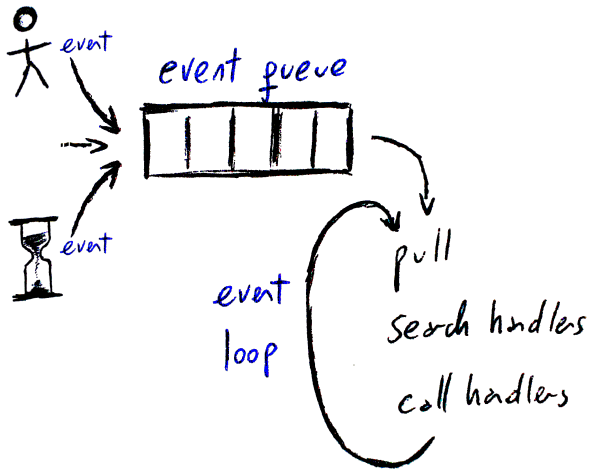
Programación orientada a eventos

Di Paola Martín
martinp.dipaola <at> gmail.com

Facultad de Ingeniería
Universidad de Buenos Aires

Programación orientada a eventos



- Múltiple fuentes de eventos: el input del usuario (mouse, teclado) y el paso del tiempo son solo algunos ejemplos.
- El mismo programa puede generar eventos para que sean procesados luego.
- Si múltiples events son generados, estos se guardan en una cola de eventos (dependiendo de la librería los eventos son sacados en el orden en que entraron FIFO o no).

Prohibido usar handlers lentos

```
1  void save_button_handler() {
2     FILE *f = fopen("data.txt", "wt");
3
4     /* ... */
5     fwrite(data, sizeof(char), data_sz, f);
6     /* ... */
7
8     fclose(f);
9  }
```


- Los handlers no deben bloquearse ni realizar tareas que lleven mucho tiempo para no bloquear a todo el programa.
- Se puede tomar un handler y particionarlo en subpartes para ejecutarlo iterativa e incrementalmente sin bloquear el programa. No veremos esta alternativa en la materia.

Handlers lentos: multithreading

```
1  void save_background() {
2     FILE *f = fopen("data.txt", "wt");
3
4     /* ... */
5     fwrite(data, sizeof(char), data_sz, f);
6     /* ... */
7
8     fclose(f);
9  }
10
11 void save_button_handler() {
12    std::thread t1 {save_background};
13 }
```

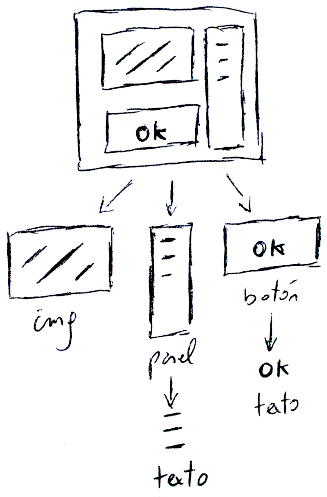
- Si es necesario, el handler puede lanzar un hilo para hacer la tarea en background.
- Esto plantea una serie de preguntas: quien hace el `join` sobre el hilo? Podría haber otro hilo haciendo joins (ineficiente) o prodríamos usar más eventos.

Handlers lentos: multithreading - join

```
1  void save_background() {
2     FILE *f = fopen("data.txt", "wt");
3
4     /* ... */
5     fwrite(data, sizeof(char), data_sz, f);
6     /* ... */
7
8     fclose(f);
9     emit_event("joinme", thread);
10 }
11
12 void save_button_handler() {
13    std::thread t1 {save_background};
14 }
15
16 void joinme_handler(thread) {
17     thread.join();
18 }
```

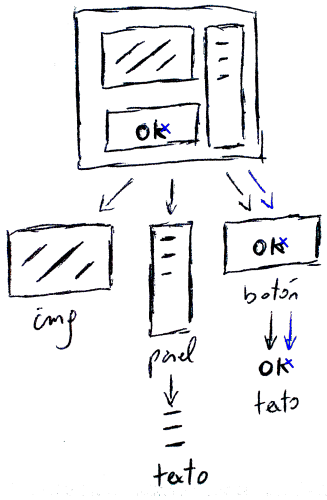

- Al finalizar `save_background` emite un evento y un handler registrado hara el `join`.
- El `joinme_handler` es llamado al ocurrir el evento y hace el `join` sin bloquearse.

Búsqueda de los handlers (tree version)



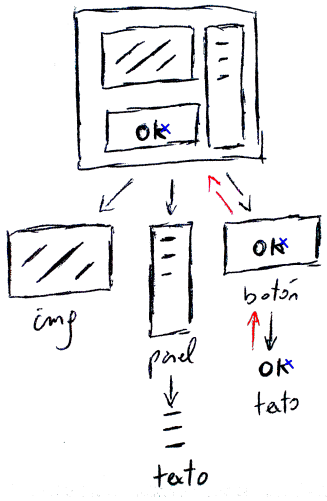
- Dado un evento, cómo se busca a su handler? Depende de la librería usada.
- En las librerías gráficas y en los web browsers, las ventanas y páginas web son representadas por estructuras jerárquicas (árboles).

Búsqueda de los handlers (tree version): capture phase



- En estas estructuras tipo árbol un evento puede ser capturado por varios objetos: un click en un texto puede verse como un click en el botón que tiene el texto o como un click en el todo que tiene al botón.
- En los web browsers la búsqueda del handler se hace del todo al específico en lo que se conoce como **`capture phase`**.

Búsqueda de los handlers (tree version): bubble phase



- En una etapa posterior, el web browser hace una segunda pasada en el orden inverso de lo específico a lo general, de ahí el nombre **`bubble phase`**.
- Otras librerías gráficas tienen búsquedas similares, otras sólo tienen una fase de búsqueda. Otras incluso simplemente levantan un bit en un array.
- En general, un handler puede notificar que el evento debe cancelarse: el resto de los handlers no es llamado. Pero como todo, dependerá de la librería usada.

Búsqueda de los handlers (bit version): select

```
1  fd_set rfds;
2  struct timeval tv;
3
4  while (...) {
5     FD_ZERO(&rfds);
6     FD_SET(0, &rfds); /* 0 es la entrada estandar */
7
8     /* timeout de 5 segundos */
9     tv.tv_sec = 5;
10    tv.tv_usec = 0;
11
12    if (select(1, &rfds, NULL, NULL, &tv) == -1)
13        perror("select()");
14    else if (FD_ISSET(0, &rfds))
15        read(0, ...,); /* no deberia bloquearse */
16    else
17        /* time out */
18 }
```


- En este caso el sistema operativo nos permite ver a los file descriptors como fuentes de eventos: lectura disponible, escritura disponible, excepcion disponible.
- A diferencia de una librería gráfica, el proceso de búsqueda se hace con un array de bits manipulado con las macros **`FD_ZERO`**, **`FD_SET`** y **`FD_ISSET`**
- **`select`** se bloquea hasta que uno o varios eventos llegan sobre los file descriptors marcados: no hay tal FIFO.
- En la cátedra no se usara **`select`** ni similares pero se deja como ejemplo.