Assignment 2

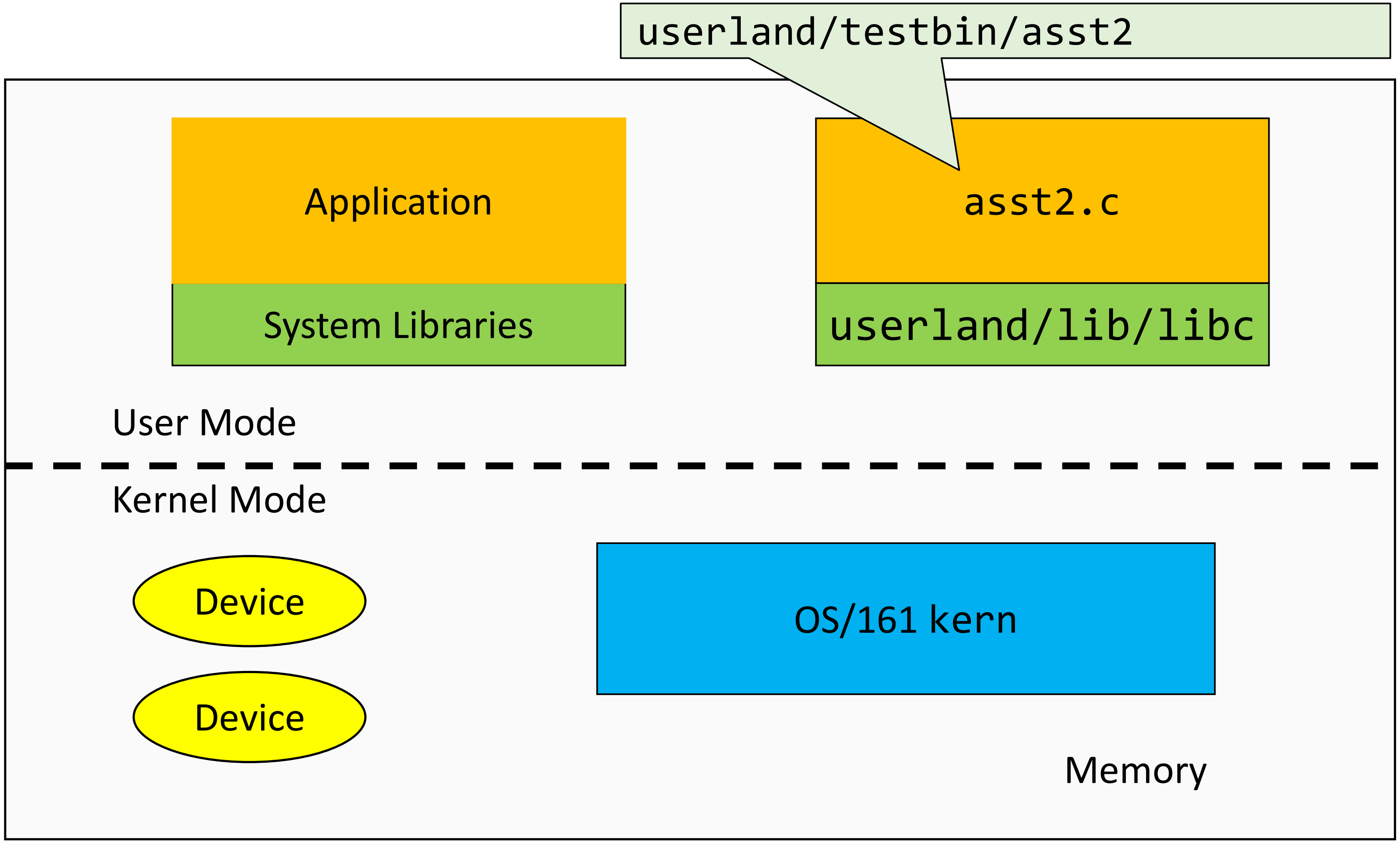
The Basic ASST2 Spec

- 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

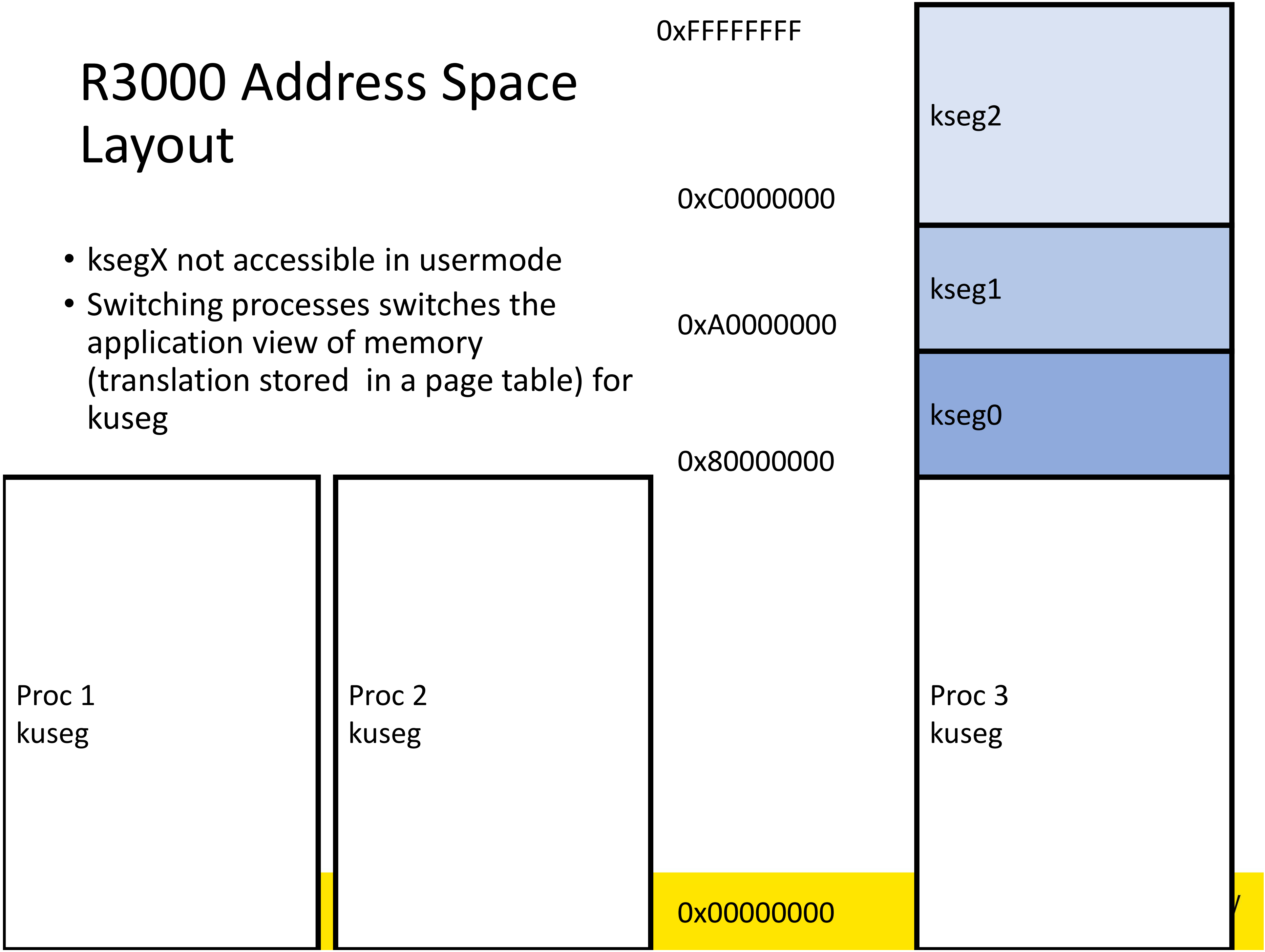
- Overall structure
  - User-level
    - Process structure
  - In-kernel
    - The storage stack
    - Overview of VFS and emufs functionality
- Details
  - Understanding the system interface
  - Argument passing
  - System call dispatching
  - Moving data across the user-kernel boundary
  - Connecting the interface to the VFS

