Fundamentos de Software de Comunicaciones

Máquinas de estado

Diseño de protocolos

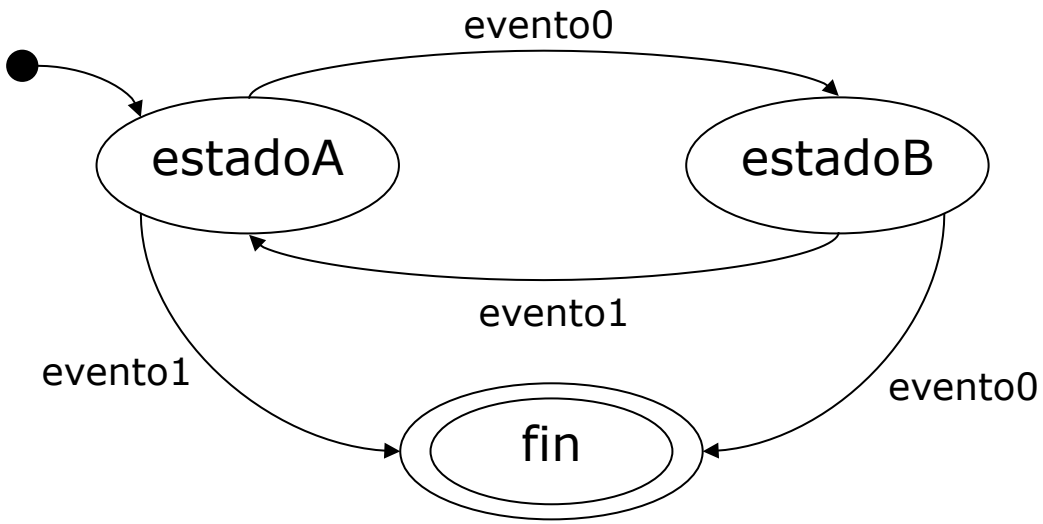
- Protocolo de comunicación
  - Conjunto de reglas que define cómo deben dialogar dos o más entidades
  - Entidades: procesos

- Diseño de protocolos: perspectivas profesionales
  - Implementar una norma internacional (5G/6G…) en
    - Redes de telefonía
    - Routers
    - Máquinas virtuales en la nube
    - Sensores

- Elementos de un protocolo
  - Servicio que da: ¿Para qué sirve?
  - Contexto en el que se utiliza
  - Vocabulario de mensajes
  - Formato de cada mensaje
  - Reglas de cómo proceder:
    - Secuencia de mensajes que se intercambian
    - Quién empieza a hablar
    - ¿Qué pasa si se excede un tiempo sin que aparezca un mensaje esperado?
    - ¿Cómo reacciona el protocolo cuando ocurre un evento?

