Programación Orientada a Eventos

75.42 - Taller de Programación I

Contenidos

- Algunos paradigmas de programación
- Concepto de POE
- Event
- Event Queue
- Event Loop
- Handlers

Contenidos

- **Algunos paradigmas de programación**
- Concepto de POE
- Event
- Event Queue
- Event Loop
- Handlers

Algunos paradigmas de programación

- Programación Secuencial o Imperativa: Se basa en definir la secuencia de pasos que el programa debe seguir
- Programación Orientada a Objetos: Se abstraen las entidades del problema, su comportamiento, su estado y sus interrelaciones
- Programación Funcional: Las funciones se utilizan como predicados matemáticos
- Programación Orientada a Eventos: se define cómo reaccionar a distintos sucesos

Algunos paradigmas de programación

- Programación Secuencial o Imperativa: Se basa en definir la secuencia de pasos que el programa debe seguir
- Programación Orientada a Objetos: Se abstraen las entidades del problema, su comportamiento, su estado y sus interrelaciones
- Programación Funcional: Las funciones se utilizan como predicados matemáticos
- Programación Orientada a Eventos: se define cómo reaccionar a distintos sucesos

Algunos paradigmas de programación

- Programación Secuencial o Imperativa: Se basa en definir la secuencia de pasos que el programa debe seguir
- Programación Orientada a Objetos: Se abstraen las entidades del problema, su comportamiento, su estado y sus interrelaciones
- Programación Funcional: Las funciones se utilizan como predicados matemáticos
- **Programación Orientada a Eventos: se define cómo reaccionar a distintos sucesos**

Algunos paradigmas de programación

- Programación Secuencial o Imperativa: Se basa en definir la secuencia de pasos que el programa debe seguir
- Programación Orientada a Objetos: Se abstraen las entidades del problema, su comportamiento, su estado y sus interrelaciones
- Programación Funcional: Las funciones se utilizan como predicados matemáticos
- **Programación Orientada a Eventos**: se define cómo reaccionar a distintos sucesos

Contenidos

- **Algunos paradigmas de programación**
- Concepto de POE
- Event
- Event Queue
- Event Loop
- Handlers

POE: ¿En qué consiste?

1. Toda **acción** que ejecute el sistema será en respuesta a los **sucesos** que acontezcan
2. A esos **sucesos** los llamamos **eventos**
3. Las **acciones a ejecutar**, lo que debemos programar, son los **manejadores**
4. A las fuentes de eventos las vamos a nombrar **actores**

