Entrada/Salida - Drivers

Sistemas Operativos

Departamento de Computación, FCEyN, UBA

14 de Mayo de 2024

Primer Cuatrimestre de 2024

Dispositivo de entrada-salida

Para nosotros, un dispositivo de E/S va a tener, conceptualmente, dos partes:

- El dispositivo fisico.
- Un controlador del dispositivo: interactúa con el SO mediante algún tipo de bus o registro.

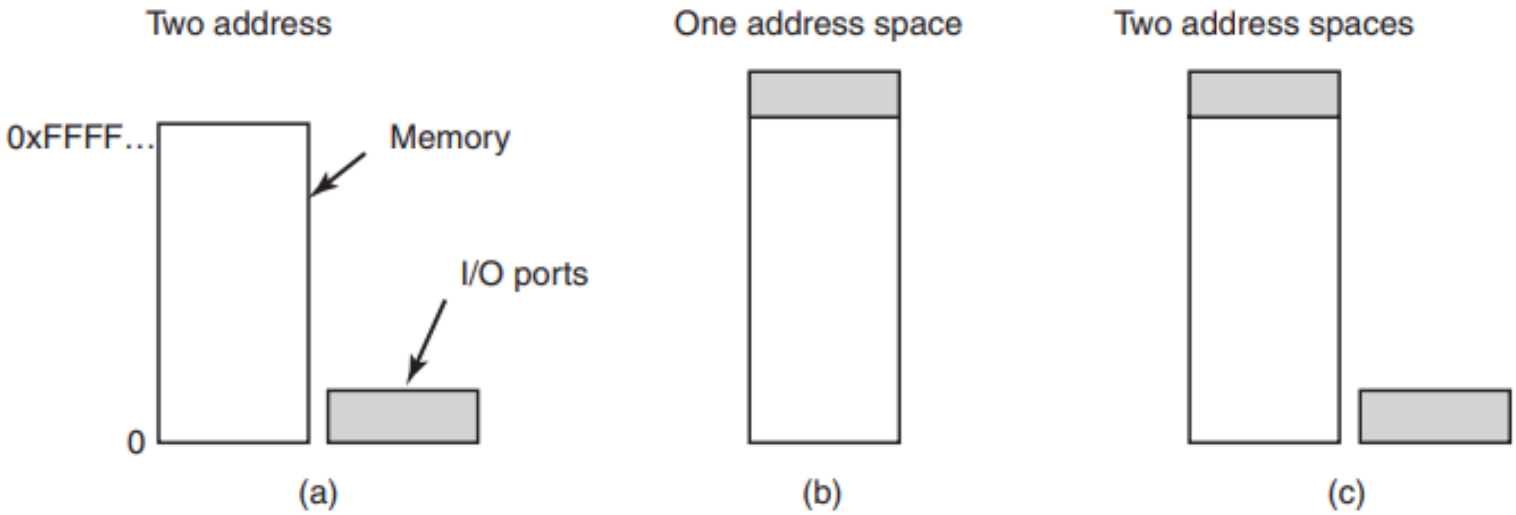
Dispositivo de entrada-salida

Hay dos alternativas para que el SO se comunique con los dispositivos de E/S mediante registros.

