OS161

File System and IO

OS161 File System

The file system implementation has two directories

- Virtual file system: kern/vfs/
  file system independent layer. A framework into which you can add new file systems easily.
  Go look at vfs.h and vnode.h before looking at this directory.

- Actual file system: kern/fs/
  This is where the actual file system implementations go. The subdirectory sfs contains the implementation of the simple file system. Emufs not present here, as bypassed (go directly to device drivers).

IO devices - LAMEBUS

LAMEbus is a simple system bus architecture:

- 32 slots, each of which is associated with a fixed-size addressable region.
- regions are mapped in order into the system physical memory space.
- no DMA; data transfers are done by the CPU using transfer buffers appearing within a device's address space. (However, DMA may be added in a future version.)

Kern/dev/lamebus/lamebus.h

```
/* CS161 devices */
#define LBCS161_UPBUSCTL     1
#define LBCS161_TIMER        2
#define LBCS161_DISK         3
#define LBCS161_SERIAL       4
#define LBCS161_SCREEN       5
#define LBCS161_NET          6
#define LBCS161_EMUFS        7
#define LBCS161_TRACE        8
#define LBCS161_RANDOM       9
#define LBCS161_MPBUSCTL     10
```

Structure of the Bus Region

There are 32 slots, and each slot has a 64K address space. Thus, the whole bus region takes 64K*32 = 2MB.

The base physical address of this region is called LAMEBASE; because different processor types impose different restrictions on the organization of physical memory, its value depends on the processor architecture.

The bus controller always appears in slot 31; its 64K address space is divided in two.

- The lower half is divided into 32 1K config regions, one per slot.
- The upper half is divided into 32 1K control regions, one per CPU. In addition to configuration, the config region of the bus controller's config region contains the bus controller's own registers, described below.

Full documentation

- SFS
  - http://os161.eecs.harvard.edu/resources/sfs.html
- LAMEBUS
  - http://os161.eecs.harvard.edu/documentation/sys161/lamebus.html
- HARDWARE devices
  - http://os161.eecs.harvard.edu/documentation/sys161/devices.html

Serial Console

- A very simple console interface. Writing to the first register causes a character to be printed. Reading from it returns the last character typed.
- You must wait until one write has completed before starting another one, or the output may be garbled.

Device id	4
Oldest revision	1
Current revision	1

Registers	
Offset	**Description**
0-3	Character buffer
4-7	Write IRQ register
8-11	Read IRQ register
12-15	Reserved

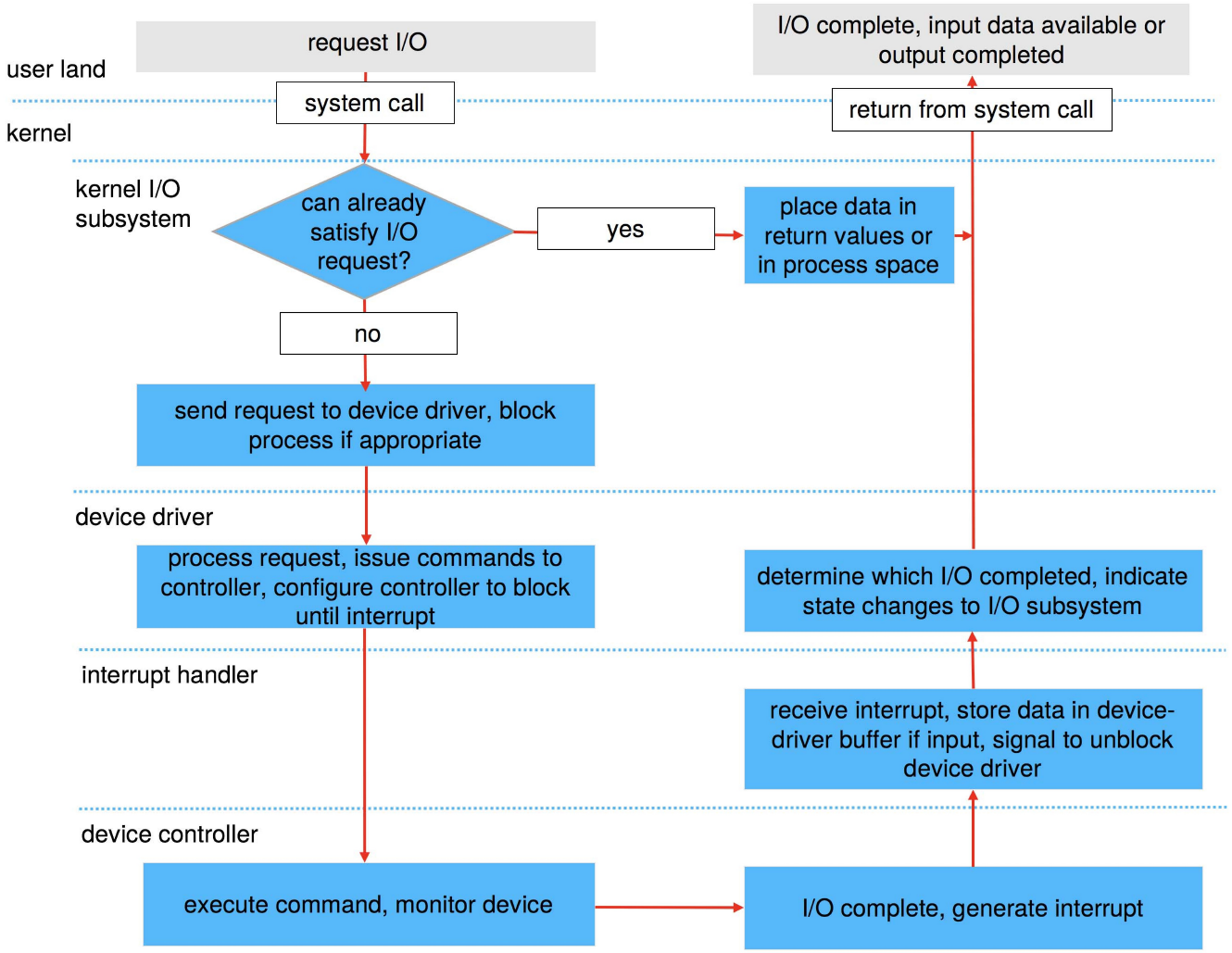
lser.c

```
/* Registers (offsets within slot) */
#define LSER_REG_CHAR  0     /* Character in/out */
#define LSER_REG_WIRQ  4     /* Write interrupt status */
#define LSER_REG_RIRQ  8     /* Read interrupt status */

/* Bits in the IRQ registers */
#define LSER_IRQ_ENABLE  1
#define LSER_IRQ_ACTIVE  2
#define LSER_IRQ_FORCE   4

...
/* functions: lser_irq, lser_write, ... */
```