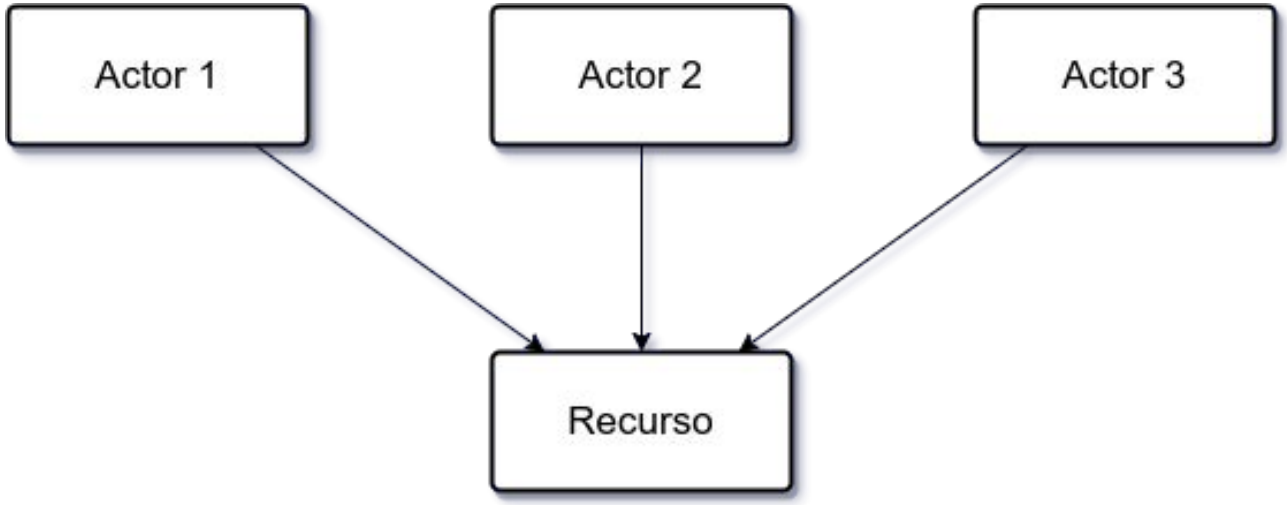
Contenidos

- **Algunos paradigmas de programación**
- Concepto de POE
- Event
- Event Queue
- Event Loop
- Handlers

¿De dónde vienen los eventos?

- Acciones del usuario
  - Click en un botón de una interfaz
  - Movimiento del puntero del mouse por sobre un Widget
  - Key-down/Key-up
  - Gestos!!
- Basados en tiempo
  - Pasaron 5 segundos desde cierto evento (timeout)
  - Se alcanza el horario 18hs del martes
- Generados por otro evento
  - Nuestro código puede disparar eventos también
- Definidos por el usuario
- Eventos del entorno
  - hotplug

Mucha Concurrencia



Como tenemos muchos actores accediendo a recursos compartidos vamos a tener **muchos** problemas de concurrencia.

¿Cómo atender a todos?

- Antes en el curso vimos algunas técnicas para evitar problemas de concurrencia, y destacamos dos:
  - Mutex para encerrar critical sections (CS)
  - Colas bloqueantes
- ¿Qué tienen en común? La **serialización** (no marshaling)
  - Usando mutex podemos ver una serialización de CS (se hace primero la CS que obtuvo el mutex primero, después otra, etc)
  - Usando colas lo que se serializan son objetos, por la naturaleza FIFO de la estructura

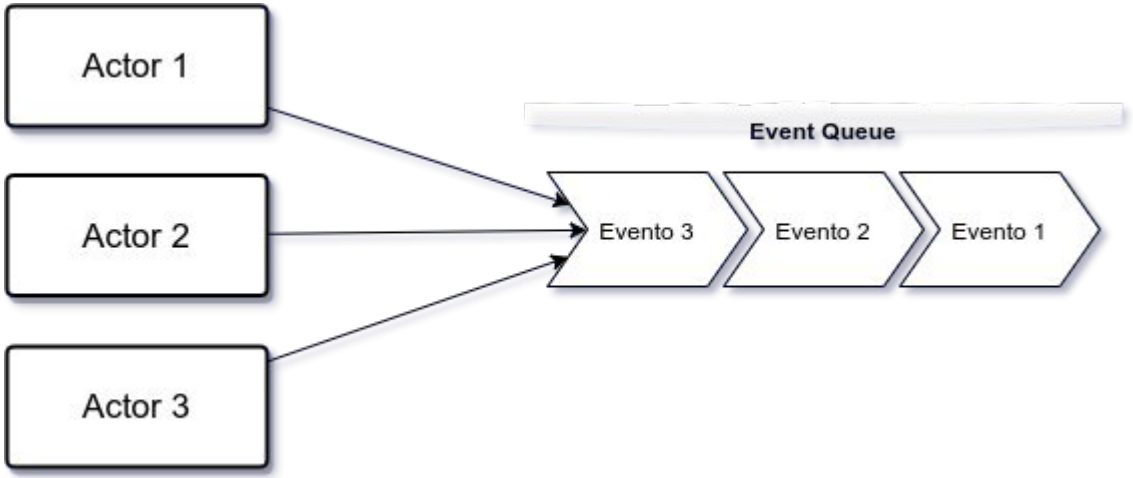
¿Entonces cuál usamos?

