

Exclusión mutua

Programación concurrente

Hasta ahora vimos:

Programas secuenciales:

Programas concurrentes:

Hasta ahora vimos:

Programas secuenciales:

- ▶ Existe una **única** traza de ejecución.

Programas concurrentes:

- ▶ Existe un **conjunto** de trazas de ejecución.

Hasta ahora vimos:

Programas secuenciales:

- ▶ Existe una **única** traza de ejecución.
- ▶ Existe un **único** estado final.

Programas concurrentes:

- ▶ Existe un **conjunto** de trazas de ejecución.
- ▶ Existe un **conjunto** de estados finales.

Hasta ahora vimos:

Programas secuenciales:

- ▶ Existe una **única** traza de ejecución.
- ▶ Existe un **único** estado final.
- ▶ Criterio de correctitud sobre el estado final.

Programas concurrentes:

- ▶ Existe un **conjunto** de trazas de ejecución.
- ▶ Existe un **conjunto** de estados finales.
- ▶ Nos interesan propiedades de **safety** y **liveness**.

Hasta ahora vimos:

- ▶ Los programas concurrentes pueden hacer un uso eficiente de los recursos computacionales.

Hasta ahora vimos:

- ▶ Los programas concurrentes pueden hacer un uso eficiente de los recursos computacionales.
- ▶ Sin embargo, en un modelo con memoria compartida *interleavings* indeseables pueden generar resultados erróneos.

Ejemplo

N threads suman 1 a un contador compartido e imprimen el valor:

```
global int counter = 0;

repeat (N)
  thread {
    counter = counter + 1;
    print(counter);
  }
```


Ejemplo

N threads suman 1 a un contador compartido e imprimen el valor:

```
global int counter = 0;

repeat (N)
  thread {
    counter = counter + 1;
    print(counter);
  }
```

Ejemplo de traza indeseable:

T1 -> counter + 1 ----> { counter = 0 }

Ejemplo

N threads suman 1 a un contador compartido e imprimen el valor:

```
global int counter = 0;

repeat (N)
  thread {
    counter = counter + 1;
    print(counter);
  }
```

Ejemplo de traza indeseable:

```
T1 -> counter + 1 ----> { counter = 0 }
T2 -> counter + 1 ----> { counter = 0 }
```

Ejemplo

N threads suman 1 a un contador compartido e imprimen el valor:

```
global int counter = 0;

repeat (N)
  thread {
    counter = counter + 1;
    print(counter);
  }
```

Ejemplo de traza indeseable:

```
T1 -> counter + 1 ----> { counter = 0 }
T2 -> counter + 1 ----> { counter = 0 }
T1 -> counter = 1 ----> { counter = 1 }
```

Ejemplo

N threads suman 1 a un contador compartido e imprimen el valor:

```
global int counter = 0;

repeat (N)
  thread {
    counter = counter + 1;
    print(counter);
  }
```

Ejemplo de traza indeseable:

```
T1 -> counter + 1 ----> { counter = 0 }
T2 -> counter + 1 ----> { counter = 0 }
T1 -> counter = 1 ----> { counter = 1 }
T1 -> print(counter) -> { counter = 1 }
```

Ejemplo

N threads suman 1 a un contador compartido e imprimen el valor:

```
global int counter = 0;

repeat (N)
  thread {
    counter = counter + 1;
    print(counter);
  }
```

Ejemplo de traza indeseable:

```
T1 -> counter + 1 ----> { counter = 0 }
T2 -> counter + 1 ----> { counter = 0 }
T1 -> counter = 1 ----> { counter = 1 }
T1 -> print(counter) -> { counter = 1 }
T2 -> counter = 1 ----> { counter = 1 }
```

Ejemplo

N threads suman 1 a un contador compartido e imprimen el valor:

```
global int counter = 0;

repeat (N)
  thread {
    counter = counter + 1;
    print(counter);
  }
```

Ejemplo de traza indeseable:

```
T1 -> counter + 1 ----> { counter = 0 }
T2 -> counter + 1 ----> { counter = 0 }
T1 -> counter = 1 ----> { counter = 1 }
T1 -> print(counter) -> { counter = 1 }
T2 -> counter = 1 ----> { counter = 1 }
T2 -> print(counter) -> { counter = 1 }
```

Sección crítica

Llamamos sección crítica a la parte del programa que accede a memoria compartida y que deseamos que sea ejecutada atómicamente.

