IIC2523
Sistemas Distribuidos

Hernán F. Valdivieso López
(2025 - 2 / Clase 01)

Recordatorios importantes

1. La clase pasada revisamos con detalle las reglas de las evaluaciones.

   ¡Es su responsabilidad estar al tanto de ellas!

2. El programa del curso está disponible en el Syllabus

   ¡Es su responsabilidad leerlo!

3. Las clases primero estará disponible, como *Google Slides*, en una carpeta pública. Y finalizada la clase se publica el PDF en el Syllabus.

4. Si ven la clase en PDF. No se adelanten mientras estoy haciendo la clase 🙏 🤦‍♂️.

Base del Sistema Distribuido
Un computador

Temas de la clase

1. Introducción
2. Procesos
    a. Características
    b. Comunicación entre procesos
    c. Programación en Python
3. *Thread*
    a. Características
    b. Mecanismos de sincronización
    c. Programación en Python

Introducción

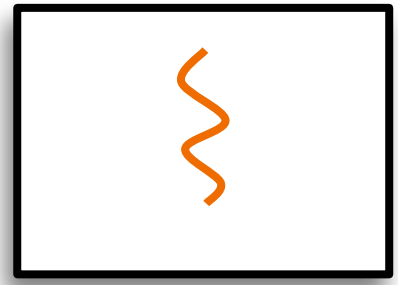
Introducción

- Antes de estudiar un sistema distribuido, necesitamos recordar cómo funciona su componente principal: un computador (o nodo).
- Dentro de cada nodo, hay diferentes procesos que se están ejecutando y cada proceso puede tener más de un hilo (*thread*) corriendo al mismo tiempo.
  - Un nodo que es cliente y servidor simultáneamente.
  - Un nodo que dispone de más de una API.
  - Un nodo que permite comunicarse con múltiples clientes simultáneamente.

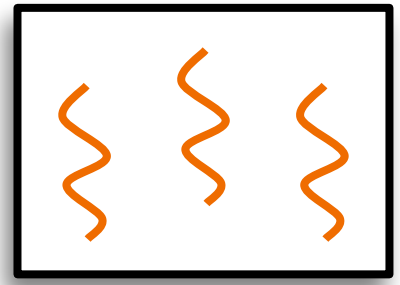
Introducción

- Antes de estudiar un sistema distribuido, necesitamos recordar cómo funciona su componente principal: un computador (o nodo).
- Dentro de cada nodo, hay diferentes procesos que se están ejecutando y cada proceso puede tener más de un hilo (*thread*) corriendo al mismo tiempo.
  - Un nodo que es cliente y servidor simultáneamente.
  - Un nodo que dispone de más de una API.
  - Un nodo que permite comunicarse con múltiples clientes simultáneamente.
- **Vamos a recordar (o aprender nuevamente) algunos conceptos claves que utilizamos en las siguientes clases.**