- Podemos usar mutex y programar de manera imperativa u orientada a objetos, y cuando vamos a acceder a un recurso protegerlo
- O bien usar una cola
  - Modelizamos los eventos como estructuras de datos
  - Los eventos son generados por los distintos actores
  - Y luego son agregados a una cola para que alguien los atienda
- Hoy estamos vendiendo la segunda, así que sigamos ese camino

Contenidos

- **Algunos paradigmas de programación**
- Concepto de POE
- Event
- Event Queue
- Event Loop
- Handlers

Event Queue



- A esa cola la llamaremos Cola de Eventos
- Proteger esa cola nos brinda mucha seguridad respecto a la concurrencia (pero no nos salva)

Contenidos

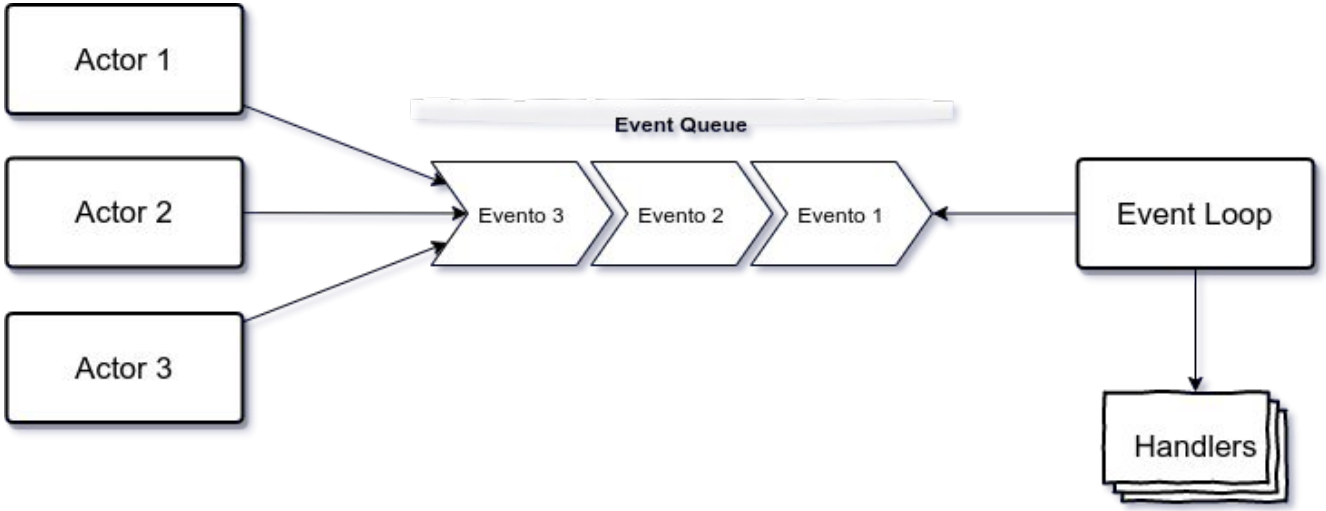
- **Algunos paradigmas de programación**
- Concepto de POE
- Event
- Event Queue
- **Event Loop**
- Handlers

¿Quién lee los eventos de la Cola?

```
1 while se_debe_continuar:
2   evento = cola.obtener_evento()
3
4   if evento == salir:
5     se_debe_continuar = false
6
7   elif existe_manejador_para(evento):
8     ejecutar_manejador_para(evento)
```

- A ese bucle se lo llama **Event Loop** y es un patrón muy usado en toda GUI
- Es muy similar también al **Game Loop** que se usa en los juegos para simular paso del tiempo

Actualizando nuestro diagrama...