Definición

Llamamos exclusión mutua (o mutex) al problema de asegurar que dos *threads* no ejecuten una sección crítica simultáneamente.

Asumiendo:

- ▶ No hay variables compartidas entre sección crítica y no crítica.
- ▶ La sección crítica siempre termina.
- ▶ El *scheduler* es débilmente *fair*.

Requerimientos de la exclusión mutua

1. **Mutex:** En cualquier momento hay como máximo un proceso en la región crítica.

Requerimientos de la exclusión mutua

1. **Mutex:** En cualquier momento hay como máximo un proceso en la región crítica.
2. **Ausencia de deadlocks y livelocks:** Si varios procesos intentan entrar a la sección crítica alguno lo logrará.

Requerimientos de la exclusión mutua

1. **Mutex:** En cualquier momento hay como máximo un proceso en la región crítica.
2. **Ausencia de deadlocks y livelocks:** Si varios procesos intentan entrar a la sección crítica alguno lo logrará.
3. **Garantía de entrada:** Un proceso intentando entrar a su sección crítica tarde o temprano lo logrará.

Esquema general

```
global variable compartida;
```

```
thread T:
```

```
    seccion no critica // no usa la variable compartida
```

```
    entrada a la seccion critica
```

```
    SECCION CRITICA    // puede usar la variable compartida
```

```
    salida de la seccion critica
```

```
    seccion no critica // no usa la variable compartida
```

Pregunta

¿Podemos resolver el problema de la exclusión mutua para dos procesos asumiendo que las únicas operaciones atómicas son la **lectura** y la **escritura** en de variables?

Intento I

```
global boolean flag = false;

thread { //                                thread {
    // seccion no critica                // seccion no critica
    while (flag);                        while (flag);
    flag = true;                         flag = true;
    // SECCION CRITICA                   // SECCION CRITICA
    flag = false;                       flag = false;
    // seccion no critica                // seccion no critica
}
```

Intento I

```
global boolean flag = false;

thread { //                thread {
    // seccion no critica    // seccion no critica
    while (flag);            while (flag);
    flag = true;             flag = true;
    // SECCION CRITICA        // SECCION CRITICA
    flag = false;            flag = false;
    // seccion no critica    // seccion no critica
}
```

- ▶ **Mutex:**
- ▶ **Ausencia dead/live-locks:**
- ▶ **Garantía de entrada:**

Intento I

```
global boolean flag = false;

thread { //                thread {
    // seccion no critica    // seccion no critica
    while (flag);            while (flag);
    flag = true;             flag = true;
    // SECCION CRITICA       // SECCION CRITICA
    flag = false;            flag = false;
    // seccion no critica    // seccion no critica
}
```

- ▶ **Mutex:** No
- ▶ **Ausencia dead/live-locks:**
- ▶ **Garantía de entrada:**

Intento I

```
global boolean flag = false;

thread { //                thread {
    // seccion no critica    // seccion no critica
    while (flag);            while (flag);
    flag = true;             flag = true;
    // SECCION CRITICA        // SECCION CRITICA
    flag = false;            flag = false;
    // seccion no critica    // seccion no critica
}
```

- ▶ **Mutex:** No
- ▶ **Ausencia dead/live-locks:** Sí
- ▶ **Garantía de entrada:**

Intento I

```
global boolean flag = false;

thread { //                thread {
    // seccion no critica    // seccion no critica
    while (flag);            while (flag);
    flag = true;              flag = true;
    // SECCION CRITICA        // SECCION CRITICA
    flag = false;             flag = false;
    // seccion no critica    // seccion no critica
}
```

- ▶ **Mutex:** No
- ▶ **Ausencia dead/live-locks:** Sí
- ▶ **Garantía de entrada:** Sí

