: `java -jar hydra.jar`).

Para el TP se utilizará Java.

Introducción

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

Introducción

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

Introducción

- ¿Qué es un programa?
  - Descripción ejecutable.
  - 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?
  - Descripción ejecutable.
  - 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?
  - Descripción ejecutable.
  - 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?
  - Descripción ejecutable.
  - 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

- **Estado:** Datos (valores en memoria) + Puntero de instrucción.
- **Proceso:** Programa que ejecuta en su propio espacio de direcciones.
- **Threads:** Programas secuenciales que ejecutan dentro del espacio de direcciones de un proceso.
- **Scheduler:** Un módulo del sistema operativo que decide cual es el proceso que ejecutará en el próximo intervalo de tiempo.

Ejemplo I

Escribamos un programa secuencial que muestre el mensaje “Hola” cada 3 segundos.

Ejemplo I

Escribamos un programa secuencial que muestre el mensaje “Hola” cada 3 segundos.

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

Ejemplo II

Modifiquemos el programa anterior para que también muestre el mensaje “Adiós” cada 5 segundos.

Ejemplo II

Modifiquemos el programa anterior para que también muestre el mensaje “Adiós” cada 5 segundos.

```
time = 0;
while (true) {
  if (time % 3 == 0)
    print("Hola");
  if (time % 5 == 0)
    print("Adios");
  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("Adios");
    sleep(5000);
  }
```

Scheduling de procesos

En una máquina de Von Neumann 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 operaciones.
- 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.

Operaciones

Operación atómica

Una operación es llamada atómica si se ejecuta de forma indivisible, es decir, que el proceso ejecutándola no puede ser desplazado (por el scheduler) hasta que se completa su ejecución.

Operaciones

Operación atómica

Una operación es llamada atómica si se ejecuta de forma indivisible, es decir, que el proceso ejecutándola no puede ser desplazado (por el scheduler) hasta que se completa su ejecución.

Las operaciones atómicas son la unidad más pequeña en la que se puede listar una traza, pues no hay *interleavings* posibles en tales operaciones.

Trazas I

¿Cuáles son todos las trazas posibles del siguiente programa? (Asumiendo que la operación `print` es atómica)

```
thread Jose: {
  print("Hola Don Pepito");
}
```

```
thread Pepito: {
  print("Hola Don Jose");
}
```

Trazas II

¿Cuáles son todas las trazas posibles del siguiente programa? (Asumiendo que la operación `print` es atómica)

```
thread Jose: {
  print("Hola Don Pepito");
  print("Adios Don Pepito");
}
```

```
thread Pepito: {
  print("Hola Don Jose");
  print("Adios Don Jose");
}
```

Memoria compartida

Considerar los threads (y la lectura y escritura de `int` atómicas):

```
global int x;

thread T1:
  x = 0;

thread T2:
  x = 1;
```

Memoria compartida

Considerar los threads (y la lectura y escritura de `int` atómicas):

```
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 (y la lectura y escritura de `int` atómicas):

```
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 (y la lectura y escritura de `int` atómicas):

```
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 (y la lectura y escritura de `int` atómicas):

```
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 individualmente 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 individualmente 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 individualmente 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 individualmente 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 individualmente 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

Aspectos a analizar en programas concurrentes

- No determinismo
- Interacción/Comunicación entre Threads
- Identificar interleavings “interesantes”
- Considerar programas que no terminan

Propiedades

Una propiedad de un programa concurrente 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 de un programa concurrente 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".