En los casos más simples, los handlers se ejecutan en serie y por ende no tenemos problemas de concurrencia. **Pero solamente en los casos más simples**

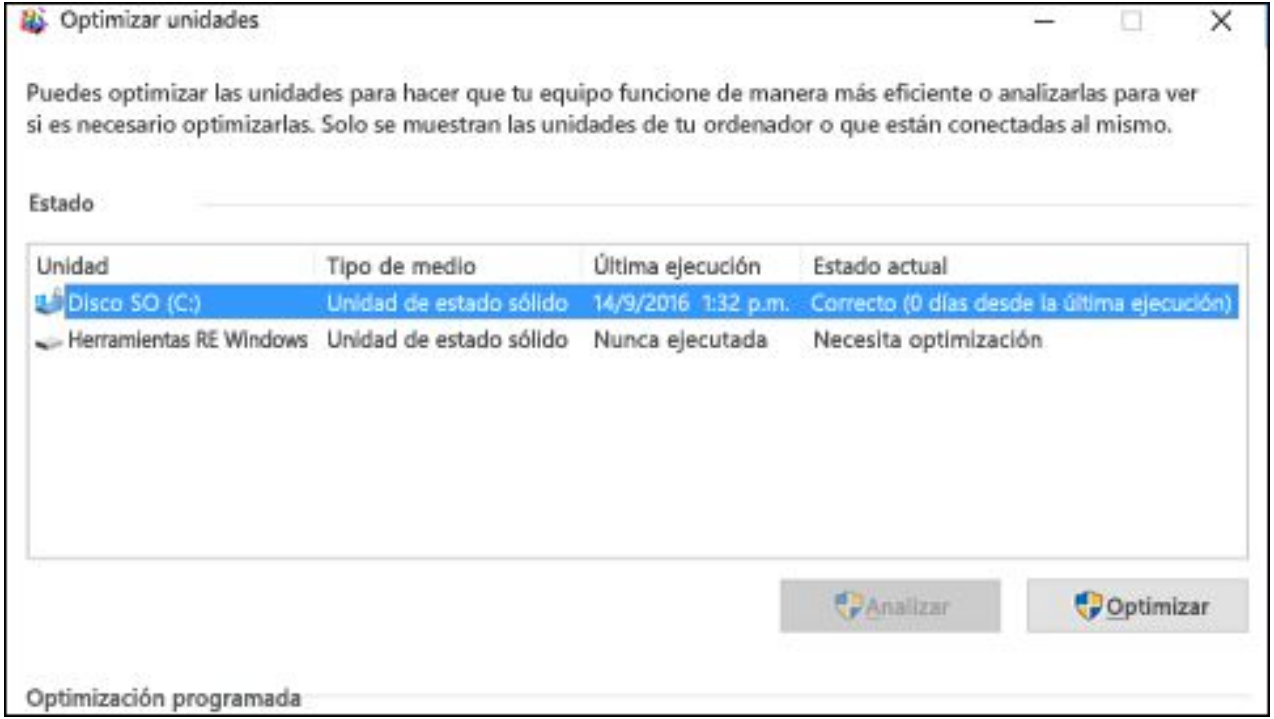
Contenidos

- **Algunos paradigmas de programación**
- Concepto de POE
- Event
- **Event Queue**
- **Event Loop**
- **Handlers**

Handlers / Listeners

- Los **handlers** son secciones de código que saben cómo responder a la aparición de un **evento**
- Vamos a unir (**bind**) los **handlers** a los **eventos**
- Pueden requerir cierta información sobre el **evento** (en qué coordenadas de la pantalla se hizo click, cuánto duró la pulsación de un botón, o alguna información propia de la aplicación)
- En muchos casos (simples), la ejecución secuencial nos permite programar de manera thread-safe, pero... **¿qué pasa si la información que requiere un handler está en el contexto de otro handler?**
- **¿Y si un handler tarda mucho?**