Intento II

```
global boolean[] flags = {false, false};

thread {
    id = 0;
    // seccion no critica
    otro = (id + 1) % 2;
    flags[id] = true;
    while (flags[otro]);
    // SECCION CRITICA
    flags[id] = false;
    // seccion no critica
}

thread {
    id = 1;
    // seccion no critica
    otro = (id + 1) % 2;
    flags[id] = true;
    while (flags[otro]);
    // SECCION CRITICA
    flags[id] = false;
    // seccion no critica
}
```

Intento II

```
global boolean[] flags = {false, false};

thread {
    id = 0;
    // seccion no critica
    otro = (id + 1) % 2;
    flags[id] = true;
    while (flags[otro]);
    // SECCION CRITICA
    flags[id] = false;
    // seccion no critica
}

thread {
    id = 1;
    // seccion no critica
    otro = (id + 1) % 2;
    flags[id] = true;
    while (flags[otro]);
    // SECCION CRITICA
    flags[id] = false;
    // seccion no critica
}
```

- ▶ **Mutex:**
- ▶ **Ausencia dead/live-locks:**
- ▶ **Garantía de entrada:**

Intento II

```
global boolean[] flags = {false, false};

thread {
    id = 0;
    // seccion no critica
    otro = (id + 1) % 2;
    flags[id] = true;
    while (flags[otro]);
    // SECCION CRITICA
    flags[id] = false;
    // seccion no critica
}

thread {
    id = 1;
    // seccion no critica
    otro = (id + 1) % 2;
    flags[id] = true;
    while (flags[otro]);
    // SECCION CRITICA
    flags[id] = false;
    // seccion no critica
}
```

- ▶ **Mutex:** Sí
- ▶ **Ausencia dead/live-locks:**
- ▶ **Garantía de entrada:**

Intento II

```
global boolean[] flags = {false, false};

thread {
    id = 0;
    // seccion no critica
    otro = (id + 1) % 2;
    flags[id] = true;
    while (flags[otro]);
    // SECCION CRITICA
    flags[id] = false;
    // seccion no critica
}

thread {
    id = 1;
    // seccion no critica
    otro = (id + 1) % 2;
    flags[id] = true;
    while (flags[otro]);
    // SECCION CRITICA
    flags[id] = false;
    // seccion no critica
}
```

- ▶ **Mutex:** Sí
- ▶ **Ausencia dead/live-locks:** No
- ▶ **Garantía de entrada:**

Intento II

```
global boolean[] flags = {false, false};

thread {
    id = 0;
    // seccion no critica
    otro = (id + 1) % 2;
    flags[id] = true;
    while (flags[otro]);
    // SECCION CRITICA
    flags[id] = false;
    // seccion no critica
}

thread {
    id = 1;
    // seccion no critica
    otro = (id + 1) % 2;
    flags[id] = true;
    while (flags[otro]);
    // SECCION CRITICA
    flags[id] = false;
    // seccion no critica
}
```

- ▶ **Mutex:** Sí
- ▶ **Ausencia dead/live-locks:** No
- ▶ **Garantía de entrada:** Sí (salvo en caso de deadlock)

Intento III

```
global int turno = 0;
```

```
thread {  
    id = 0;  
    // seccion no critica  
    while (turno != id);  
    // SECCION CRITICA  
    turno = (id + 1) % 2;  
    // seccion no critica  
}
```

```
thread {  
    id = 1;  
    // seccion no critica  
    while (turno != id);  
    // SECCION CRITICA  
    turno = (id + 1) % 2;  
    // seccion no critica  
}
```

Intento III

```
global int turno = 0;

thread {
    id = 0;
    // seccion no critica
    while (turno != id);
    // SECCION CRITICA
    turno = (id + 1) % 2;
    // seccion no critica
}

thread {
    id = 1;
    // seccion no critica
    while (turno != id);
    // SECCION CRITICA
    turno = (id + 1) % 2;
    // seccion no critica
}
```

- ▶ **Mutex:**
- ▶ **Ausencia dead/live-locks:**
- ▶ **Garantía de entrada:**

Intento III

```
global int turno = 0;

thread {
    id = 0;
    // seccion no critica
    while (turno != id);
    // SECCION CRITICA
    turno = (id + 1) % 2;
    // seccion no critica
}

thread {
    id = 1;
    // seccion no critica
    while (turno != id);
    // SECCION CRITICA
    turno = (id + 1) % 2;
    // seccion no critica
}
```