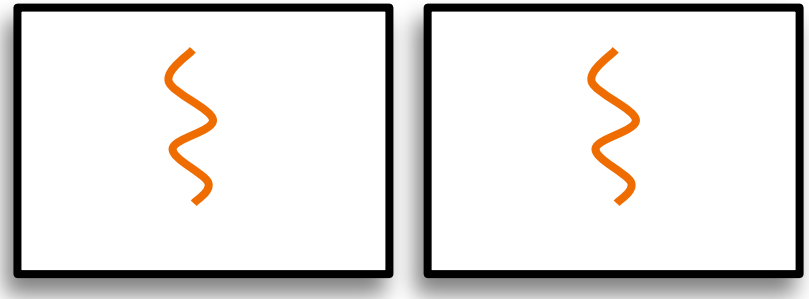
Introducción



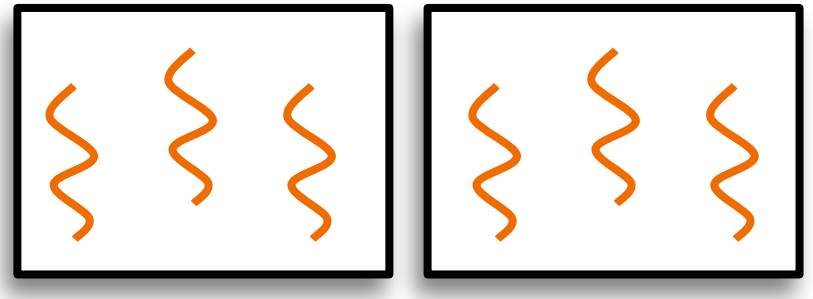
Un proceso
Un *thread*



Un proceso
Múltiples *thread*



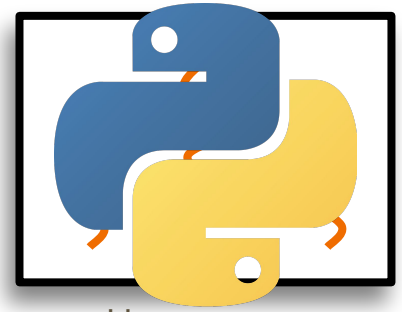
Múltiples procesos
Un *thread* por proceso



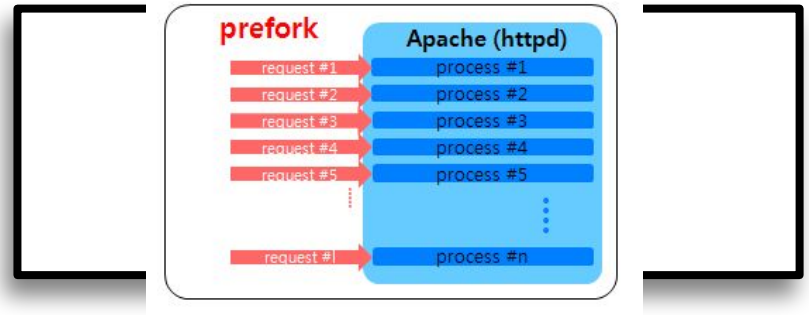
Múltiples procesos
Múltiples *thread* por proceso

Introducción

Un proceso
Un *thread*



Un proceso
Múltiples *thread*



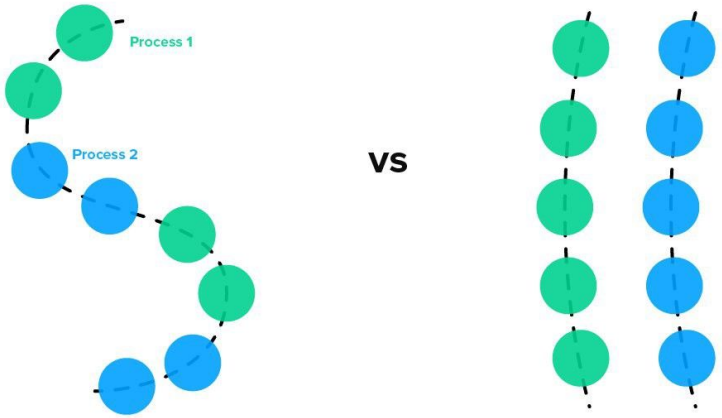
Múltiples procesos
Un *thread* por proceso

Múltiples procesos
Múltiples *thread* por proceso

Introducción

Existen 2 conceptos claves que es importante entender

- **Paralelismo** es la ejecución **simultánea** de múltiples tareas. Esto requiere de tener más de un procesador disponible.
- **Concurrencia** ejecución de múltiples tareas a la vez, alternando entre ellas para dar la **impresión que es simultáneo**.



Proceso

Características

Comunicación

Procesos en Python

Proceso

- Un proceso se puede definir como un **programa en ejecución**.
- La creación de un proceso implica asignar su propio espacio de direcciones **independiente**. Incluso si es el mismo programa ejecutado más de una vez.
- Pueden existir múltiples procesos utilizando el mismo recurso (disco duro), pero cada proceso no está conciente de eso.

Proceso - Costo

- El paso de un proceso a otro **es muy costoso**. Dado que implica múltiples operaciones realizadas por el Sistema Operativo para garantizar un correcto funcionamiento
  - Invalidar componentes del SSOO (como TLB) para asegurar correcta virtualización.
  - Guardar los registros del proceso anterior para cargar el nuevo proceso.
  - Definir el siguiente proceso a ejecutar.

Proceso - Comunicación

- Imaginemos que quiero Imprimir un documento y escanear otro, pero solo cuento con una única impresora que también puede escanear.
- Será necesario **coordinar ambos procesos** para que uno se ejecute después que el otro... pero cada proceso vive en su "propio mundo".
- Es necesario establecer un **mecanismo para comunicar los procesos**, y como no tienen, por defecto, una memoria compartida a la que ambos puedan acceder directamente, deben enviarse **mensajes explícitamente**.