lser.h

```
struct lser_softc {
    ...
    /* Initialized by lower-level attachment function */
    void *ls_busdata;
    uint32_t ls_buspos;
    /* Initialized by higher-level attachment function */
    void *ls_devdata;
    void (*ls_start)(void *devdata);
    void (*ls_input)(void *devdata, int ch);
};
/* Functions called by lower-level drivers */
void lser_irq(/*struct lser_softc*/ void *sc);
/* Functions called by higher-level drivers */
void lser_write(/*struct lser_softc*/ void *sc, int ch);
void lser_writepolled(/*struct lser_softc*/ void *sc, int ch);
```

Life Cycle of An I/O Request



putch()

```
putch(int ch) {
  struct con_softc *cs = the_console;
  ...
  putch_intr(cs, ch);
}
```

console.c .h (kernel IO subsystem)

```
putch_intr(struct con_softc *cs, int ch) {
  P(cs->cs_wsem);
  cs->cs_send(cs->cs_devdata,ch);
}
```

```
con_start(void *vcs) {
  struct con_softc *cs = vcs;
  V(cs->cs_wsem);
}
```

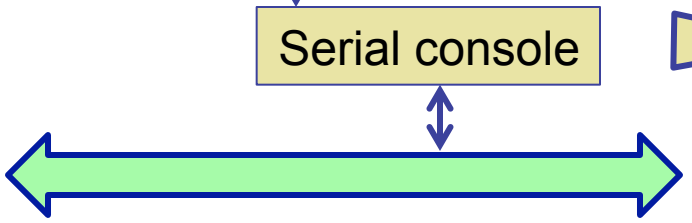
lser.c .h (driver)

```
lser_write(void *vls, int ch) {
  struct lser_softc *ls = vls;

  bus_write_register(ls->ls_busdata,
    ls->ls_buspos, LSER_REG_CHAR, ch);
}
```

```
lser_irq(void *vsc) {
  struct lser_softc *ls = vsc;
  sc->ls_start(sc->ls_devdata);
}
```

Serial console

```
/* write done interrupt */
lamebus_interrupt(…) {
  handler(data);
}
```



getch()

```
int getch(void) {
  struct con_softc *cs = the_console;
  ...
  return getch_intr(cs);
}
```

console.c .h (kernel IO subsystem)

```
getch_intr(struct con_softc *cs) {
  P(cs->cs_rsem);
  /* read from buffer and return */
}
```

```
con_input(void *vcs, int ch) {
  struct con_softc *cs = vcs;
  /* write ch to buffer */
  V(cs->cs_rsem);
}
```

lser.c .h (driver)

```
lser_irq(void *vsc) {
  struct lser_softc *ls = vsc;
  sc->ls_input(sc->ls_devdata, ch);
}
```

Serial console

```
/* write done interrupt */
lamebus_interrupt(…) {
  handler(data);
}
```