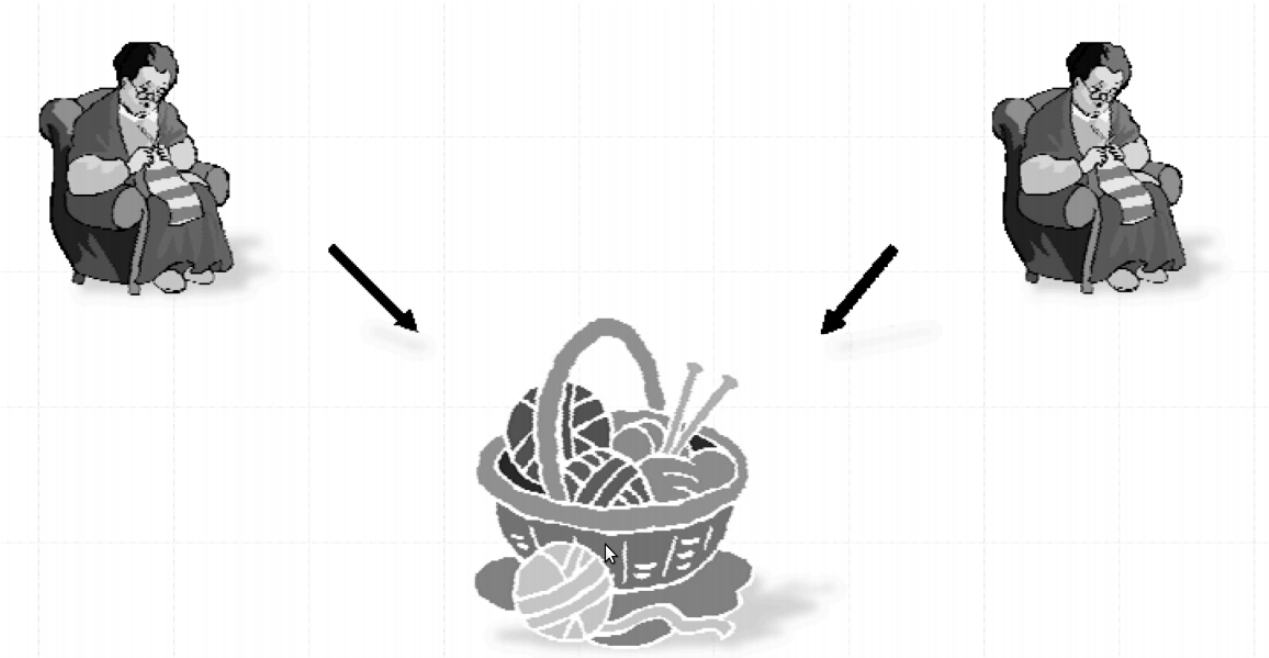
Sincronización

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

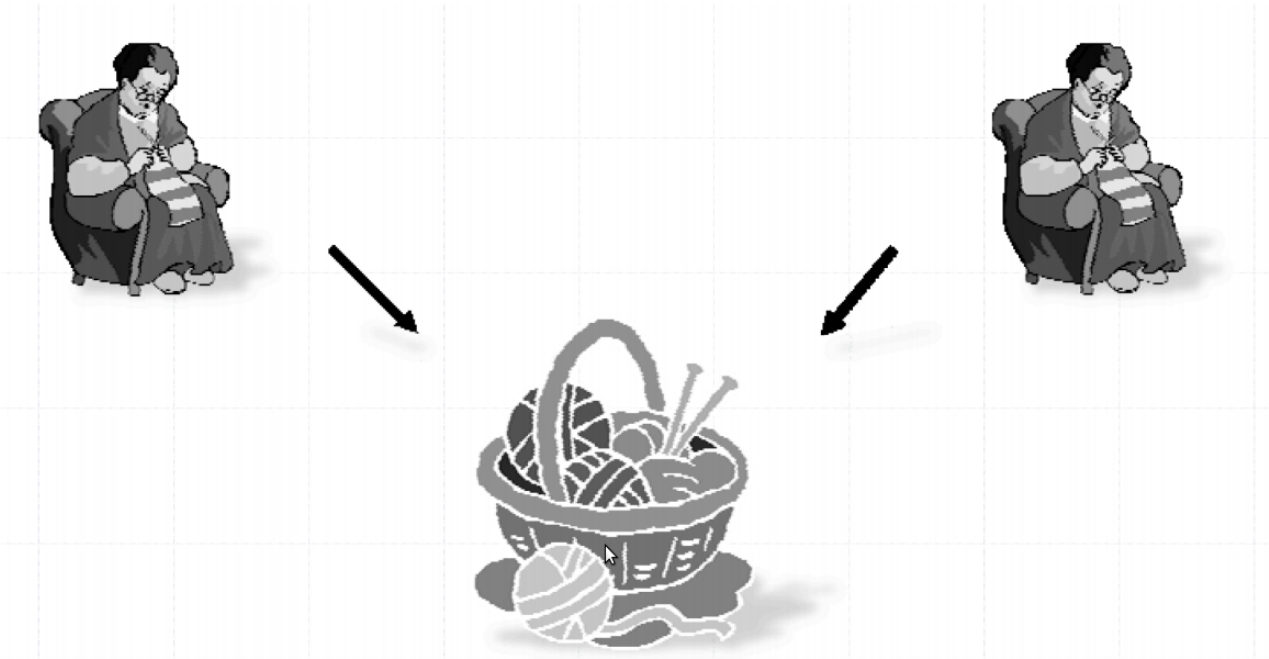
Procesos independientes

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

Procesos competitivos