Proceso - Comunicación

- La comunicación entre procesos (IPC por *Inter-Process Communication*) es una función básica de los Sistemas Operativos.
- El Sistema Operativo utiliza **diferentes mecanismos para lograr esta comunicación**.
  - *File system* para acceder al disco duro desde cualquier proceso.
  - Semáforos para controlar el acceso a recursos compartidos.
  - *Message passing* para comunicarse a otro proceso.
  - *Sockets* para generar un vía de comunicación al proceso de otro computador.
  - Memoria compartida para habilitar que 2 o más procesos usen el mismo espacio de memoria para alguna operación.

Proceso - Comunicación

- *Message passing* para comunicarse a otro proceso.
  - Proceso P construye el mensaje en su propio espacio de memoria.
  - Proceso P ejecuta un *system call* para que el Sistema Operativo mande el mensaje a Proceso Q.
  - Proceso Q recibe el mensaje.

Proceso - Comunicación

- *Message passing* para comunicarse a otro proceso.
  - Proceso P construye el mensaje en su propio espacio de memoria.
  - Proceso P ejecuta un *system call* para que el Sistema Operativo mande el mensaje a Proceso Q.
  - Proceso Q recibe el mensaje.
- Ahora solo falta un protocolo para esta comunicación...

Proceso - Comunicación

- *Message passing* para comunicarse a otro proceso.
  - Proceso P construye el mensaje en su propio espacio de memoria.
  - Proceso P ejecuta un *system call* para que el Sistema Operativo mande el mensaje a Proceso Q.
  - Proceso Q recibe el mensaje.
- Ahora solo falta un protocolo para esta comunicación...

Iremos estudiando diferentes mecanismos de comunicación, con sus propias características y regla durante las siguientes clases 😁

Proceso - Comunicación

- Con el paso de mensaje se logran comunicar los procesos, pero...
  - Queremos coordinar los procesos para ejecutarlos con cierta secuencia.
  - La información de un proceso es vital para otro, pero el proceso falló.
  - *Spoiler*: y si los procesos están en distintos computadores 🙀

Proceso - Comunicación

- Con el paso de mensaje se logran comunicar los procesos, pero...
  - Queremos coordinar los procesos para ejecutarlos con cierta secuencia.
  - La información de un proceso es vital para otro, pero el proceso falló.
  - *Spoiler*: y si los procesos están en distintos computadores 🙀
- Además del paso de mensaje, necesitamos mecanismos para coordinarlos:

Algoritmos de exclusión mutua distribuida	Protocolos de replicación de datos	Algoritmos de elección de líder
Algoritmos de consenso	Sincronizar relojes	Protocolos para hacer *commits*

Proceso - Comunicación

- Con el paso de mensaje se logran comunicar los procesos, pero...
  - Queremos coordinar los procesos para ejecutarlos con cierta secuencia.
  - La información de un proceso es vital para otro, pero el proceso falló.
  - *Spoiler*: y si los procesos están en distintos computadores 🙀
- Además del paso de mensaje, necesitamos mecanismos para coordinarlos:

Iremos estudiando estos algoritmos y protocolos durante el transcurso del semestre 😁

Algoritmos de consenso	Sincronizar relojes	Protocolos para hacer *commits*

Procesos - Python

```python
from multiprocessing import Process
import os


def saludar():
    print("Waku Waku")
    print("PID del otro proceso:", os.getpid())


if __name__ == "__main__":
    print("PID del proceso actual:", os.getpid())
    p = Process(target=saludar)
    p.start()
```

Procesos - Python

```python
import subprocess

# Asumimos que otro_script.py existe en el mismo directorio
# Run espera a que el script termine antes de continuar
subprocess.run(["python3", "otro_script.py"])

print("Script ejecutado correctamente.")
```

Procesos - Python

```python
import subprocess

# Asumimos que otro_script.py existe en el mismo directorio
# Popen no espera a que el script termine antes de continuar
# stdout=subprocess.PIPE es para capturar los print del proceso
# stderr=subprocess.PIPE es para capturar los errores del proceso
process = subprocess.Popen(["python3", "otro_script.py", "4444", "localhost"],
                           stdout=subprocess.PIPE, stderr=subprocess.PIPE)

print("Script ejecutandose correctamente.")
```