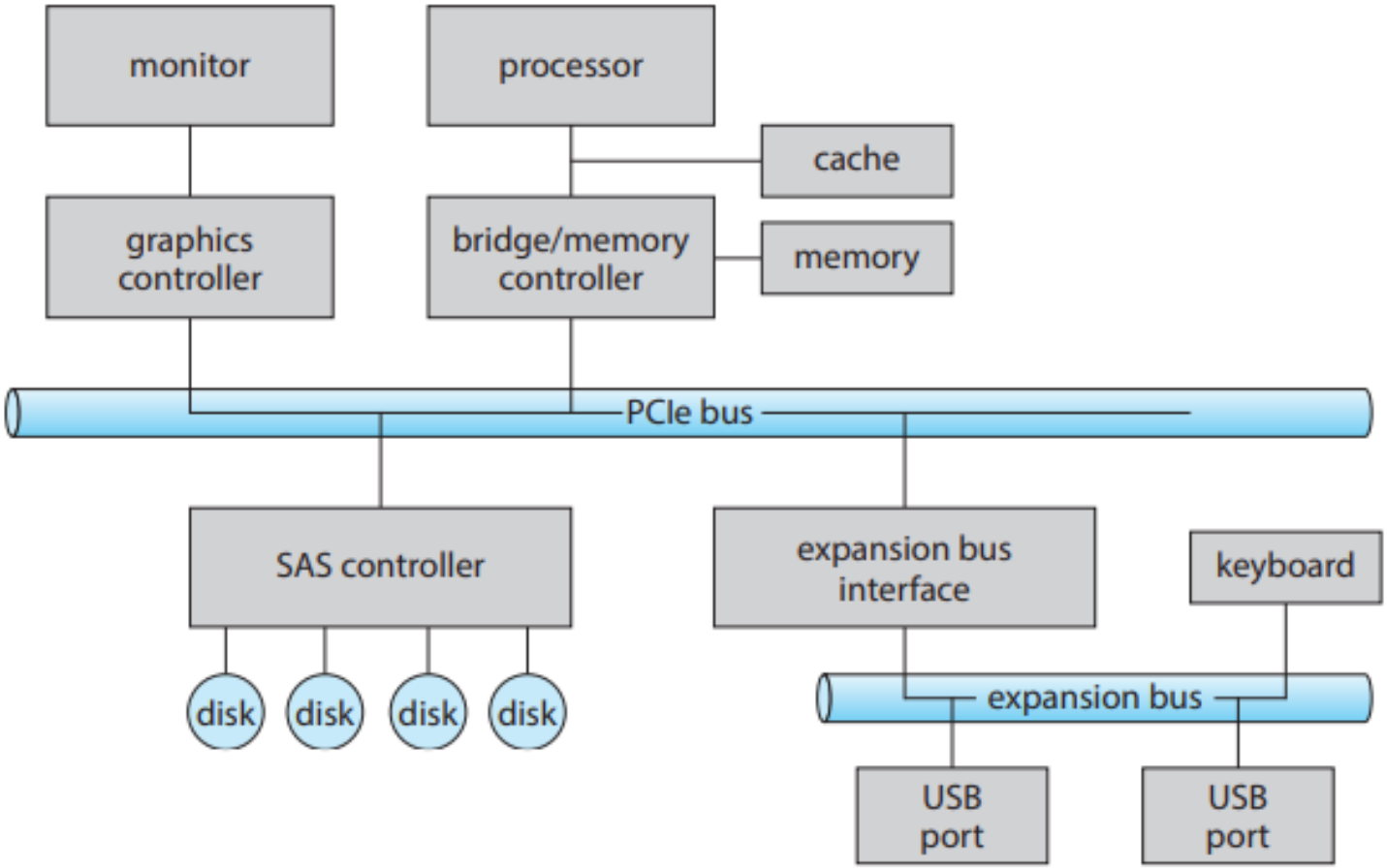
- Cada registro de control se le asigna un puerto de E/S. Estos están protegidos, donde solo el kernel puede acceder a ellos
- A cada registro de control se le asigna una única dirección de memoria que no es usada.

Tambien se utilizan una mezcla de ambas estrategias.

