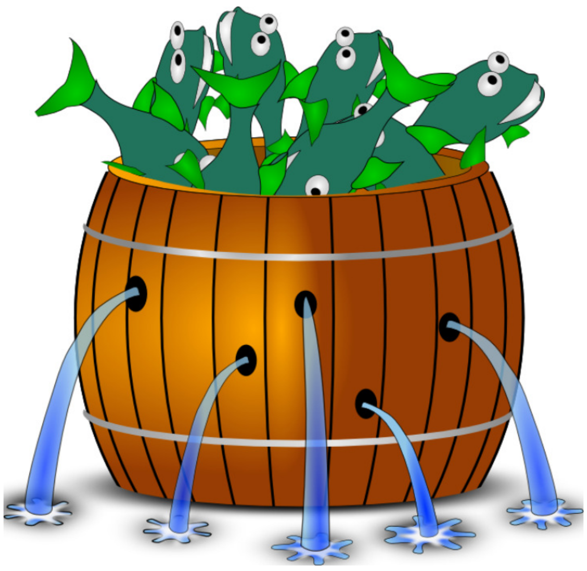
Barrelfish Project
ETH Zurich



Serial ports in Barrelfish

Barrelfish Technical Note 016

Timothy Roscoe

08.07.2012

Systems Group
Department of Computer Science
ETH Zurich
CAB F.79, Universitätstrasse 6, Zurich 8092, Switzerland
`http://www.barrelfish.org/`

Revision History

Revision	Date	Author(s)	Description
0.0	08.07.2012	TR	Initial version

Contents

Chapter 1

Introduction

This technical note describes the use of serial ports (UARTS) in Barrelfish.

At present, it only describes the kernel-space interface to serial ports, used internally by CPU drivers for debugging and console output.

In the future, it will also describe the interface to user-space UART drivers and other console subsystems.

Chapter 2

CPU driver interface

CPU drivers use serial port access for printing debug information, status messages during startup, and connecting a debugger such as GDB to a processor.

Each CPU driver implements a serial port subsystem which provides a small number serial ports to the rest of the CPU driver.

The interface to this subsystem is specified in the file `/kernel/include/serial.h`, and is described in this chapter. Check the header file for a (possibly) more-up-to-date description.

The interface is intentionally very basic. It assumes each UART runs at a fixed, default speed with basic settings (typically 115000 baud, 8-bit words, no parity, no flow control). Serious applications should use a full user-space UART driver instead.

2.1 Physical ports

Serial ports available inside a CPU driver are numbered starting at zero.

The number of available serial ports is given by:

```
extern const unsigned serial_num_physical_ports;
```

2.1.1 Initialization

Each port must be initialized before it can be used:

```
extern errval_t serial_init(unsigned port);
```

This call should fail with `SYS_ERR_SERIAL_PORT_INVALID` if the port cannot be initialized, or the port number is out of range.

Initializing a serial port more than once, however, is permissible: subsequent attempts to initialize an initialized port are silently ignored. The motivation for this is to handle the case where the console and debug logical ports share the same physical port.

In some cases it is necessary to do an "early" initializaton, for example while running in physical address space before the MMU is enabled. In such cases the CPU driver should call, and the code implement, the following:

```
extern errval_t serial_early_init(unsigned port);
```

This returns the same errors as `serial_init()`, but it is a bug to call this function more than once with the same port number, and doing so will cause an assertion failure.

2.1.2 Input/output

Each serial port provides polled, blocking, single-character programmed I/O:

```
extern void serial_putchar(unsigned port, char c);
extern char serial_getchar(unsigned port);
```

It is a bug to call these with a port number that is out of range (i.e. one that is greater than or equal to `serial_num_physical_ports`), and doing so will cause an assertion failure. It may also cause an assertion failure to call these functions on a serial port which has not yet been initialized with a call to `serial_init()` or `serial_early_init()`.

Note that implementations of these functions should not attempt to use DMA or interrupts. They should also not buffer any data in software (though they may use a hardware FIFO in the UART if appropriate).

2.2 Logical ports

Barrelfish CPU drivers have access to two logical serial ports:

- The *console* port is used for general logging and debug information. It is principally called by `kputchar()`, which implements line-buffering and is in turn called by the kernel `printf()` function.
- The *debug* port can be used to attach an external debugger like `gdb`.

Both logical ports are assigned to sensible default physical ports by the serial port implementation, but they can be changed by writing these variables:

```
extern unsigned serial_console_port;
extern unsigned serial_debug_port;
```

Note that it is not unusual for both logical ports to be assigned to the same physical port.

The file `/kernel/include/serial.h` also includes inline functions to initialize whichever physical port is currently assigned to a logical port:

```
static inline errval_t serial_console_init(void);
static inline errval_t serial_debug_init(void);
```

There are also input functions, which simply call `serial_getchar()` with the appropriate port number:

```
static inline char serial_console_getchar(void);
static inline char serial_debug_getchar(void);
```

Output functions are similar, but in addition replace any LF character with the sequence CR/LF:

```
static inline void serial_console_putchar(char c);
static inline void serial_debug_putchar(char c);
```

2.3 Implementing a serial port subsystem

Typically, a rudimentary console driver is the first thing to be implemented when writing a CPU driver for a new architecture.

The serial port subsystem for a CPU driver must implement all the functionality in `/kernel/include/serial.h` that deals with physical serial ports, in other words:

```
const unsigned serial_num_physical_ports;
errval_t serial_init(unsigned port);
errval_t serial_early_init(unsigned port);
void serial_putchar(unsigned port, char c);
char serial_getchar(unsigned port);
unsigned serial_console_port;
unsigned serial_debug_port;
```

Functions for logical ports are provided by inline functions in the header file.

Initializing a serial port typically consists of configuring the UART hardware, and then mapping the registers appropriately into the kernel's virtual address space.

Examples of serial port implementations at time of writing include:

- `/kernel/arch/omap44xx/omap_uart.c`
- `/kernel/arch/x86/serial.c`

2.3.1 Multiprocessor considerations

This document does not address accessing a single UART from more than CPU driver. If this happens at all, it is typically hidden by the implementation (for example, some x86 CPU drivers use a global spinlock, since performance is not an issue anyway). Care should be taken not to initialize the UARTs multiple times in such circumstances.