Thread

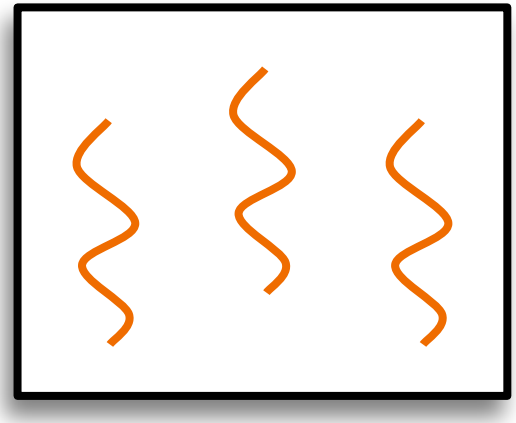
Características

Mecanismos de sincronización

***Thread* en Python**

Thread

- Es un proceso ligero o subproceso muy pequeño que puede ser ejecutada por un Sistema Operativo.
- Un *thread* se crea **dentro de un proceso** y **comparten memoria y otros recursos** del proceso creador, facilitando la comunicación entre los *threads*.



Thread - Costos

- La creación de un ***thread* es más liviana** porque no es necesario asignar un nuevo espacio de direcciones ni configurar todos los recursos como si fuera un nuevo proceso.
- En caso de fallar un *thread*, puede repercutir en los demás *thread* del proceso o incluso en todo el proceso.
  - Borrar una lista compartida por todos.
  - Modificar incorrectamente un valor.
  - Levantar una excepción que detiene el proceso.

Thread - Mecanismos de sincronización

- Se requiere **coordinar el acceso** a recursos compartidos garantizando la integridad de los datos o de operaciones.
- Existen varios mecanismos:
  - *Locks* o *Mutex*
  - Semáforos
  - Eventos
  - *Join*

Thread - Mecanismos de sincronización

Locks* o *Mutex

- Un solo *thread* tiene acceso a la vez y se hace **dueño** del *lock*.
- Solo el *thread* que toma el *lock* lo puede liberar para que otro *thread*, que espera el recurso, lo pueda solicitar.

Thread - Mecanismos de sincronización

Algoritmo de la panadería de Lamport [1974]

- Algoritmo para asegurar exclusión mediante un contador global.
- Cada *thread* es un cliente y el mostrador de la panadería como la sección crítica.
    1. Cada cliente pide un número del contador y luego se aumenta el contador global.
    2. El mostrador atiende por orden de los números.
    3. En caso que 2 o más clientes pidan número al mismo tiempo, se desempata por el identificador que es único por entidad, por ejemplo, el PID.
    4. Cuando el cliente está siendo atendido, no puede sacar otro número.
    5. Cuando el cliente termina, deja libre el mostrador para llamar al siguiente número.
- Si el Sistema Operativo habilita una memoria compartida para los procesos, este algoritmo también se puede aplicar en ellos.

Thread - Mecanismos de sincronización

Algoritmo de la panadería de Lamport [1974] (pseudocódigo)