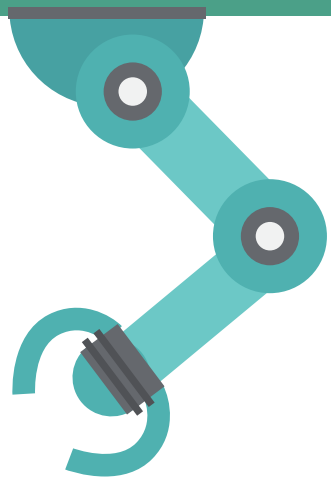
Dispositivo de entrada-salida



Dispositivo de entrada-salida

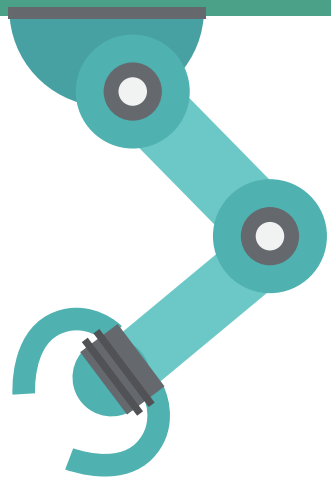


El bot y la caja



Una pequeña empresa de logística acaba de adquirir un **ROBOT** que permite localizar y obtener cajas en su depósito.

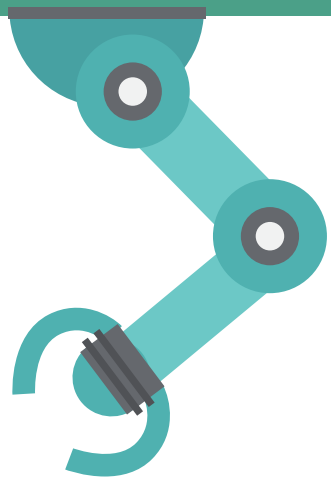
El bot y la caja



Una pequeña empresa de logística acaba de adquirir un **ROBOT** que permite localizar y obtener cajas en su depósito.

Cuando se le ingresa un código en el registro de 32 bits `LOC_TARGET` y la constante `START` en el registro `LOC_CTRL`, el robot comienza la operación de búsqueda, escribiendo el valor `BUSY` en el registro `LOC_STATUS`.

El bot y la caja

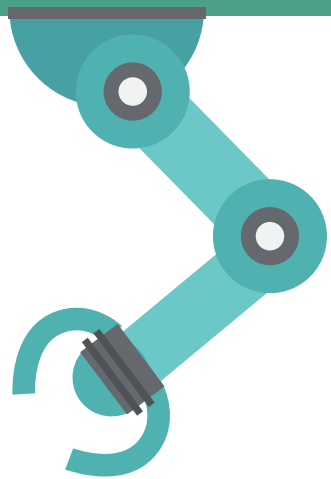


Una pequeña empresa de logística acaba de adquirir un **ROBOT** que permite localizar y obtener cajas en su depósito.

Cuando se le ingresa un código en el registro de 32 bits `LOC_TARGET` y la constante `START` en el registro `LOC_CTRL`, el robot comienza la operación de búsqueda, escribiendo el valor `BUSY` en el registro `LOC_STATUS`.

Al encontrar la caja, la deposita en la bandeja de salida, escribe el valor `FOUNDED` en el registro `LOC_CTRL` y el valor `READY` en el registro `LOC_STATUS`.

El bot y la caja



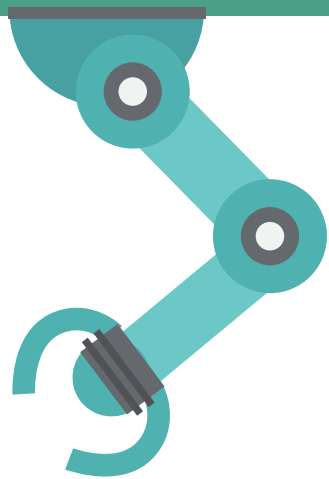
Una pequeña empresa de logística acaba de adquirir un **ROBOT** que permite localizar y obtener cajas en su depósito.

Cuando se le ingresa un código en el registro de 32 bits `LOC_TARGET` y la constante `START` en el registro `LOC_CTRL`, el robot comienza la operación de búsqueda, escribiendo el valor `BUSY` en el registro `LOC_STATUS`.

Al encontrar la caja, la deposita en la bandeja de salida, escribe el valor `FOUNDED` en el registro `LOC_CTRL` y el valor `READY` en el registro `LOC_STATUS`.