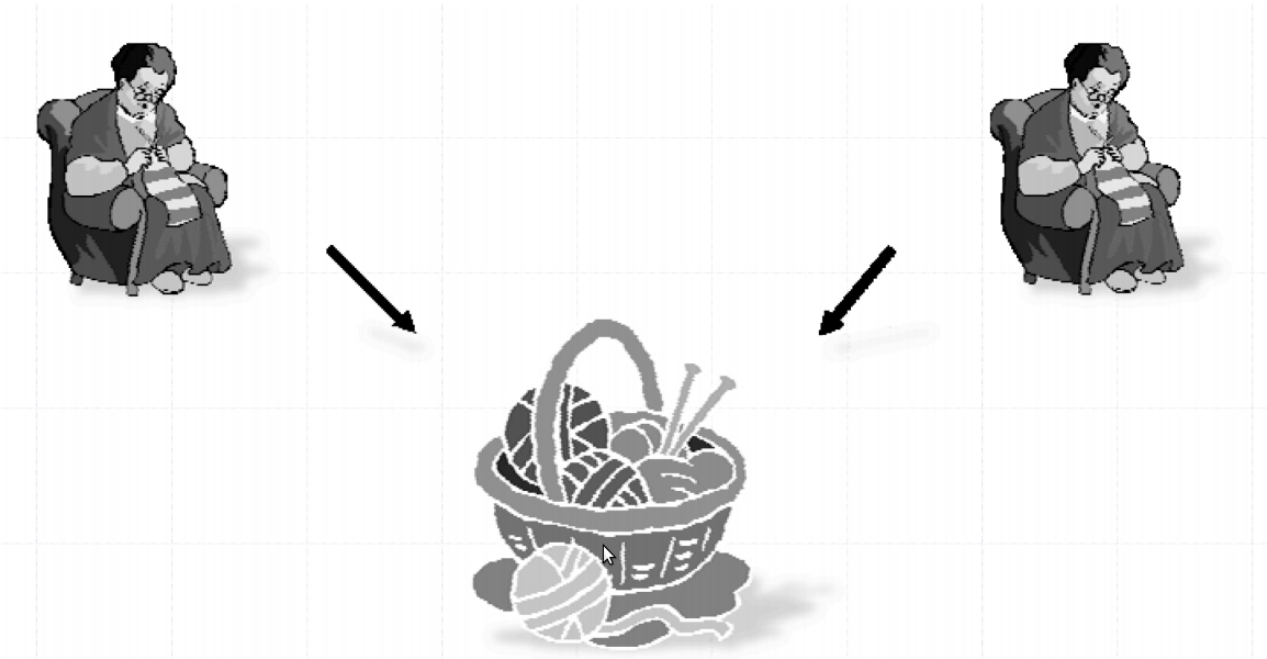


Procesos competitivos



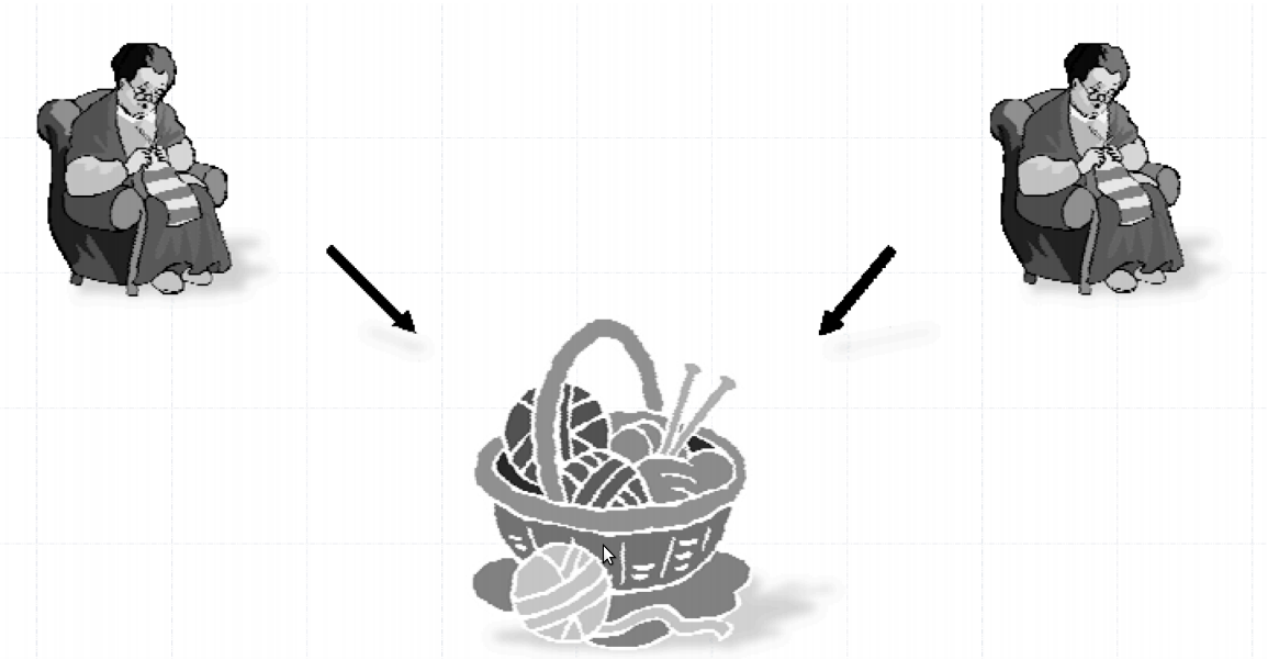
- **Deadlock:** cada abuela toma una aguja y espera indefinidamente hasta que la otra libere la que tiene.

Procesos competitivos