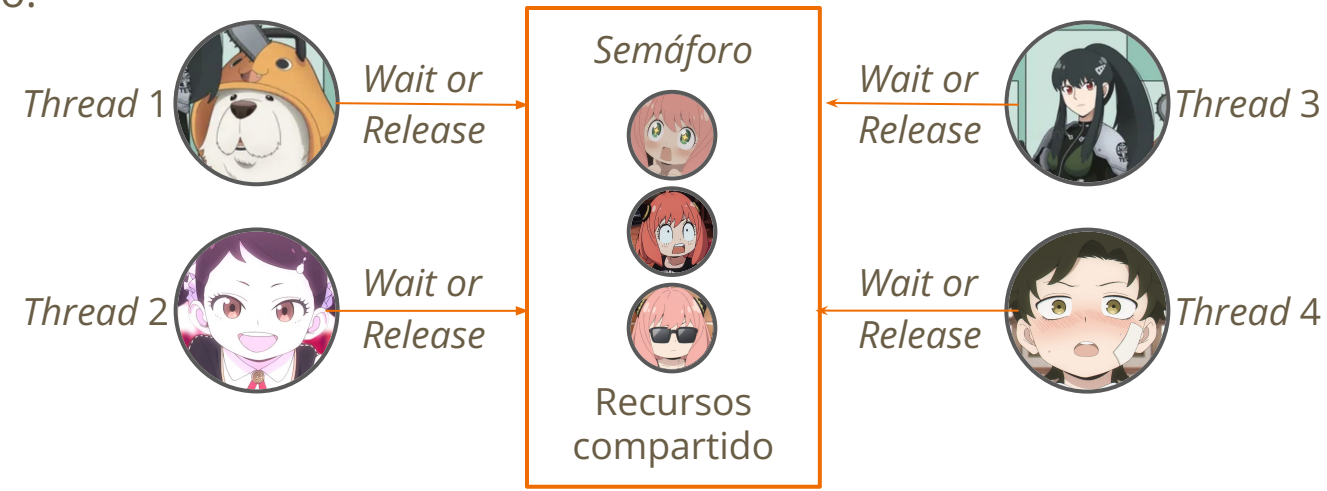
```
Entrando: array[NUM_CLIENT - 1] of bool = {false};
Número: array[NUM_CLIENT - 1] of integer = {0};

# Código cliente i
Entrando[i] = True;
Número[i] = 1 + max(Número);
Entrando[i] = false;
# Revisar todos los clientes (j)
for (integer j = 0; j <= NUM_CLIENT - 1; j++)
    # Si otro cliente está sacando número, esperar para comparar el valor correcto.
    while (Entrando[j] == True) { /* nada */ }
    # Esperar que todo cliente < i NO esté en Zona Crítica
    while (((Número[j] != 0) && (Número[j], j) < (Número[i], i))) { /* nada */ }
# Critical section...
Número[i] = 0;
# non-critical section...
```

Fuente: Pseudocódigo adaptado de A New Solution of Dijkstra's Concurrent Programming Problem (Paper de Lamport)

Thread - Mecanismos de sincronización

Semáforos

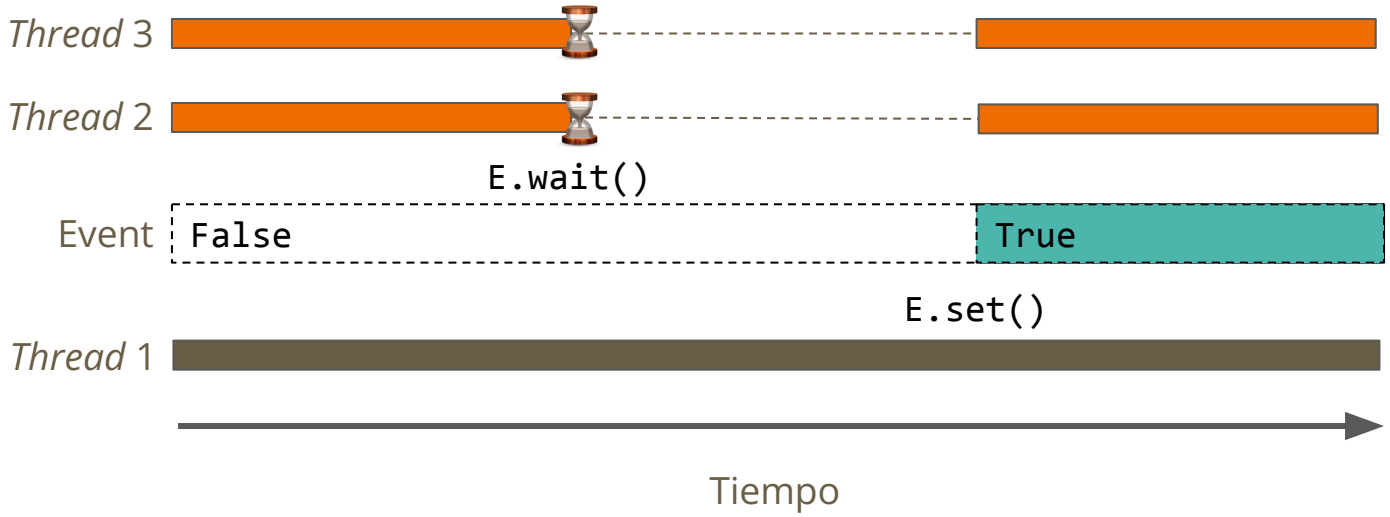
- Más de un *thread* pueden acceder y no existe un sistema de propiedad. Cualquier *thread* puede manipular el semáforo, incluso alguien que no solicitó acceso.
- Funciona mediante un contador, donde *wait* (acción para entrar) disminuye el contador y *release* aumenta el contador. Si el contador es 0, el *wait* bloquea el acceso.