Si no puede encontrar la caja, escribe el valor `NOT_FOUND` en el registro `LOC_CTRL` y el valor `READY` en el registro `LOC_STATUS`.

El bot y la caja



Una pequeña empresa de logística acaba de adquirir un **ROBOT** que permite localizar y obtener cajas en su depósito.

Cuando se le ingresa un código en el registro de 32 bits `LOC_TARGET` y la constante `START` en el registro `LOC_CTRL`, el robot comienza la operación de búsqueda, escribiendo el valor `BUSY` en el registro `LOC_STATUS`.

Al encontrar la caja, la deposita en la bandeja de salida, escribe el valor `FOUNDED` en el registro `LOC_CTRL` y el valor `READY` en el registro `LOC_STATUS`.

Si no puede encontrar la caja, escribe el valor `NOT_FOUND` en el registro `LOC_CTRL` y el valor `READY` en el registro `LOC_STATUS`.

En todos los casos el contenido de `LOC_TARGET` se mantiene hasta tanto se vuelva a escribir otro valor.

El bot y la caja

El robot vino con el siguiente **SOFTWARE**:

```
1  int main (int argc, char *argv[]) {
2      int robot = open("/dev/chinbot", "w");
3      int codigo;
4      int resultado;
5      while (1) {
6          printf("Ingrese el código de la caja\n");
7          scanf ("%d", &codigo);
8          resultado = write(robot, codigo);
9          if (resultado == 1) {
10             printf("Su orden ha llegado\n");
11         } else {
12             printf("No podemos encontrar su caja %d\n", codigo);
13         }
14     }
15 }
```

El bot y la caja

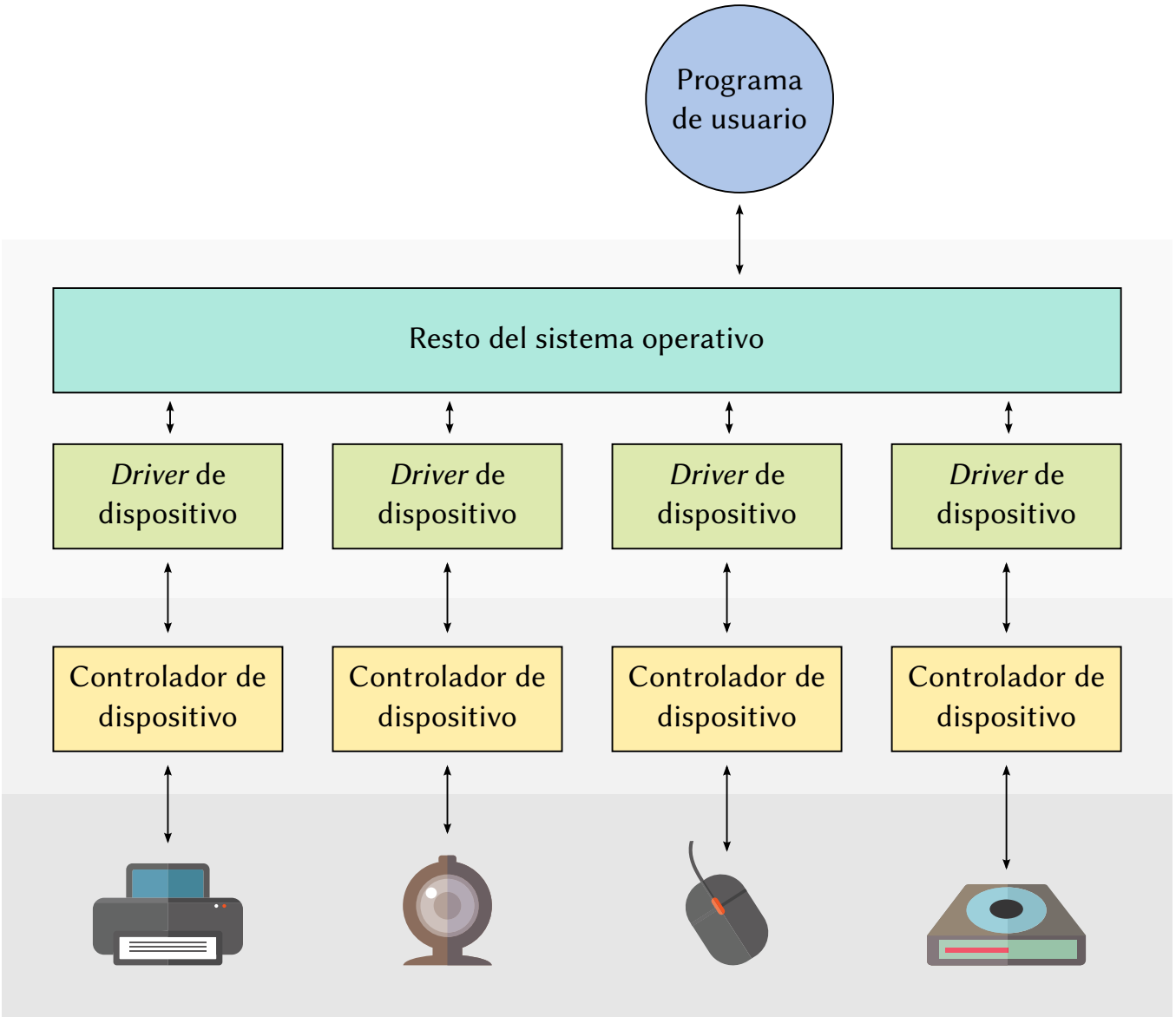
Desafortunadamente, el **DRIVER** que vino con el robot parece no ser compatible con el **SISTEMA OPERATIVO** que utiliza la empresa. Al intentar comunicarse con los fabricantes para obtener soporte, la respuesta no fue la esperada. Por lo tanto, han decidido recurrir a nuestra ayuda.

