BSD 4.3 Unixserver and Emulator Internals

Randall W. Dean

Carnegie Mellon

Mach

Outline

- Overview of new interfaces
    - Trap Redirection Mechanism
    - Emulator
    - Syscall Handler
    - IO Glue
- Unix System Code Internals
    - Overall Threading Structure
    - Internal Emulation
    - Difficult System Calls
- Getting Started
- Further Complications
- Problems

Trap Redirection Mechanism

- task_set_emulation

```
for (i = MIN_ENTRY; i <= MAX_ENTRY; i++) {
        rc = task_set_emulation(task,
                                emul_vector_base[i],
                                i);
}
rc = task_set_emulation(task,
                        emul_task_by_pid,
                        -33);
rc = task_set_emulation(task,
                        emul_pid_by_task,
                        -34);
```

- Fast return to user space
- Registers and stack unchanged

Emulator

- Runs on own stack
- Generic syscall
  - Copy arguments off stack
  - Send generic message
  - Copy data back to user
- Other syscalls
  - Massage arguments
  - Send bsd_* message
  - Massage returned data

Examples

```
int
e_open(proc_port, interrupt, fname, mode, crtmode, rval)
        mach_port_t     proc_port;
        boolean_t       *interrupt;
        char            *fname;
        int             mode;
        int             crtmode;
        int             *rval;
{
        return bsd_open(proc_port,
                        interrupt,
                        fname, strlen(fname) + 1,
                        mode,
                        crtmode,
                        rval);
}

int
e_write(serv_port, interrupt, fileno, data, count, rval)
        mach_port_t     serv_port;
        boolean_t       *interrupt;     /* out */
        int             fileno;
        char            *data;
        unsigned int    count;
        int             *rval;
{
        extern int      bsd_write_short();
        extern int      bsd_write_long();

        return (((count <= SMALL_ARRAY_LIMIT)
                  ? bsd_write_short
                  : bsd_write_long
                )(serv_port,
                  interrupt,
                  fileno,
                  data,
                  count,
                  &rval[0])
               );
}
```

Syscall Handler

- Identify Process
- Unix Master
- Collect Arguments
- Issue Syscall
- Fixup Return Code
- Return Reply Message

IO Glue

- bdevsw

```
#define bdev_ops        bdev_open, bdev_close, bio_strategy

struct bdevsw   bdevsw[] =
{
        /*0*/   { "ip",   C_BLOCK(8), bdev_ops },
};
```

- cdevsw

```
#define disk_ops \
        disk_open, disk_close, disk_read, disk_write, disk_ioctl, \
        seltrue, nulldev, nulltty

struct cdevsw   cdevsw[] =
{
        /*3*/   { "ip", C_BLOCK(8), disk_ops, },
}
```

- Device Reply Handler
- Network Input Thread

Device Reply Handler

- Wired at elevated priority
- Handles reply half of device_* calls
- Used by ttys
- Calls interface completion routine

Network Input Thread

- Wired at elevated priority
- Receiving end of Packet Filter
- Wraps mbuf around packet
- Enqueues packets to be handled later

Overall Threading Structure

- Net Input Thread
- Device Reply Handler
- Softclock Thread
- Vnode Pager
- Pool of User Threads
    - Not bound until receives message
    - Uarea on stack
    - Registered internally
    - Runs in context of calling proc for duration

Internal Emulation

- SPL’s
  - Array of Locks and Conditions
  - Must acquire in increasing order
  - Must release in descending order
- Interrupts
  - Grabs SPL desired w/o acquiring imtermediate
  - Packet interrupts special case
- Copyin/Copyout

Difficult System Calls

- Exec()
- Fork()
- Signals

Exec()

- Allocates space in emulator and saves arguments
- Calls server
- Destroys address space except emulator region
- Load new image
- Return to emulator
  - Returns entry point
  - Returns new arguments if shell script
- Build argument list on new stack
- Return to new image entry pont

Fork()

- Create copy of registers
- Call bsd_fork with that copy
- Create new task inheriting from parent
- Set registers as specified by parent emulator
- Continue the child
- Return to parent
- Childs entry point is emulator initialization code

