SYSTEM CALL IN MINIX

Zhen Mo
What is the MINIX System?

- Mini Unix (Minix) basically, a UNIX compatible operating system.
- Open source: intend to be studied in universities
  - Very small (kernel is under 4000 lines)
  - Simple
  - Design to be readable (thousands of comments)
- Written in C, with some very small parts in assembly language
- Minix is a microkernel-based operating system.
What is kernel?

- A program
- Central component of operating systems
- Manages the system's resources
- Lowest-level abstraction layer for the resources (especially memory, processors and I/O devices) that application software must control to perform its functions.
What is microkernel?
What is the Minix 3 microkernel Architecture?

The MINIX 3 Microkernel Architecture
What is a System Call in Minix?

- **A system call** in Linux is how a program requests a service from the kernel.

- There are three types of calls in Minix:
  - **A system call** in Minix is how a program requests a service from a server and/or a driver.
  - **A kernel call** in Minix is how a server or driver requests a service from the kernel.
  - **Message/IPC/trap** in Minix is used for interprocess communication.
Servers

- **Reincarnation Server (RS):** responsible for the reliability of the entire operating system.
- **Datastore Server (DS):** DS provides a persistent storage of server state in memory.
- **Virtual Memory server (VM):** responsible for managing both virtual and physical memory mappings.
- **Process Management server (PM):** responsible for creating, destroying and managing processes in MINIX.
Servers

- **Virtual File system Server (VFS):** responsible for providing a unified interface to all mounted file systems in the system.

- **Peripheral Component Interconnect Server (PCI):** allows device drivers to access devices on the PCI bus.

- **Internet Network Server (INET):** responsible for the implementation of network protocols.
Drivers

- **Terminal Driver (TTY):** TTY is responsible for the operation of the system console:
  - Keyboard/Screen
  - Serial: serial cable
  - Pseudo: OpenSSH

- **Disk Driver:** The disk driver reads and writes disk blocks from and to the local disk(s)

- **Memory Driver:** The memory device driver is used during the bootstrapping of MINIX to serve as an initial file system. It contains configuration files and programs needed to startup MINIX

- **Network Driver:** MINIX supports various types of network cards. Each type of network card is implemented in a network device driver.
Inter Process Communication

- Is handled by the kernel

- A process sends a destination and a message to the kernel, which then copies the message to destination process

- A process must be waiting for the message in order to receive

- `send`

- `receive`

- `sendrec` (user processes are only allowed to use this one)
MINIX 3 source file organization

- **usr/src/kernel** - layer 4 (scheduling, messages and IO)
- **usr/src/drivers** - layer 3 (device drivers for disk, console, printer, other drivers)
- **usr/src/servers** - layer 2 (process manager, file system, other servers)
- **usr/src/lib** - source code for library procedures (open, read, etc)
- **usr/src/include** - all kinds of header files
- Each directory has its own Makefile
Learn MINIX source code

• Download the source code:
  • [https://github.com/minix3/minix](https://github.com/minix3/minix)

• Use code editor and code browser:
  • Windows: source insight
  • Linux: slickedit
Example - Kill

main():
    kill((pid_t) pid, (int) numsig)

    bin/kill.c

kill():
    _syscall(PM_PROC_NR, KILL, message)

    lib/libc/sys-minix/kill.c

    #define PM_PROC_NR ((endpoint_t) 0)
    #define VFS_PROC_NR ((endpoint_t) 1)
    #define RS_PROC_NR ((endpoint_t) 2)

    include/minix/Com.h

_syscall():
    sendrec(PM_PROC_NR, message)

    lib/libc/sys-minix/Syscall.c

    #define GETGROUPS 34
    #define SYNC 36
    #define KILL 37

    include/minix/Callnr.h

    Type def struct message

    include/minix/IPC.h
Example - Kill

main():
(*call_vec[KILL])()
usr/src/servers/pm/main.c

do_kill():
check_sig(m_in.pid, m_in.sig_nr, FALSE)
usr/src/servers/pm/signal.c

check_sig():
sig_proc(rmp, signo, TRUE, ksig)
usr/src/servers/pm/signal.c

int do_kill(void)
usr/src/servers/pm/proto.h

do_kill 37
usr/src/servers/pm/Table.c

sig_proc():
sys_kill(rmp->mp_endpoint, signo)
usr/src/servers/pm/signal.c
Example - Kill

```c
check_sig(
    struct mproc *rmp;  // process pointer to signal
    struct mproc *mp;   // caller process pointer

    /* Check for permission. */
    if (mp->mp_effuid != SUPER_USER
        && mp->mp_realuid != rmp->mp_realuid
        && mp->mp_effuid != rmp->mp_realuid
        && mp->mp_realuid != rmp->mp_effuid
        && mp->mp_effuid != rmp->mp_effuid) {
        error_code = EPERM;
        continue;
    }

    usr/src/servers/pm/signal.c
```
Example - Kill

EXTERN struct mproc {

/* Real and effective uids and gids. */
  uid_t mp_realuid;  /* process' real uid */
  uid_t mp_effuid;  /* process' effective uid */
  gid_t mp_realgid;  /* process' real gid */
  gid_t mp_effgid;  /* process' effective gid */

/* Supplemental groups. */
  int mp_ngroups;  /* number of supplemental groups */
  gid_t mp_sgroups[NGROUPS_MAX];  /* process' supplemental groups */

};

usr/src/servers/pm/mproc.c
Compile source code

- su
- cd /usr/src/releasetools
- make install
- reboot
- Press 2

- Create a tar file in MINIX:
  - tar -cf file.tar file1 file2 file3