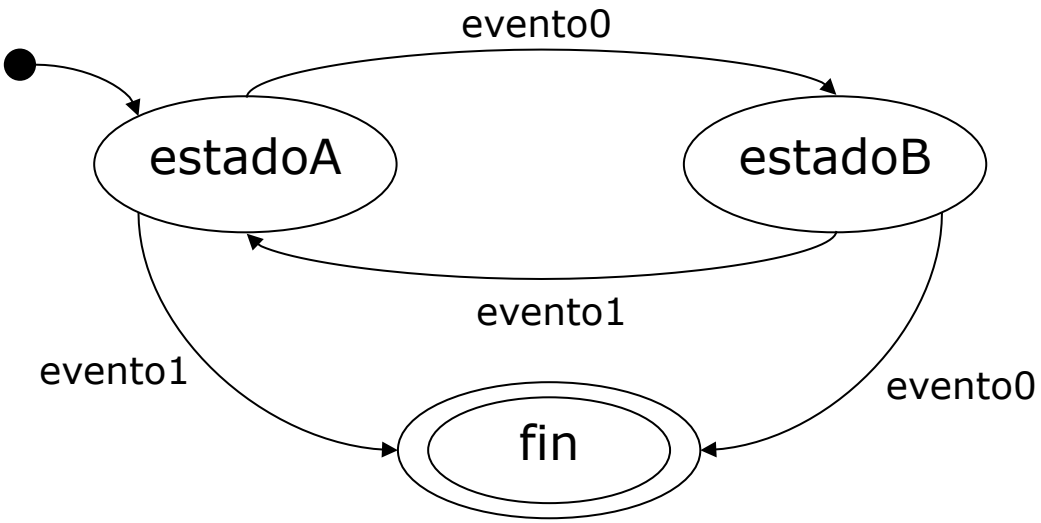
- Reglas → Máquinas de estado

Protocolos y máquinas de estados



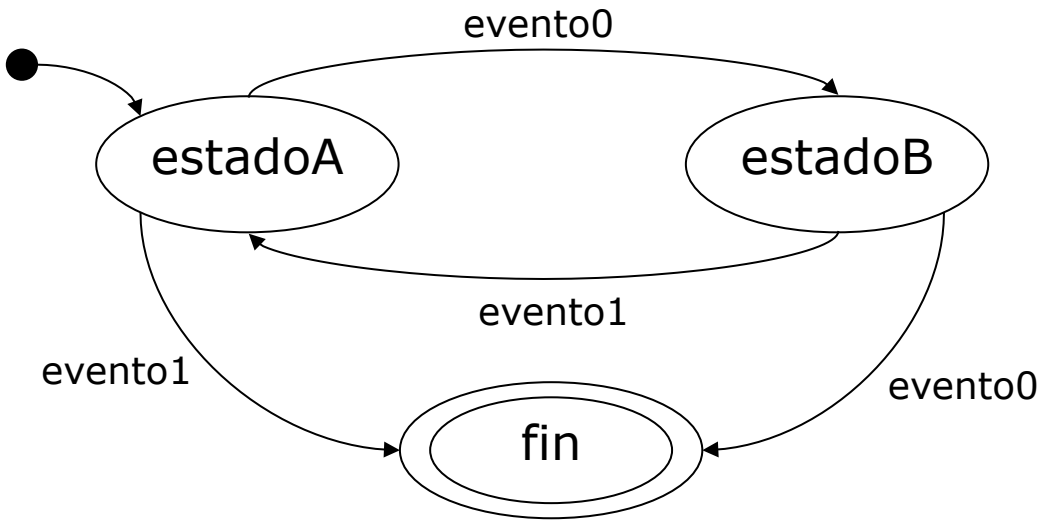
- Elementos:
  - **Estados**: representan un estado estable del programa (las variables que definen su comportamiento tienen un valor fijo)
    - Distinguidos: **inicial** y **final**
  - **Eventos**: ocurrencia de algo que permite transitar hasta otro estado

Máquinas de estados: implentación



- En programación imperativa existen dos aproximaciones:
  1. Lineal: a través de sentencias anidadas
     - switch(estado)/case { switch(evento)/case}
     - rápido, poco modular
  2. Tabla de punteros a función manejadora (no
     - protocolo[estado][evento](argumentos)
     - costoso en memoria, pero modular

Máquinas de estados: implementación



- Se puede usar cualquier valor, siempre que sean distinto
- La identificación de eventos y estados es independiente

```
#define estadoA 0
#define estadoB 1

#define eventoA 0
#define eventoB 1
```

Máquinas de estado: implementación

```
27      int fin=0;
28      int estado = estadoA; /*estado inicial*/
29      while(!fin){
30          /*bloquea hasta que llegue un evento (podria utilizar un read/select/pause, etc.)*/
31          int evento = espera_evento();
32
33          switch(estado){
34              case estadoA:
35                  switch(evento){
36                      case evento0:
37                          printf("conmutaA->B\n"); estado = estadoB;
38                          break;
39                      case evento1:
40                          printf("fin\n");
41                          fin=1;
42                          break;
43                  }
44                  break;
45              case estadoB:
46                  switch(evento){
47                      case evento0:
48                          printf("fin\n");
49                          fin=1;
50                          break;
51                      case evento1:
52                          printf("conmutaB->A\n");
53                  }
54                  break;
55              default:
56                  printf("Estado no reconocido\n"); fin=1; break;
57          }
58      }
```