Thread - Mecanismos de sincronización

Eventos

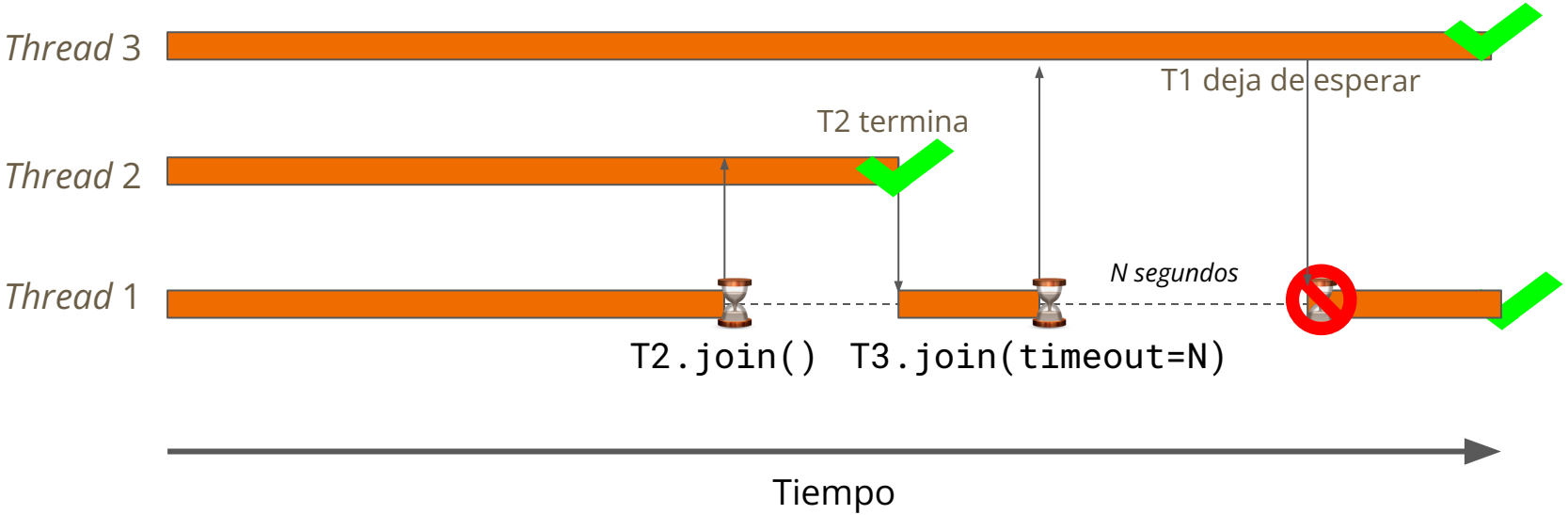
- Mecanismo para los *thread* puedan esperar a cierta acción indicada por otro *thread*.
- Cualquier *thread* puede esperar al evento y también puede empezarlo.



Thread - Mecanismos de sincronización

Join

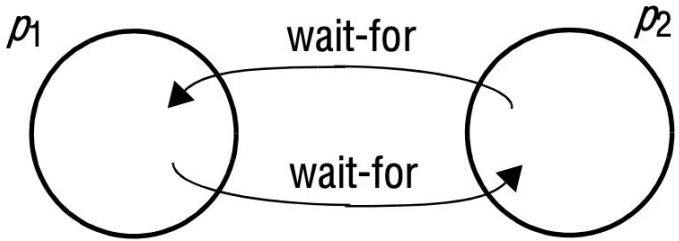
- Mecanismo para que un *thread* espere una cantidad de tiempo o que finalice otro *thread* antes de continuar con su ejecución.
- Cada *thread* solo puede esperar a un *thread* a la vez.