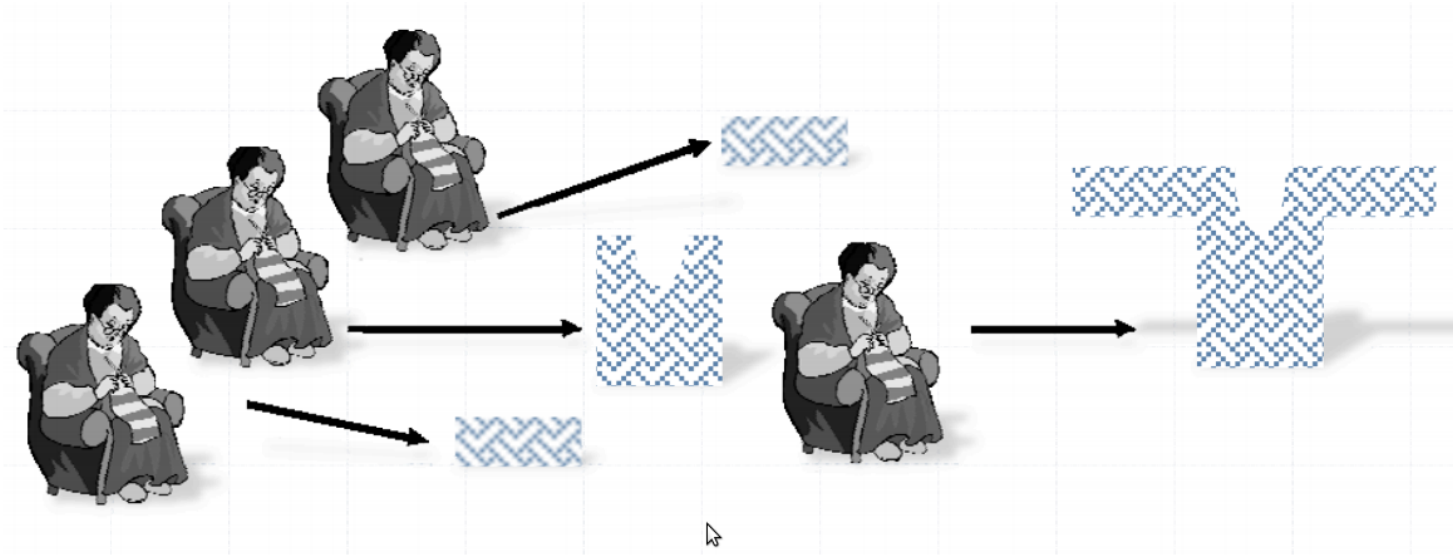
- **Deadlock:** cada abuela toma una aguja y espera indefinidamente hasta que la otra libere la que tiene.
- **Livelock:** cada abuela toma una aguja, ve que la otra tiene la otra y la deja (se repite indefinidamente).