El bot y la caja

Desafortunadamente, el **DRIVER** que vino con el robot parece no ser compatible con el **SISTEMA OPERATIVO** que utiliza la empresa. Al intentar comunicarse con los fabricantes para obtener soporte, la respuesta no fue la esperada. Por lo tanto, han decidido recurrir a nuestra ayuda.

- Identificar en el siguiente diagrama los elementos resaltados del enunciado.

El SO y los dispositivos de E/S



Pensando nuestro primer driver

- ¿Cuándo el código de usuario que vino con el robot necesita hacer uso del *driver* del dispositivo?

Pensando nuestro primer driver

```
1  int main (int argc, char *argv[]) {
2      int robot = open("/dev/chinbot", "w");
3      int codigo;
4      int resultado;
5      while (1) {
6          printf("Ingrese el código de la caja\n");
7          scanf ("%d", &codigo);
8          resultado = write(robot, codigo);
9          if (resultado == 1) {
10             printf("Su orden ha llegado\n");
11         } else {
12             printf("No podemos encontrar su caja %d\n", codigo);
13         }
14     }
15 }
```

Pensando nuestro primer driver

```
1  int main (int argc, char *argv[]) {
2      int robot = open("/dev/chinbot", "w"); // open device
3      int codigo;
4      int resultado;
5      while (1) {
6          printf("Ingrese el código de la caja\n");
7          scanf ("%d", &codigo);
8          resultado = write(robot, codigo); // write on device
9          if (resultado == 1) {
10             printf("Su orden ha llegado\n");
11         } else {
12             printf("No podemos encontrar su caja %d\n", codigo);
13         }
14     }
15 }
```

Pensando nuestro primer driver

- ¿Cuándo el código de usuario que vino con el robot necesita hacer uso del *driver* del dispositivo?