Máquinas de estado: implementación

- Detalles de implementación usando doble anidamiento
  1. Hay una variable que contiene el estado en el que se encuentra la máquina
  2. Hay una función **espera_evento()**
     - Encargada de poner al proceso en espera
       - La función **pause()**
       - La función **read()**
       - La función **select()**
     - Se desbloquea cuando se produce un evento
     - Devuelve el evento recibido
  3. Doble **switch/case**
     1. Primer nivel: estado en el que se encuentra la máquina
     2. Segundo nivel: evento que se recibe
        1. Puede ocurrir que no todos los eventos se puedan recibir en todos los estados

- Hay flexibilidad
  - espera_eventos puede recibir argumentos
  - La máquina puede estar en una función
  - …

Fallos típicos

- No existe el bucle while
  - La máquina sólo procesa un estado y sale

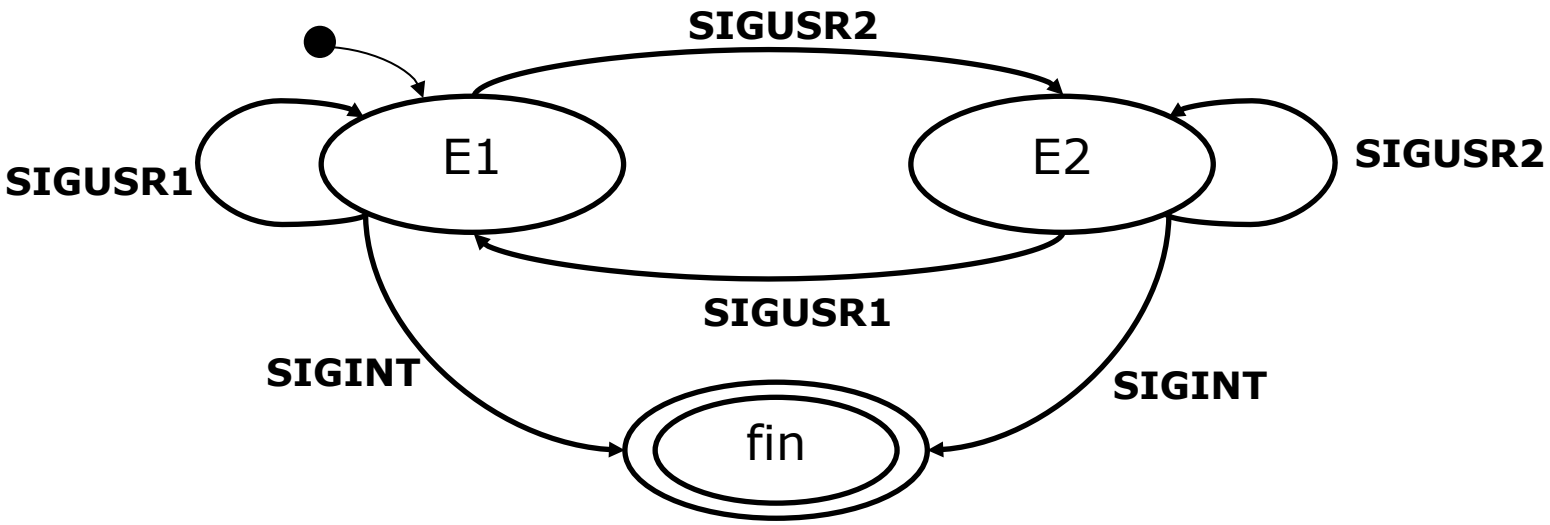
- Se esperan los eventos fuera del bucle

```
int estado = estadoA; /*estado inicial*/
int evento = espera_evento();
while(!fin){
    switch(estado){
        /*aqui vienen los case y dentro de
        cada uno el switch de eventos*/
    }
}
```

- Espera de eventos dentro del switch

```
switch(estado){
    int evento = espera_evento();
    case estadoA:
        switch(evento){
            case evento0:
                printf("conmutaA->B\n"); estado = estadoB;
                break;
            case evento1:
```

Ejemplo: señales como eventos



Ejemplo 1: sólo señales

- La máquina transita entre **E1** y **E2** cuando llegan las señales **SIGUSR1** y **SIGUSR2**
- En caso de llegar **SIGINT**, la máquina finaliza su ejecución