Signals

- Asynchronous
  - Posted internally only
  - Registered thread notified if present
  - Checked for at beginning and end of all system calls
  - If signal pending, intterupt parameter to syscall set
  - Emulator initiates signal delivery
- Synchronous
  - User thread stopped because of exception
  - UnixServer is exception handler for user tasks
  - Builds signal frame and forces on user
  - Continues user which now takes the exception

Getting Started

- Bootstrapping Server
  - Started by kernel as first user task
  - Server sends message to bootstrap port to receive privuleged_host_port and device_server_port
  - Starts internal threads
  - Loads first task
  - Turns into first server pool thread
- Loading emulator and starting Init
  - Create task
  - Load emulator into high memory with exec primitives
  - Load name of first program onto stack as arguments
  - Start emulator
  - Emulator "execs" the first program
  - Emulator code inherited by all subsequent tasks

Further Complications

- Mapped Uarea
- Mapped Files

Mapped Uarea

- Structure
  - Readonly shared page
  - Readwrite shared page
  - Datacopy readwtite shared page
- Difficulties
  - Locking
  - Security

Shared data structures

```
struct ushared_ro {
        int             us_proc_pointer;
        int             us_version;
        int             us_mach_nbc;
        int             us_nofile;              /* NOFILE */
/* user fields */
        struct          rlimit us_rlimit[RLIM_NLIMITS];
                                                /* uu_rlimit */
        caddr_t         us_data_start;          /* uu_data_start */
        uid_t           us_uid;                 /* uu_uid */
        uid_t           us_ruid;                /* uu_ruid */
        gid_t           us_gid;                 /* uu_gid */
        gid_t           us_rgid;                /* uu_rgid */
        gid_t           us_groups[NGROUPS];     /* uu_groups */
        int             us_lastfile;            /* uu_lastfile */
/* proc fields */
        int             us_flag;                /* p_flag */
        short           us_pid;                 /* p_pid */
        short           us_ppid;                /* p_ppid */
        short           us_cursig;              /* p_cursig */
};

struct ushared_rw {
        int             us_inuse;
        int             us_map_files;
        int             us_debug;
        shared_lock_t   us_lock;
/* file fields */
        struct file_info us_file_info[NOFILE];
        int             us_closefile;
/* user fields */
        size_t          us_dsize;               /* uu_dsize */
        short           us_cmask;               /* uu_cmask */
        struct          sigstack us_sigstack;   /* uu_sigstack */
        int             (*us_signal[NSIG+1])(); /* uu_signal */
        int             us_usigmask[NSIG+1];    /* uu_sigmask */
        int             us_sigonstack;          /* uu_sigonstack */
        int             us_sigintr;             /* uu_sigintr */
        int             (*us_sigtramp)();       /* uu_sigtramp */
/* proc fields */
        shared_lock_t   us_siglock;             /* p_siglock */
        int             us_sigmask;             /* p_sigmask */
        int             us_sig;                 /* p_sig */
        int             us_sigignore;           /* p_sigignore */
        int             us_sigcatch;            /* p_sigcatch */
};
```

Shared Locks

- Server cant trust users

- ```
  typedef struct shared_lock {
          spin_lock_t lock;
          int who;
          int need_wakeup;
  } shared_lock_t;

  int share_lock_solid(x)
          register struct shared_lock *x;
  {
          unix_master();
          x->need_wakeup++;
          if (spin_try_lock(&x->lock)) {
                  x->need_wakeup--;
                  unix_release();
                  return 1;
          }
          if (tsleep(x, SHARED_PRIORITY, BACKOFF_SECS * hz) == TS_TIME) {
                  if ((x->who & ~KERNEL_USER >= (int)proc) &&
                      (x->who & ~KERNEL_USER <= (int)procNPROC)) {
                      printf("[%x]Unable to get lock from %x\n",(int)x, x->who);
                      panic("share_lock_solid");
                  } else {
                      printf("[%x]Taking scribbled share_lock\n",(int)x);
                      share_lock_init(x);
                      x->need_wakeup++;
                  }
          }
          (x)->need_wakeup--; /* This protected by the master lock */
          unix_release();
          return 0;
  }
  ```
  ```

Mapped Files

- Backed by pager
- Mapped on open
- File pointer and size in mapped uarea
- Uses token scheme for sharing
- Interface to server
    - maprw_request_it
    - maprw_release_it
    - maprw_remap

Problems

- Signal semantics not correct
- Security problems with setuid
- New failure modes