Assignment 2
The Basic ASST2 Spec

• Implement open(), read(), write(), lseek(), close(), and dup2()
  • Assume you need to support fork()
    • Document the concurrency issues introduced by fork()
    • However, you should not synchronise the actual code
      • Can assume we will only test with a single process at a time.
    • Your data structures should not need significant changes to support fork()
      • Except for synchronisation
  • User-level exists
    • asst2
    • C libraries
  • An existing framework and code for:
    • system call dispatching,
    • VFS
    • Emufs
    • drivers
Overview

• Overall structure
  • User-level
    • Process structure
  • In-kernel
    • The storage stack
    • Overview of VFS and emufs functionality

• Details
  • Understanding the system interface
  • Argument passing
  • System call dispatching
  • Moving data across the user-kernel boundary
  • Connecting the interface to the VFS
Structure of a Computer System

- Application
- System Libraries
- Userland/testbin/asst2
- asst2.c
- userland/lib/libc
- User Mode
- Kernel Mode
- Device
- OS/161 kern
- Memory
R3000 Address Space Layout

- ksegX not accessible in usermode
- Switching processes switches the application view of memory (translation stored in a page table) for kuseg
Process Layout

- Where is asst2 code/data (from asst2.c)?
Calling open()

```c
int open(const char *filename,
         int flags, ...);
```

- Where is the function “open()”?
Structure of a Computer System

- Application
- System Libraries
- OS/161
- Device
- Memory

Interaction via System Calls

User Mode
Kernel Mode
OS/161 storage stack

- asst2
- emufs
- sys161
- ~/cs3231/root

Application

Syscall dispatching
- FD table
- OF table
- VFS
- FS
- Device driver
`open()`?

```c
int open(const char *filename, int flags, ...);
```

- Where is “open()’s” implementation?
- By convention, it’s called `sys_open()` in the kernel.

This is what you are implementing in ASST2
Existing storage stack

- Application
  - Syscall dispatching
  - FD table
  - OF table
  - VFS
  - FS
  - Device driver

- asst2
- emufs
- sys161
- ~/cs3231/root
OS 161 root filesystem

Provided Storage Stack

Host filesystem

VFS emufs driver sys161
Details
System Call Interface

int open(const char *filename, int flags);
int open(const char *filename, int flags, mode_t mode);
int close(int fd);
ssize_t read(int fd, void *buf, size_t buflen);
ssize_t write(int fd, const void *buf, size_t nbytes);
int dup2(int oldfd, int newfd);
off_t lseek(int fd, off_t pos, int whence);

Solution should work with fork() if implemented
pid_t fork(void);
open/close

int open(const char *filename, int flags);
int open(const char *filename, int flags, mode_t mode);
int close(int fd);
Read/write

ssize_t read(int fd, void *buf, size_t buflen);
ssize_t write(int fd, const void *buf, size_t nbytes);
dup2

int dup2(int oldfd, int newfd);
lseek

off_t lseek(int fd, off_t pos, int whence);
fork

pid_t fork(void);
Argument passing

#include <unistd.h>

int reboot(int code);

Description
reboot reboots or shuts down the system. The specific action depends on the code passed:

- RB_REBOOT The system is rebooted.
- RB_HALT The system is halted.
- RB_POWEROFF The system is powered off.

Return Values
On success, reboot does not return. On error, -1 is returned, and errno is set according to the error encountered.
Preserved for C calling convention

Success?

Result

SysCall No.
By creating a pointer to here of type `struct trapframe *`, we can access the user’s saved registers as normal variables within ‘C’
syscall(struct trapframe *tf)
{
    callno = tf->tf_v0;
    retval = 0;

    switch (callno) {
        case SYS_reboot:
            err = sys_reboot(tf->tf_a0);
            break;

        /* Add stuff here */

        default:
            kprintf("Unknown syscall %d\n", callno);
            err = ENOSYS;
            break;
    }
}
if (err) {
    tf->tf_v0 = err;
    tf->tf_a3 = 1; /* signal an error */
}
else {
    /* Success. */
    tf->tf_v0 = retval;
    tf->tf_a3 = 0; /* signal no error */
}

tf->tf_epc += 4;
}
System Call Interface

int open(const char *filename, int flags);
int open(const char *filename, int flags, mode_t mode);
int close(int fd);
ssize_t read(int fd, void *buf, size_t buflen);
ssize_t write(int fd, const void *buf, size_t nbytes);
int dup2(int oldfd, int newfd);
off_t lseek(int fd, off_t pos, int whence);
lseek() Offset

uint64_t offset;
int whence;
off_t retval64;

join32to64(tf->tf_a2, tf->tf_a3, &offset);

copyin((userptr_t)tf->tf_sp + 16, &whence, sizeof(int));

split64to32(retval64, &tf->tf_v0, &tf->tf_v1);
Pointers

- What about the first argument to `open()`
  - It’s a string?

- What are the problems with accessing a string (i.e. user-specified region of memory)?
Copy in/out(str)

int copyin(const_userptr_t usersrc, void *dest, size_t len);
int copyout(const void *src, userptr_t userdest, size_t len);
int copyinstr(const_userptr_t usersrc, char *dest, size_t len, size_t *got);
int copyoutstr(const char *src, userptr_t userdest, size_t len, size_t *got);
Buffers – e.g. read()

• Kernel framework for safely handling buffers
  • Does error/range/validity checking for you

ssize_t read(int fd, void *buf, size_t buflen);

struct iovec {
    union {
        userptr_t iov_ubase; /* user-supplied pointer */
        void   *iov_kbase; /* kernel-supplied pointer */
    };
    size_t iov_len; /* Length of data */
};
VFS READ

A macro with sanity checking

VOP_READ(vn, uio)

Invokes a function point of following prototype:
int (*vop_read)(struct vnode *file, struct uio *uio);

What are the arguments?
/* Source/destination. */
enum uio_seg {
    UIO_USERISPACE,  /* User process code. */
    UIO_USERSPACE,   /* User process data. */
    UIO_SYSSPACE,    /* Kernel. */
};

struct uio {
    struct iovec *uio_iov;     /* Data blocks */
    unsigned uio_iovcnt;       /* Number of iovecs */
    off_t uio_offset;          /* Desired offset into object */
    size_t uio_resid;          /* Remaining amt of data to xfer */
    enum uio_seg uio_segflg;   /* What kind of pointer we have */
    enum uio_rw uio_rw;        /* Whether op is a read or write */
    struct addrspace *uio_space;  /* Address space for user pointer */
};
Sample Helper function

```c
uiu_uinit(struct iovec *iov, struct uio *u, userptr_t buf,
size_t len, off_t offset, enum uio_rw rw)
{
    iov->iov_ubase = buf;
    iov->iov_len = len;
    u->uio iov = iov;
    u->uio iovcnt = 1;
    u->uio offset = offset;
    u->uio resid = len;
    u->uio segflg = UIO_USERSPACE;
    u->uio rw = rw;
    u->uio space = proc_getas();
}
```
System call implementation

1. sys_open()
2. sys_close()
3. sys_read()
4. sys_write()
5. sys_lseek()
6. sys_dup2()

1. vfs_open()
   • copyinstr()
2. vfs_close()
3. VOP_READ()
4. VOP_WRITE()
5. VOP_ISSEEKABLE()
6. VOP_STAT()