[1] "Character vs. block devices": `https://tldp.org/LDP/khg/HyperNews/get/devices/basics.html`

Pensando nuestro primer driver

- ¿Cuándo el código de usuario que vino con el robot necesita hacer uso del *driver* del dispositivo?
- ¿Con qué tipo de dispositivo estamos trabajando? ¿Es un *char device*, o un *block device*? [1]

[1] "Character vs. block devices": `https://tldp.org/LDP/khg/HyperNews/get/devices/basics.html`

Pensando nuestro primer driver

- ¿Cuándo el código de usuario que vino con el robot necesita hacer uso del *driver* del dispositivo?
- ¿Con qué tipo de dispositivo estamos trabajando? ¿Es un *char device*, o un *block device*? [1]
- ¿Qué funciones debería proveer el *driver* que programemos?

[1] "Character vs. block devices": `https://tldp.org/LDP/khg/HyperNews/get/devices/basics.html`

La API de un Driver

Un *driver* debe implementar los siguientes procedimientos para ser cargado por el sistema operativo.

- `int driver_init()`

  Invocada durante la carga del SO.

- `int driver_open()`

  Invocada al solicitarse un *open*.

- `int driver_close()`

  Invocada al solicitarse un *close*.

- `int driver_read(int *data)`

  Invocada al solicitarse un *read*.

- `int driver_write(int *data)`

  Invocada al solicitarse un *write*.

- `int driver_remove()`

  Invocada durante la descarga del SO.

Funciones del kernel para drivers

Además para la programación de un *driver*, se dispone de las siguientes *syscalls* (listado NO exhaustivo...):

- `void OUT(int IO_address, int data)`

  Escribe `data` en el registro de E/S.
- `int IN(int IO_address)`

  Devuelve el valor almacenado en el registro de E/S.
- `int request_irq(int irq, void *handler)`

  Permite asociar el procedimiento `handler` a la interrupción `IRQ`. Devuelve `IRQ_ERROR` si ya está asociada a otro *handler*.
- `int free_irq(int irq)`

  Libera la interrupción `IRQ` del procedimiento asociado.

Pensando nuestro primer driver

- ¿Cuándo el código de usuario que vino con el robot necesita hacer uso del *driver* del dispositivo?
- ¿Con qué tipo de dispositivo estamos trabajando? ¿Es un *char device*, o un *block device*?
- ¿Qué funciones debería proveer el *driver* que programemos?
- Pensar, a grandes rasgos, cómo podríamos implementar la función `int driver_write(void* data)` del *driver*.

Pensando nuestro primer driver

```
1  int driver_write(void* data) {
2
3
4
5      OUT(LOC_TARGET, *data);
6      OUT(LOC_CTRL, START);
7
8      while (IN(LOC_STATUS) != BUSY) {}
9      while (IN(LOC_STATUS) != READY) {}
10
11     resultado = IN(LOC_CTRL);
12     if (resultado == FOUND)
13         return 1;
14     else if (resultado == NOT_FOUND)
15         return 0;
16     return -1;
17 }
```

Pensando nuestro primer driver

```
1  int driver_write(void* data) {
2
3
4
5      OUT(LOC_TARGET, *data);
6      OUT(LOC_CTRL, START);
7
8      while (IN(LOC_STATUS) != BUSY) {}
9      while (IN(LOC_STATUS) != READY) {}
10
11     resultado = IN(LOC_CTRL);
12     if (resultado == FOUND)
13         return 1;
14     else if (resultado == NOT_FOUND)
15         return 0;
16     return -1;
17 }
```

- ¿Este código funciona bien?

Ojo con los punteros que nos pasa el usuario

```
1  int driver_write(void* data) {
2      // Copio los datos que me pasa usuario
3      int codigo;
4      copy_from_user(&codigo, data, sizeof(int));
5
6      OUT(LOC_TARGET, codigo);
7      OUT(LOC_CTRL, START);
8
9      while (IN(LOC_STATUS) != BUSY) {}
10     while (IN(LOC_STATUS) != READY) {}
11
12     resultado = IN(LOC_CTRL);
13     if (resultado == FOUND)
14         return 1;
15     else if (resultado == NOT_FOUND)
16         return 0;
17     return -1;
18 }
```