Thread - Mecanismos de sincronización

Deadlock o interbloqueo

*Deadlock occurs when each of a collection of processes **waits for another process** to send it a message, and where there is a **cycle** in the graph of this **'waits-for' relationship.***

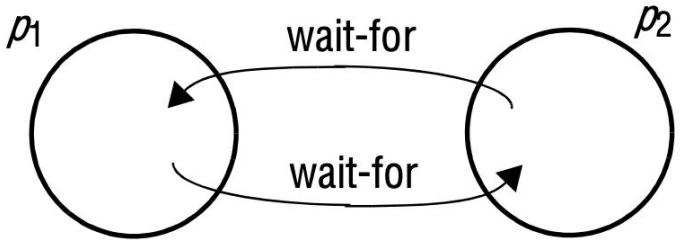


Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

Thread - Mecanismos de sincronización

Deadlock o interbloqueo

*Deadlock occurs when each of a collection of processes **waits for another process** to send it a message, and where there is a **cycle** in the graph of this **'waits-for' relationship.***



- Típicamente ocurre por un uso incorrecto de *locks*, pero también puede ocurrir con algún otro mecanismo de sincronización.

Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

Thread - Python

```python
import threading

def presentar(personaje: str, edad: int):
    nombre_thread = threading.current_thread().name
    print("Thread:", nombre_thread)
    print("Personaje:", personaje)
    print("Edad:", edad)


t1 = threading.Thread(target=presentar, name="Boby Jackson",
                      args=("Shinichi Kudo", 17))

t2 = threading.Thread(target=presentar, name="Carlos Guzman",
                      kwargs={"personaje": "Kogoro Mouri", "edad": 38})
t1.start()
t2.start()
```

Thread - Python

```python
import threading

class Presentador(threading.Thread):
    def __init__(self, nombre_thread: str, personaje: str, edad: int):
        super().__init__(name=nombre_thread)
        self.personaje = personaje
        self.edad = edad

    def run(self):
        print("Thread:", threading.current_thread().name)
        print("Personaje:", self.personaje)
        print("Edad:", self.edad)

Presentador("Boby Jackson", "Shinichi Kudo", 17).start()
Presentador("Carlos Guzman", "Kogoro Mouri", 38).start()
```

Thread - Python

```python
import threading, time

class Presentador(threading.Thread):
    def __init__(self, nombre_thread: str, personaje: str, edad: int):
        super().__init__(name=nombre_thread)
        self.personaje = personaje
        self.edad = edad

    def mostrar_edad(self):
        time.sleep(1)
        print("Edad:", self.edad)

    def run(self):
        print("Thread:", threading.current_thread().name)
        threading.Thread(target=self.mostrar_edad).start()
        print("Personaje:", self.personaje)

Presentador("Boby Jackson", "Shinichi Kudo", 17).start()
Presentador("Carlos Guzman", "Kogoro Mouri", 38).start()
```

Poniendo a prueba lo que hemos aprendido 👀

Respecto a los procesos y *threads* ¿Cuál de las siguientes afirmaciones es **correcta**?

a. Los *threads* siempre serás más eficiente que los procesos porque trabajan en paralelo.

b. Los procesos no requieren de ningún intermediario para acceder y modificar archivos del disco duro, mientras que los *thread* si.

c. El fallo de un *thread* es igual al fallo de un proceso, afecta únicamente al que falló.

d. Tanto los procesos como los *threads* necesitan establecer reglas de comunicación para poder compartir la información entre ellos.

e. Los *threads* son más livianos de crear porque comparten el espacio de memoria del proceso que los creó.

Poniendo a prueba lo que hemos aprendido 👀

Respecto a los procesos y *threads* ¿Cuál de las siguientes afirmaciones es **correcta**?

a. Los *threads* siempre serás más eficiente que los procesos porque trabajan en paralelo.

b. Los procesos no requieren de ningún intermediario para acceder y modificar archivos del disco duro, mientras que los *thread* si.

c. El fallo de un *thread* es igual al fallo de un proceso, afecta únicamente al que falló.

d. Tanto los procesos como los *threads* necesitan establecer reglas de comunicación para poder compartir la información entre ellos.

e. Los *threads* son más livianos de crear porque comparten el espacio de memoria del proceso que los creó.

