CENG 448 — Real Time Operating Systems

IO

Dr. Larry D. Pyeatt

South Dakota School of Mines and Technology

All computer systems include some sort of input/output.

There are many types of devices that can be used.

Often, embedded systems are designed explicitly to deal with I/O devices.

Cell phone, pager, MP3 player, etc.

System Software Developer

I/O operations imply communicating with the device:

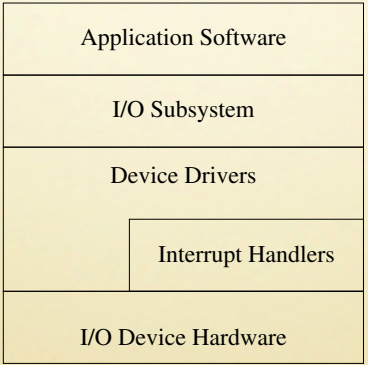
1. program device to initiate I/O request
2. perform the data transfer
3. notify the requestor

System software engineer must understand the physical design (register set and operation) of the device.

The system software engineer(s) develop drivers to allow the applications software engineers to use the devices.

Ideal World

The RTOS contains drivers for all the devices, so the developer just needs to find the correct device, and call the driver functions.



Port mapped vs memory mapped

Many processor families support separate address spaces for memory and I/O devices, and have separate instructions for reading/writing devices and memory.

Other processors just treat devices in the same way as they treat memory.

Many systems include Direct Memory Access (DMA) controller(s), which are programmable devices that can handle moving data between I/O devices and memory without requiring the CPU to do all of the work.

Character vs Block devices

Character devices typically input/output one byte at a time. UART, keyboard, mouse, etc.

Block devices can only input/output blocks of data. The size of a block is determined by the device hardware. Disk drives, SD cards, etc.

Block device drivers typically use some sort of caching technique to allow the user to read/write portions of a block.

I/O Subsystem

Each I/O device driver can provide driver-specific application interface, which means that the applications programmer must learn how to use the devices individually.

It is much better if the system's programmer can define a standard set of functions.

1. The I/O subsystem defines a standard API set.
2. A device driver implements the functions for a specific device.
3. The device driver does the work necessary to prepare the device for use, and registers the functions with the I/O subsystem.
4. The I/O system assigns a unique ID to each device.
5. The applications program can make a standard API call, passing the ID of the device to the I/O subsystem, which calls the matching function for the specified device.

Standard Functions

create	creates an instance of an I/O device
destroy	deletes an instance of an I/O device
open	prepares I/O device for use
close	clean up and shut down the device
read	transfer one or more bytes from the device to memory
write	transfer one or more bytes from memory to the device
ioctl	issues control commands to modify the operation of the device. the ioctl function may accept device-specific parameters.

C implementation

```
typedef struct {
  int (*create)();
  int (*destroy)();
  int (*open)();
  int (*close)();
  int (*read)();
  int (*write)();
  int (*ioctl)();
}dev_driver;
```

```
dev_driver *my_driver = malloc(sizeof(dev_driver));
my_driver->create = my_create_function;
my_driver->destroy = my_destroy_function;
my_driver->open = my_open_function;
my_driver->close = my_close_function;
my_driver->read = my_read_function;
my_driver->write = my_write_function;
my_driver->ioctl = my_ioctl_function;
```

Using the device

```
1    int descriptor = open(my_dev_id);
2    write(descriptor,"Hello World",strlen("Hello World"));
3    close(descriptor);
```