Ojo con los punteros que nos pasa el usuario

```
1  int driver_write(void* data) {
2      // Copio los datos que me pasa usuario
3      int codigo;
4      copy_from_user(&codigo, data, sizeof(int));
5
6      OUT(LOC_TARGET, codigo);
7      OUT(LOC_CTRL, START);
8
9      while (IN(LOC_STATUS) != BUSY) {}
10     while (IN(LOC_STATUS) != READY) {}
11
12     resultado = IN(LOC_CTRL);
13     if (resultado == FOUND)
14         return 1;
15     else if (resultado == NOT_FOUND)
16         return 0;
17     return -1;
18 }
```

- ¿Ahora sí?

Ay, la concurrencia...

```
1  int driver_write(void* data) {
2      int codigo;
3      copy_from_user(&codigo, data, sizeof(int));
4
5      mutex.lock(); // Inicia sección crítica
6      OUT(LOC_TARGET, codigo);
7      OUT(LOC_CTRL, START);
8
9      while (IN(LOC_STATUS) != BUSY) {}
10     while (IN(LOC_STATUS) != READY) {}
11
12     resultado = IN(LOC_CTRL);
13     mutex.unlock(); // Fin sección crítica
14
15     if (resultado == FOUND)
16         return 1;
17     else if (resultado == NOT_FOUND)
18         return 0;
19     return -1;
20 }
```

Cosas para tener en cuenta

- Un *driver* corre dentro del contexto de un proceso.

Cosas para tener en cuenta

- Un *driver* corre dentro del contexto de un proceso.
- Esto significa que puede acceder a sus datos.

Cosas para tener en cuenta

- Un *driver* corre dentro del contexto de un proceso.
- Esto significa que puede acceder a sus datos.
- ¡Cuidado con los punteros que nos pasa el usuario! (`copy_from_user()`, `copy_to_user()`).

Cosas para tener en cuenta

- Un *driver* corre dentro del contexto de un proceso.
- Esto significa que puede acceder a sus datos.
- ¡Cuidado con los punteros que nos pasa el usuario! (`copy_from_user()`, `copy_to_user()`).
- Muchos procesos pueden querer ejecutar el *driver* a la vez. El resultado: horribles *race conditions*.

Cosas para tener en cuenta

- Un *driver* corre dentro del contexto de un proceso.
- Esto significa que puede acceder a sus datos.
- ¡Cuidado con los punteros que nos pasa el usuario! (`copy_from_user()`, `copy_to_user()`).
- Muchos procesos pueden querer ejecutar el *driver* a la vez. El resultado: horribles *race conditions*.
- ¿Cuándo inicializamos las primitivas de sincronización? ¿Y las estructuras de datos que pueda necesitar el *driver*?

Cosas para tener en cuenta

- Un *driver* corre dentro del contexto de un proceso.
- Esto significa que puede acceder a sus datos.
- ¡Cuidado con los punteros que nos pasa el usuario! (`copy_from_user()`, `copy_to_user()`).
- Muchos procesos pueden querer ejecutar el *driver* a la vez. El resultado: horribles *race conditions*.
- ¿Cuándo inicializamos las primitivas de sincronización? ¿Y las estructuras de datos que pueda necesitar el *driver*? Respuesta: al cargar el *driver* en el *kernel* (`driver_init()`).

Cosas para tener en cuenta

- Un *driver* corre dentro del contexto de un proceso.
- Esto significa que puede acceder a sus datos.
- ¡Cuidado con los punteros que nos pasa el usuario! (`copy_from_user()`, `copy_to_user()`).
- Muchos procesos pueden querer ejecutar el *driver* a la vez. El resultado: horribles *race conditions*.
- ¿Cuándo inicializamos las primitivas de sincronización? ¿Y las estructuras de datos que pueda necesitar el *driver*? Respuesta: al cargar el *driver* en el *kernel* (`driver_init()`).
- Un *driver* no se *linkea* contra bibliotecas, así que solo se pueden usar funciones que sean parte del *kernel*.