Dato freak

Acabamos de ver el algoritmo del panadero de Lamport

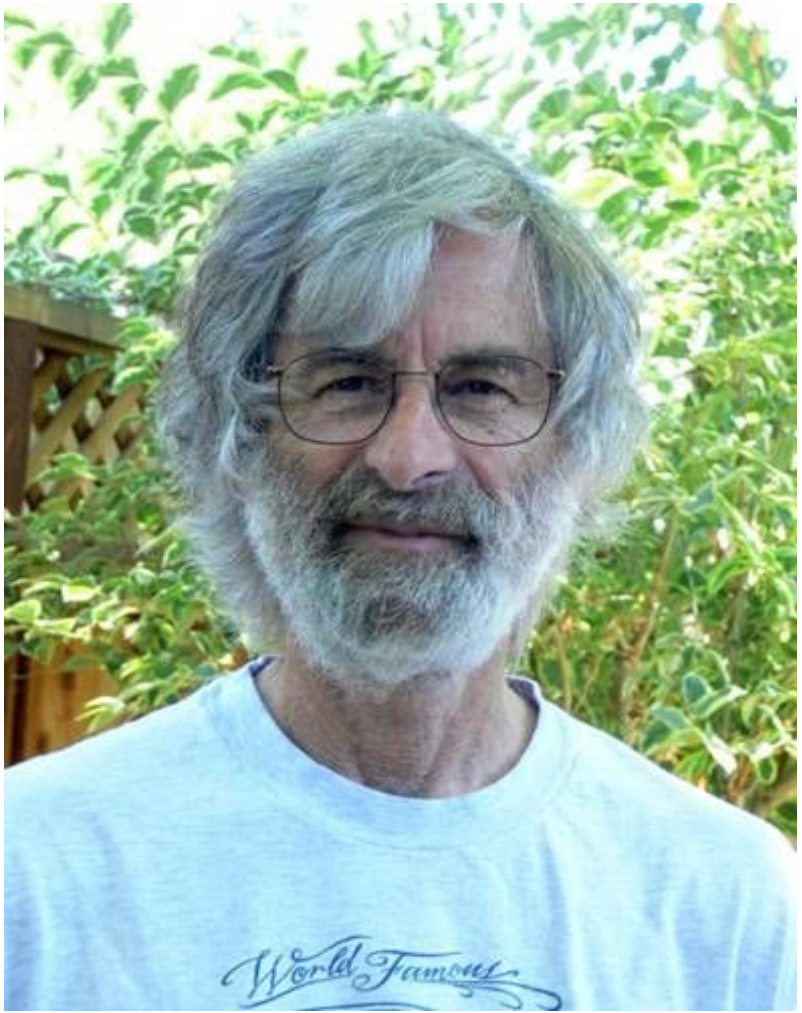
Pero ¿quién es Lamport?

Dato freak

Acabamos de ver el algoritmo del panadero de Lamport

Pero ¿quién es Lamport? **Leslie Lamport**

- Importante figura en el mundo de Sistemas Distribuidos.
- Creó varios algoritmos o conceptos que fueron formalizados en estos sistemas distribuidos.
- Desarrollador inicial del sistema de formateo de textos LaTeX.



Dato freak

Acabamos de ver el algoritmo del panadero de Lamport

Pero ¿quién es Lamport? **Leslie Lamport**

- Importante figura en el mundo de Sistemas Distribuidos.
- Creó varios algoritmos o conceptos que fueron formalizados en estos sistemas distribuidos.
- Desarrollador inicial del sistema de formateo de textos LaTeX.

Próximos eventos

Próxima clase

- Introducción a los Sistemas Distribuidos
  - Formalizaremos la definición y sus diferentes objetivos.
  - Entenderemos los distintos tipos de sistemas que existen.

Evaluación

- Control 1 se publicará el martes siguiente y evalúa hasta la clase siguiente.

IIC2523
Sistemas Distribuidos

Hernán F. Valdivieso López
(2025 - 2 / Clase 01)

Créditos (animes utilizados)

Detective Conan

Spy x Family

Resumen - Comparación entre Proceso y *Thread*

Característica	Procesos	*Thread*
Memoria	Separada	Compartida
Comunicación	Compleja	Aprovecha la memoria compartida
Creación	Costosa	Liviana
Fallos	Aislados	Puede afectar a otros *threads* e incluso a todo el proceso
Librerías Python	`multiprocessing`, `subprocess`	`threading`