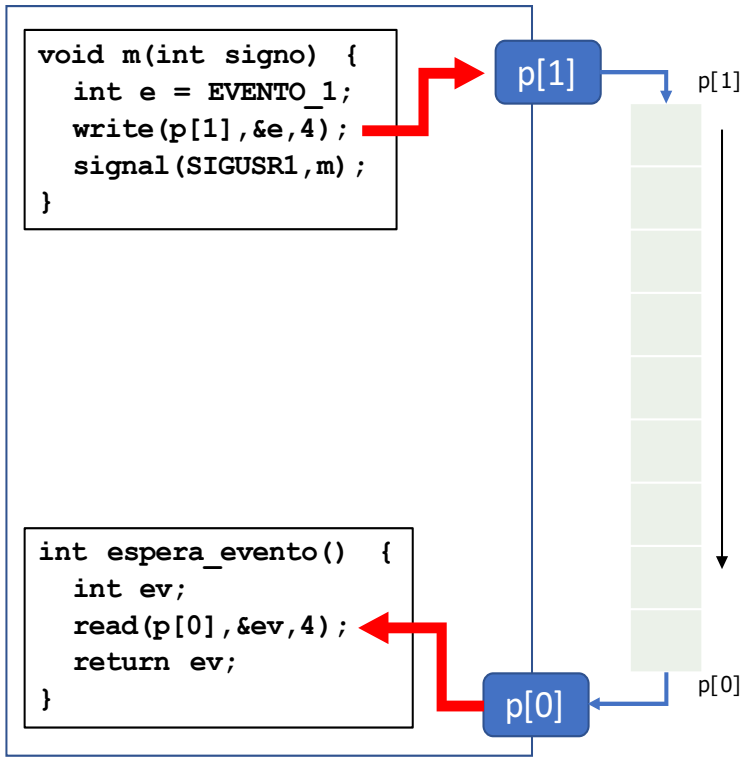
Error de diseño

- ¿Qué ocurre si, mientras se atiende, por ejemplo, la llegada de **SIGUSR1**, pero antes de bloquearse en el **pause()** llega otro **SIGUSR1**?
- Solución: almacenar las señales que llegan en tuberías
- Idea:
  - La máquina de estados crea una tubería cuyos descriptores son una **variable global**

```
24  /**
25   * Variable global
26   */
27  int p[2];
```

```
91      if (pipe(p) < 0) {
92          perror("pipe");
93          exit(-1);
94      }
```

  - Los manejadores **escriben** eventos en la pipe
  - Y **espera_evento()** los saca, en orden usando **read()**



Solución

- Cada vez que llegue una señal, el manejador de la misma escribe en la pipe el evento recibido, que así no se pierde

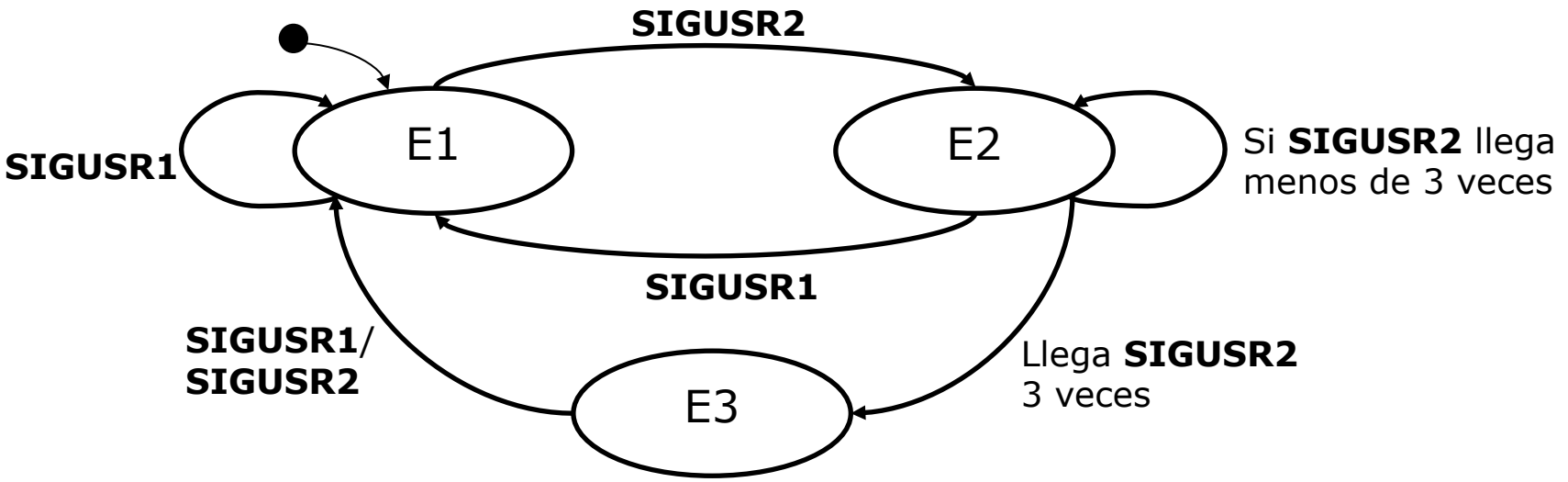
```
29  void m1(int signo) {
30      int e = EVENTO_1;
31      if (write_n(p[1], &e, sizeof(int)) != sizeof(int)) {
32          perror("write_n");
33          exit(-1);
34      }
35      signal(SIGUSR1,m1);
36  }
```