Desfragmentar el disco



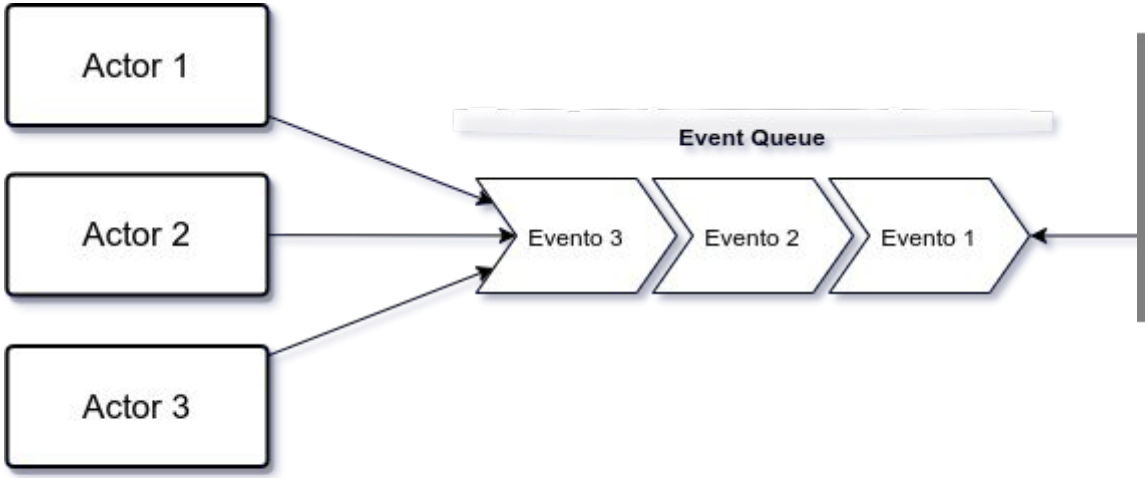
- Esta es la ventana que nos muestra Windows10 cuando queremos desfragmentar el disco
- El botón Optimizar va a generar una tarea que tarda mucho tiempo

Handler largo

```
1 def desfragmentarDisco():
2   // cursar SSOO y rellenar esta funcion
3
4 botonOptimizar.onClick = lambda ctx:
5   desfragmentarDisco()
6
```

- El botón Optimizar va a generar una tarea que tarda mucho tiempo, y la opción más sencilla para implementar el handler es simplemente desfragmentar el disco allí mismo
- Pero no queremos que se trabe la aplicación (responsiveness)

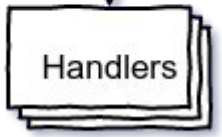
Handler largo




```
while se_debe_continuar:
  evento = cola.obtener_evento()

  if evento == salir:
    se_debe_continuar = false

  elif existe_manejador_para(evento):
    ejecutar_manejador_para(evento)
```



¡Handlers Rápidos!

```
1 def desfragmentarDisco():
2   // cursar SSOO y rellenar esta funcion
3   emit("desfragmentacion_completa")
4
5 botonOptimizar.onClick = lambda ctx:
6   botonOptimizar.disable()
7   spawn(desfragmentarDisco)
```

- Para hacer un handler más rápido, una opción es lanzar un thread que haga en paralelo la desfragmentación, y emita un evento cuando terminó
- Deshabilitar el botón que lanza este tipo de tareas es muy útil para cuidar a los usuarios más ansiosos

Hay más alternativas

- Una mejor idea sería partir el handler en etapas más cortas y usar threadpool y schedulers adecuados
- Se escapa de este curso, pero lanzar threads ante operaciones del usuario no es la idea más segura. Sí es parte del curso notar la importancia del multithreading en POE (sobre todo cuando hay GUI)
- Partir los handlers también nos permite cancelar operaciones de maneras no tan abruptas, y poder tener actualizaciones de estado (barra de progreso)

Problemas de Concurrencia (otra vez)

- Vimos entonces que, como no podemos hacer handlers muy largos, necesitamos otros hilos además del correspondiente al event loop, y por ende vamos a tener problemas de concurrencia en los recursos que compartan los distintos hilos

- El loop de eventos suele correr en el hilo principal en los frameworks para hacer GUIs, y eso a veces interfiere con nuestro modelo de concurrencia.

Conclusiones

- La POE nos evita muchos problemas de concurrencia, pero no todos. Nos da **muchas ventajas, pero hay que ser cuidadosos**
- Es un paradigma muy usado en aplicaciones con **GUI**, ya que es el usuario quien decide el flujo del programa (o al menos en el controller ¿MVC?)
- Cuando hay GUI, es importante programar **handlers cortos** y delegar operaciones largas en otros threads (si no hay GUI puede no ser tan importante)