Procesos competitivos

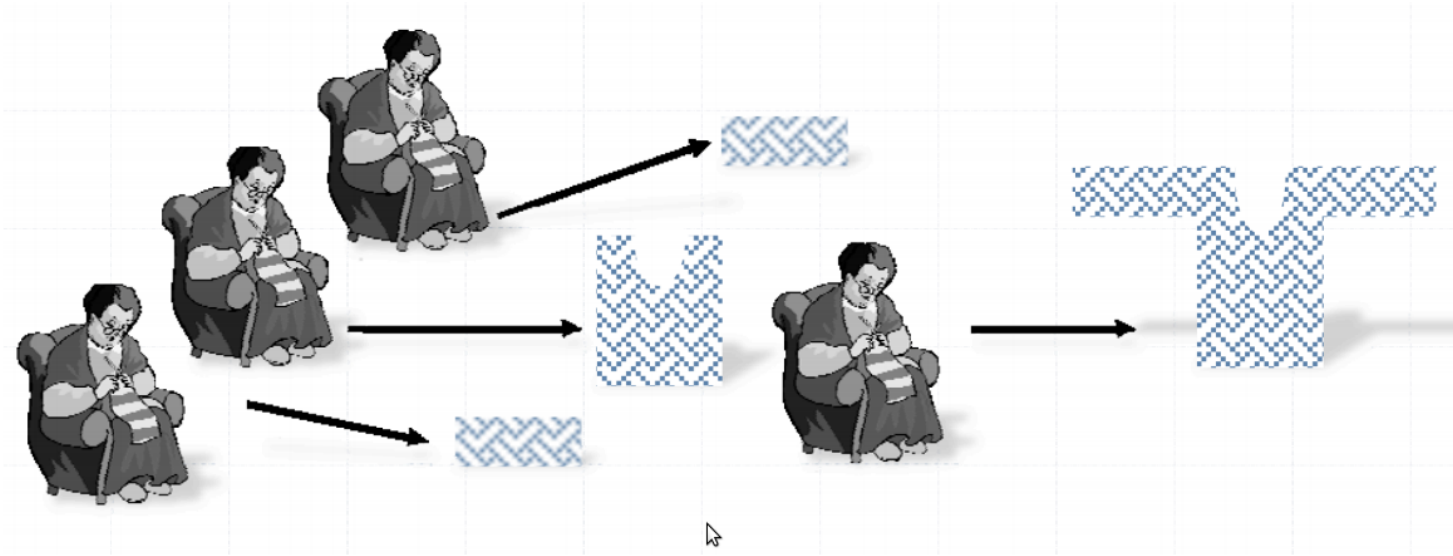


- **Deadlock:** cada abuela toma una aguja y espera indefinidamente hasta que la otra libere la que tiene.
- **Livelock:** cada abuela toma una aguja, ve que la otra tiene la otra y la deja (se repite indefinidamente).
- **Starvation:** una de las abuelas nunca toma las agujas porque siempre otra llega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

Procesos cooperativos



Procesos cooperativos



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

Aclaración

Considere el siguiente programa:

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

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

¿Cuantos resultados posibles tiene?

Aclaración

Considere el siguiente programa:

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

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

¿Cuantos resultados posibles tiene?

- Sigue teniendo dos resultados posibles.

Aclaración

Considere el siguiente programa:

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

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

¿Cuantos resultados posibles tiene?

- Sigue teniendo dos resultados posibles.
- Conclusión: No se puede asumir la velocidad de ejecución como mecanismo de sincronización.

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-deterministícos.

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.