- El uso de **write_n()** protege, además, ante la potencial llegada de otra señal, que se encolaría en la pipe
- Ahora, **espera_evento()** ya no tiene un **pause()**, si no que bloquea el proceso con un **read()**, que bloqueará al proceso mientras que la pipe esté vacía

```
58  int espera_evento() {
59      int evento_recibido;
60      if (read_n(p[0], &evento_recibido, sizeof(int)) != sizeof(int)) {
61          perror("read_n");
62          exit(-1);
63      }
64      return evento_recibido;
65  }
```

- Nótese el uso de **write_n()/read_n()**
  - Aplicación con señales
  - Se conoce de antemano la cantidad exacta de bytes a escribir/leer

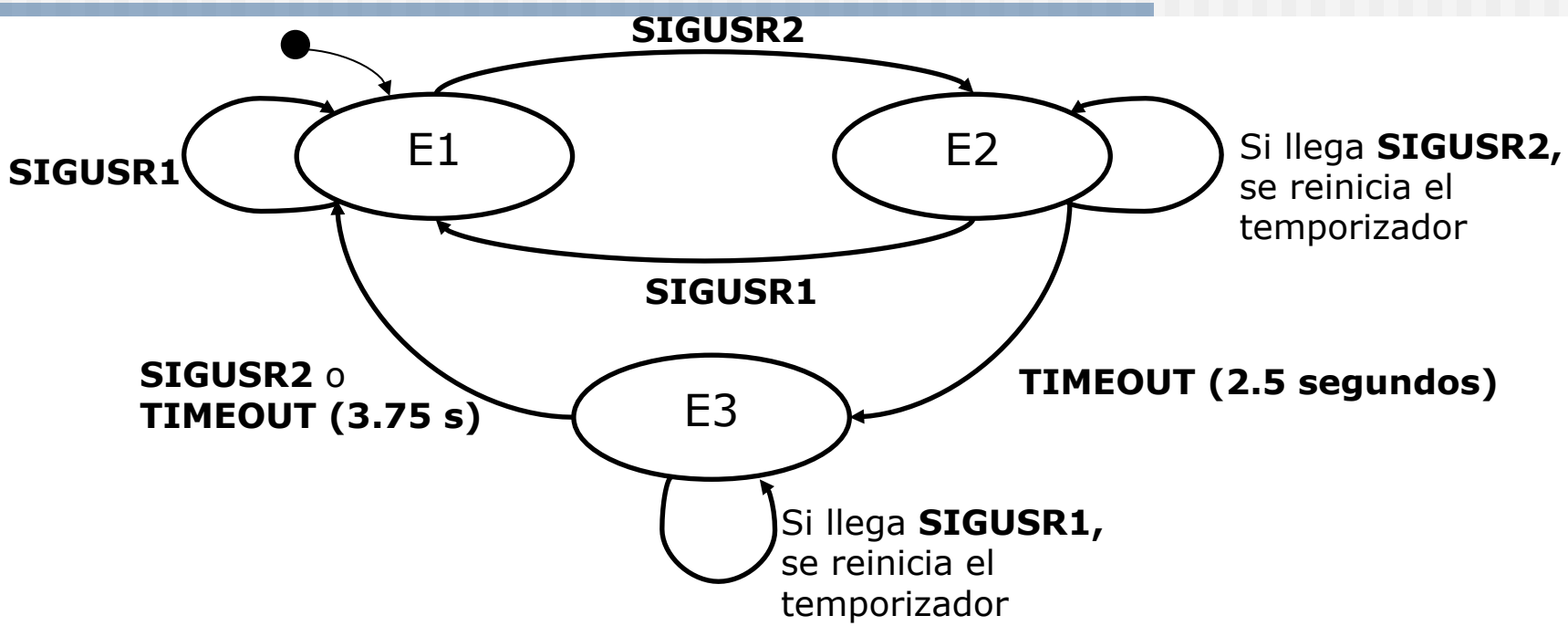
Ejemplo: señales como eventos



Ejemplo 2: señales y condiciones

- La máquina transita entre estados cuando llegan las señales **SIGUSR1** y **SIGUSR2**
- La máquina transita a **E3** cuando, estando en **E2**, llegan tres señales **SIGUSR2** seguidas
- En caso de llegar **SIGINT** en cualquier estado, la máquina finaliza su ejecución (no se representa por simplicidad)
- La condición no llega de **espera_evento()**