Métodos de acceso

- ¿Que **método de acceso** emplea nuestro *driver*?

Métodos de acceso

```
1  int driver_write(void* data) {
2      int codigo;
3      copy_from_user(&codigo, data, sizeof(int));
4
5      mutex.lock();
6      OUT(LOC_TARGET, codigo);
7      OUT(LOC_CTRL, START);
8
9      while (IN(LOC_STATUS) != BUSY) {}
10     while (IN(LOC_STATUS) != READY) {}
11
12     resultado = IN(LOC_CTRL);
13     mutex.unlock();
14
15     if (resultado == FOUND)
16         return 1;
17     else if (resultado == NOT_FOUND)
18         return 0;
19     return -1;
20 }
```

Métodos de acceso

```
1  int driver_write(void* data) {
2      int codigo;
3      copy_from_user(&codigo, data, sizeof(int));
4
5      mutex.lock();
6      OUT(LOC_TARGET, codigo);
7      OUT(LOC_CTRL, START);
8
9      while (IN(LOC_STATUS) != BUSY) {} // Polling
10     while (IN(LOC_STATUS) != READY) {} // Polling
11
12     resultado = IN(LOC_CTRL);
13     mutex.unlock();
14
15     if (resultado == FOUND)
16         return 1;
17     else if (resultado == NOT_FOUND)
18         return 0;
19     return -1;
20 }
```

Métodos de acceso

- ¿Que **método de acceso** emplea nuestro *driver*?

Métodos de acceso

- ¿Que **método de acceso** emplea nuestro *driver*?
- Así que ***polling***... ¿Y eso *es bueno o malo*?

Métodos de acceso

- ¿Que **método de acceso** emplea nuestro *driver*?
- Así que ***polling***... ¿Y eso *es bueno o malo*?
- ¿Qué alternativa tenemos? ¿Qué ventajas y desventajas tiene?

Métodos de acceso

- ¿Que **método de acceso** emplea nuestro *driver*?
- Así que ***polling***... ¿Y eso *es bueno o malo*?
- ¿Qué alternativa tenemos? ¿Qué ventajas y desventajas tiene?
- Para poder implementar el *driver* usando **interrupciones**, ¿debería cambiar algo en el *hardware* de nuestro robot?

Métodos de acceso

- ¿Que **método de acceso** emplea nuestro *driver*?
- Así que ***polling***... ¿Y eso *es bueno o malo*?
- ¿Qué alternativa tenemos? ¿Qué ventajas y desventajas tiene?
- Para poder implementar el *driver* usando **interrupciones**, ¿debería cambiar algo en el *hardware* de nuestro robot?
- Parece que el manual del robot, escrito en un dudoso castellano, contiene la siguiente información:

  > "*Robot es compatible con el acceso de interrupción. Se selecciona este modo, una operación terminada* `CHINBOT_INT` *interrupción lanzará.*"

  Aprovechando esta información, modificar el código anterior para que utilice interrupciones.

Interrupciones

```
1  mutex acceso;
2  semaforo listo;
3  bool esperando;
4
5  int driver_init() {
6      acceso = mutex_create();
7      listo = semaforo_create(0);
8      esperando = false;
9      irq_register(CHINBOT_INT, handler);
10 }
11
12 void handler() {
13     if (esperando && IN(LOC_STATUS) == READY) {
14         esperando = false;
15         listo.signal();
16     }
17 }
```

Interrupciones

```
1  int driver_write(void* data) {
2      int codigo;
3      copy_from_user(&codigo, data, sizeof(int));
4
5      acceso.lock();
6      OUT(LOC_TARGET, codigo);
7      OUT(LOC_CTRL, START);
8
9      esperando = true;
10     listo.wait();
11
12     resultado = IN(LOC_CTRL);
13     acceso.unlock();
14
15     if (resultado == FOUND)
16         return 1;
17     else if (resultado == NOT_FOUND)
18         return 0;
19     return -1;
20 }
```