

Kernel – IPC, IO and ISR programming
User space device driver programming

Programming the kernel needs the knowledge of the following main areas:

- General device driver management
- I/O port management
- Hardware interrupt control
- Thread management and thread synchronisation
- Memory management
- Remote interprocess communication (IPC)
- Local interprocess communication (RPC)

User space device driver programming

It's also possible to build any kind of device driver in user space. The device driver is just a generic user program and can be started as any usual process under Amoeba. The only difference (if at all): the user space device driver needs a special capability to gain control over I/O ports and for requesting interrupts (currently the kernel root capability for simplicity).

Name

User space port access

Synopsis

Programming Interface Header files

```
#include "ioport.h"  
#include "sys/iomap.h"
```

Programming Interface I/O access routines

```
void out_byte ( int _port ,  
               int _val );  
void out_word ( int _port ,  
               int _val );  
void out_long ( int _port ,  
               int _val );  
int in_byte ( int _port );  
int in_word ( int _port );  
long in_long ( int _port );  
void outs_byte ( int _port ,  
                char * _ptr ,  
                int _bytecnt );  
void outs_word ( int _port ,  
                char * _ptr ,  
                int _bytecnt );  
void outs_word_l ( int _port ,  
                  char * _ptr ,  
                  int _wordcnt );  
void outs_long_l ( int _port ,  
                  char * _ptr ,  
                  int _longcnt );  
void ins_byte ( int _port ,  
               char * _ptr ,  
               int _bytecnt );  
void ins_word ( int _port ,  
               char * _ptr ,  
               int _bytecnt );
```

```
void ins_word_l( int _port ,
                char *_ptr ,
                int _wordcnt );

void ins_long_l( int _port ,
                char *_ptr ,
                int _longcnt );
```

Programming Interface Port management

```
errstat io_check_region( unsigned int start ,
                        unsigned int size ,
                        capability *syscap );

errstat io_map_region( unsigned int start ,
                      unsigned int size ,
                      char *devname ,
                      capability *syscap );

errstat io_unmap_region( unsigned int start ,
                       unsigned int size ,
                       capability *syscap );

errstat io_vtop( long vaddr ,
                long vlen ,
                long *paddr ,
                capability *syscap );

errstat io_setpvl( int pvl ,
                  capability *syscap );
```

Description

Low level input and output routines for hardware port access. Different types are supported. See generic i386 processor and assembler documents for more details.

Port management

Before a user process can access I/O ports, in contrast to device drivers within the kernel, the process must register and map the desired I/O region. Because only one process can map a specific I/O region, the **io_check_region** function must be called to check the availability of the resources. After this function returns the **STD_OK** status, the **io_map_region** function can be used to map and register the I/O port region. After this call, I/O ports can be accessed with the below explained functions.

Warning: the **devname** argument specified with the **io_map_region** function MUST be allocated with the **malloc/alloc** functions, or the process will be terminated with an exception (that means, the devname string must be outside of readonly text segments!).

Either implicitly on process exit, or explicitly, the I/O region can be unmapped with the **io_unmap_region** function.

The **io_setpvl** function can be used to gain full I/O access for a program with changing the IOPL. The **io_vtop** function translates a virtual process address region of specified length to the physical (real)

address. This is needed for DMA transfers, for example. The memory region can be a mapped hardware segment, too.

Port access

Bytes of length 8bit, words of length 16bit, and longs of length 32bit can be read and written with the respective functions declared above. There are some additional functions for old and slow hardware, inserting a short delay: `out_##_p()`, `in_##_p()`. Memory region can be copied to and from hardware ports with the `outs_##()`, `ins_##()` functions. Take care about the count parameter!

Example

Here is a short example for accessing I/O ports through user processes.

Example User process I/O

```
#include <sys/iomap.h>
#include <ioport.h>

#define KERNELSYSCAP "/hosts/dio01"
#define MYNAME "MYSERVER"

int main()
{
    char *devname=malloc(sizeof(MYNAME)+1);
    errtstat err;
    capability syscap;

    err=name_lookup(KERNELSYSCAP,&syscap);
    if (err != STD_OK)
    {
        failwith("Can't lookup kernel cap");
    }
    err = io_checkregion(0x278,4,&syscap);

    if (err != STD_OK)
    {
        failwith("I/O region already used");
    }

    strcpy(devname,MYNAME);
    err = io_map_region(0x278,4,devname,&syscap);

    if (err != STD_OK)
    {
        failwith("IO mapping failed...");
    }

    out_byte(0x278,0xff);

    /* all done */

    io_unmap_region(0x278,4,&syscap);
}
```

```
return 0;  
}
```

Table of Content

Kernel programming.	2
User space device driver programming	3
Man-Page: I/O port management (i386)	4
Port management.	5
Port access	6
Example	6