- ▶ **Mutex: Sí**
- ▶ **Ausencia dead/live-locks:**
- ▶ **Garantía de entrada:**

Intento III

```
global int turno = 0;

thread {
    id = 0;
    // seccion no critica
    while (turno != id);
    // SECCION CRITICA
    turno = (id + 1) % 2;
    // seccion no critica
}

thread {
    id = 1;
    // seccion no critica
    while (turno != id);
    // SECCION CRITICA
    turno = (id + 1) % 2;
    // seccion no critica
}
```

- ▶ **Mutex:** Sí
- ▶ **Ausencia dead/live-locks:** Sí
- ▶ **Garantía de entrada:**

Intento III

```
global int turno = 0;

thread {
    id = 0;
    // seccion no critica
    while (turno != id);
    // SECCION CRITICA
    turno = (id + 1) % 2;
    // seccion no critica
}

thread {
    id = 1;
    // seccion no critica
    while (turno != id);
    // SECCION CRITICA
    turno = (id + 1) % 2;
    // seccion no critica
}
```

- ▶ **Mutex:** Sí
- ▶ **Ausencia dead/live-locks:** Sí
- ▶ **Garantía de entrada:** No (si el thread de id 0 falla)

Algoritmo de Dekker (II + III)

```
global int turno = 0;
global boolean[] flags = {false, false};
```

```
thread {
    id = 0;
    // seccion no critica
    otro = (id + 1) % 2;
    flags[id] = true;
    while (flags[otro])
        if (turno == otro) {
            flags[id] = false;
            while (turno != id);
            flags[id] = true;
        }

    // SECCION CRITICA

    turno = otro;
    flags[id] = false;
    // seccion no critica
}
```

```
thread {
    id = 1;
    ...
}
```

Algoritmo de Peterson

```
global int turno = 0;
global boolean[] flags = {false, false};

thread {
    id = 0;
    // seccion no critica
    otro = (id + 1) % 2;
    flags[id] = true;
    turno = otro;
    while (flags[otro] && turno == otro);

    // SECCION CRITICA

    flags[id] = false;
    // seccion no critica
}
```

Dekker y Peterson

- ▶ **Mutex:** Sí
- ▶ **Ausencia dead/live-locks:** Sí
- ▶ **Garantía de entrada:** Sí

Sólo sirven para dos procesos.

Dekker y Peterson

- ▶ **Mutex:** Sí
- ▶ **Ausencia dead/live-locks:** Sí
- ▶ **Garantía de entrada:** Sí

Sólo sirven para dos procesos.

La demostración formal excede el alcance de este curso.

Algoritmo de Bakery

```
global boolean[] pidiendoTicket = new boolean[n]; // {false,...,}
global int[] ticket = new int[n]; // {0,0,...,0}

thread {
    id = 0;
    // seccion no critica
    pidiendoTicket[id] = true;
    ticket[id] = 1 + maximum(ticket);
    pidiendoTicket[id] = false;
    for (j : range(0,n)) {
        while (pidiendoTicket[j]);
        while (ticket[j] != 0 &&
                (ticket[j] < ticket[id] ||
                 (ticket[j] == ticket[id] && j < id)));
    }

    // SECCION CRITICA

    ticket[id] = 0;
    // seccion no critica
}
```


Algoritmo de Bakery

```
global boolean[] pidiendoTicket = new boolean[n]; // {false,...}
global int[] ticket = new int[n]; // {0,0,...0}

thread {
    id = 0;
    // seccion no critica
    pidiendoTicket[id] = true;
    ticket[id] = 1 + maximum(ticket);
    pidiendoTicket[id] = false;
    for (j : range(0,n)) {
        while (pidiendoTicket[j]);
        while (ticket[j] != 0 &&
                (ticket[j] < ticket[id] ||
                 (ticket[j] == ticket[id] && j < id)));
    }

    // SECCION CRITICA

    ticket[id] = 0;
    // seccion no critica
}
```

Este algoritmo resuelve el problema para n threads.