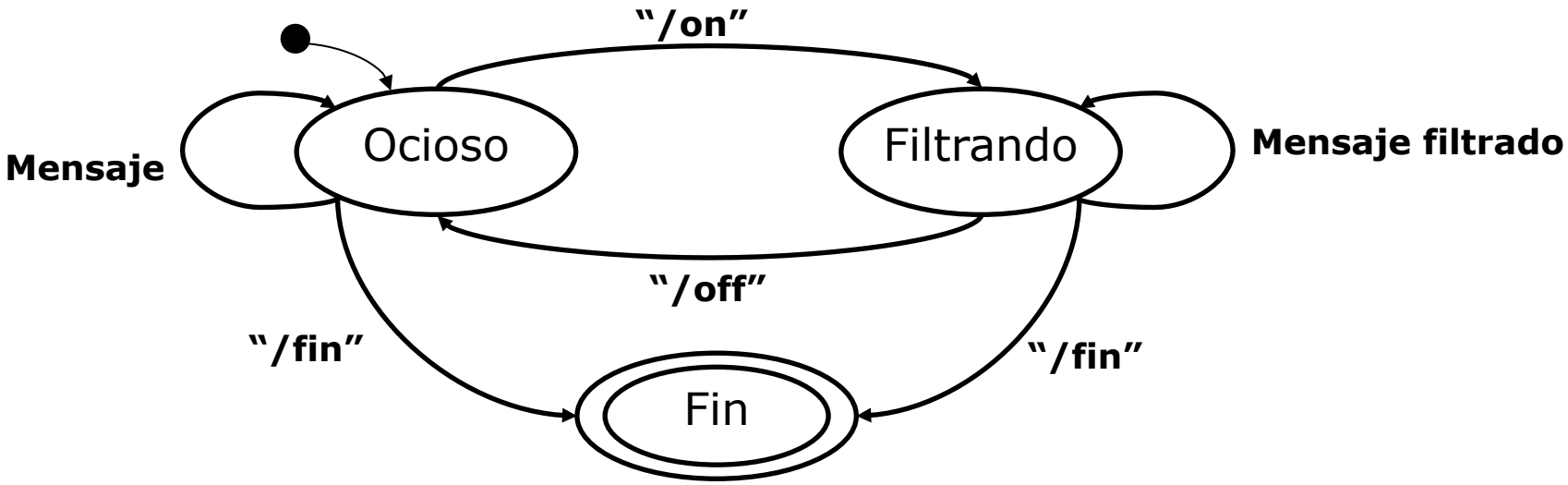
Ejemplo: señales como eventos



Ejemplo 3: señales y temporizadores

- Se mantiene en **E2** siempre que llegue **SIGUSR2** antes de **2.5 s**
- Transita de **E2** a **E3** cuando, pasan **2.5 s**
- Se mantiene en **E3** siempre que llegue **SIGUSR1** antes de **3.75 s**
- Transita a de **E3** a **E1** cuando llega **SIGUSR2** o cuando pasan **3.75 s**
- El temporizador:
  - Se **activa** antes de **cambiar de estado**: **E1** -> **E2**, y **E2** -> **E3**.
  - Se **desactiva** antes de pasar a **E1** cuando llega **SIGUSR2**
  - Se **rearma** estando en **E2** y **E3** cuando llegan **SIGUS2** y **SIGUSR1**, respectivamente

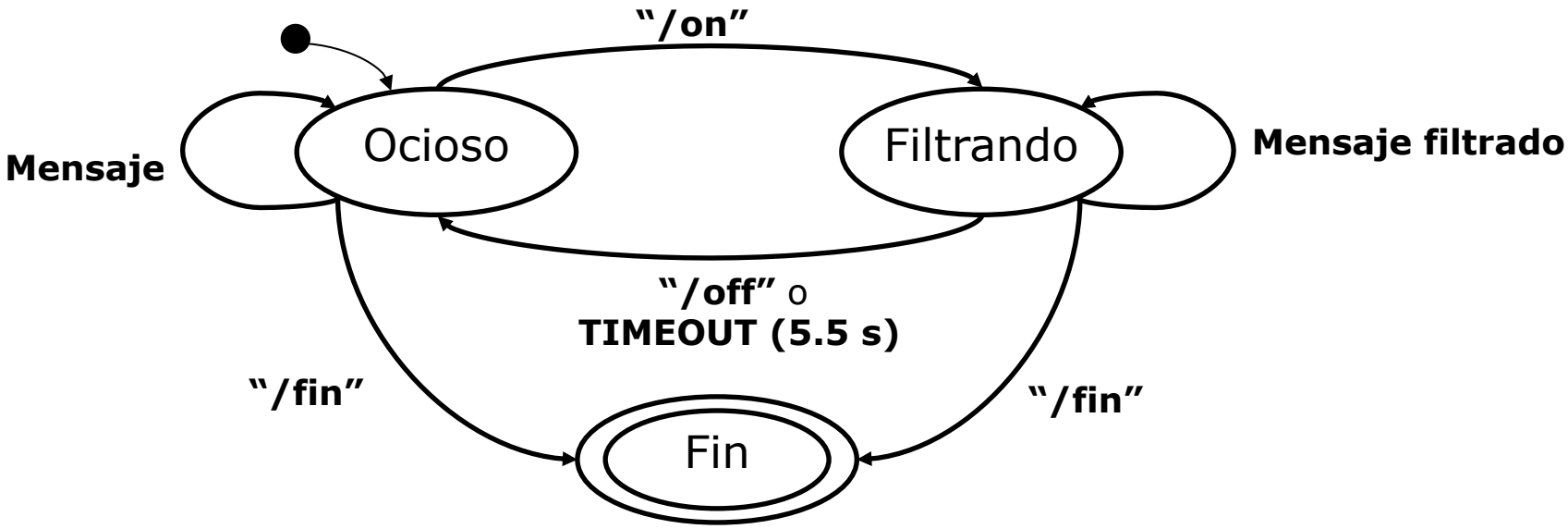
Ejemplo: llegada de mensajes como eventos



Ejemplo 4: eventos disparados por llegada de datos

- Canal de datos: **teclado**
- Eventos: la introducción de una cadena el usuario
- Filtrado: si se encuentra la palabra "**taco**" se sustituya por "****"

Ejemplo: llegada de mensajes como eventos



Ejemplo 5: eventos por llegada de datos + temporizadores

- Canal de datos: **teclado**
- Eventos:
  - Introducción de una cadena el usuario
  - Tiempo máximo
- Filtrado: si se encuentra la palabra "**taco**" se sustituya por "********"
- Se deja de filtrar datos tras **5.5 segundos** de inactividad en ese estado
- En este ejemplo, es posible usar **select()** para medir el tiempo y evitar señales/temporizadores