Structure of a Computer System



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



0xFFFFFFFF

kseg2

0xC0000000

kseg1

0xA0000000

kseg0

0x80000000

Proc 1
kuseg

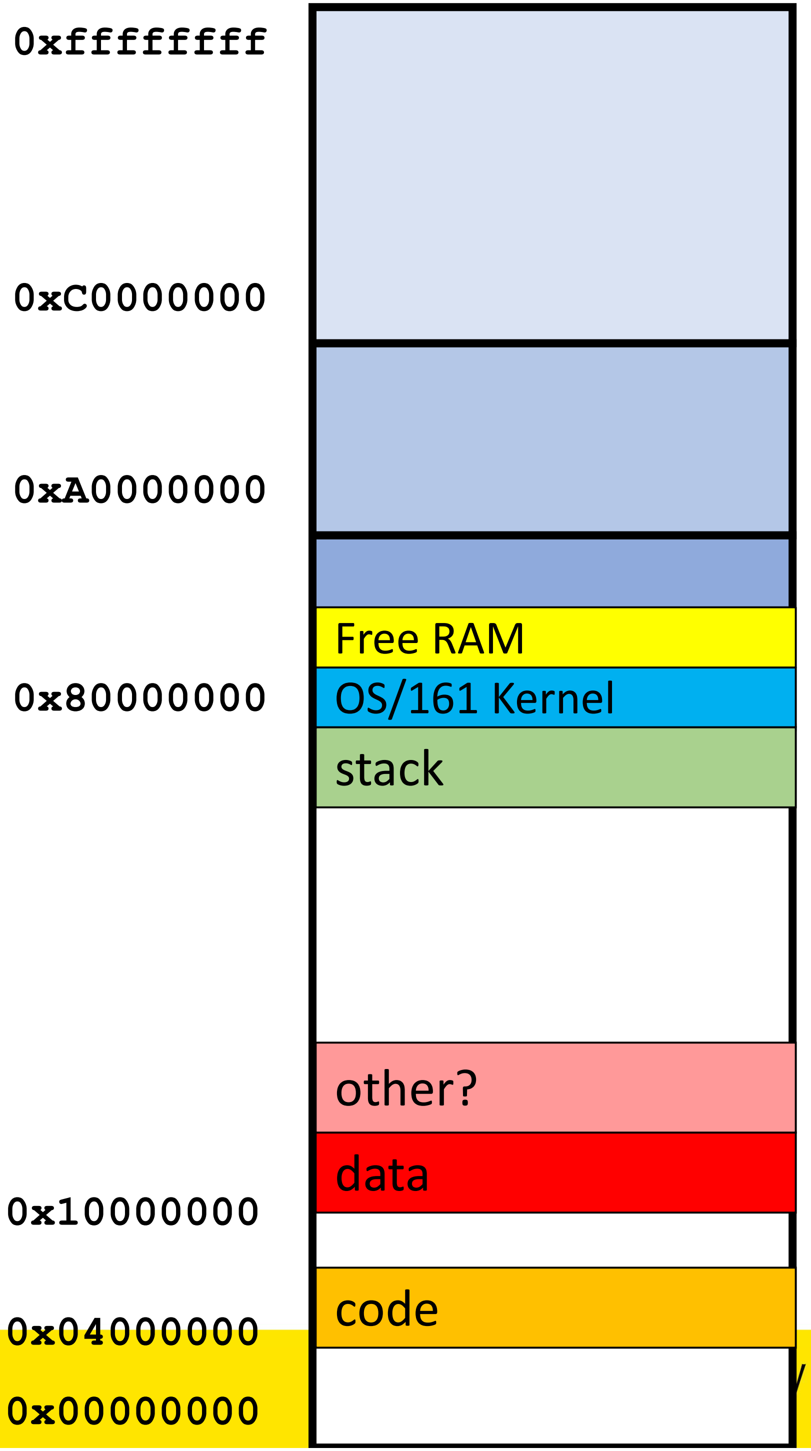
Proc 2
kuseg

Proc 3
kuseg

0x00000000

Process Layout

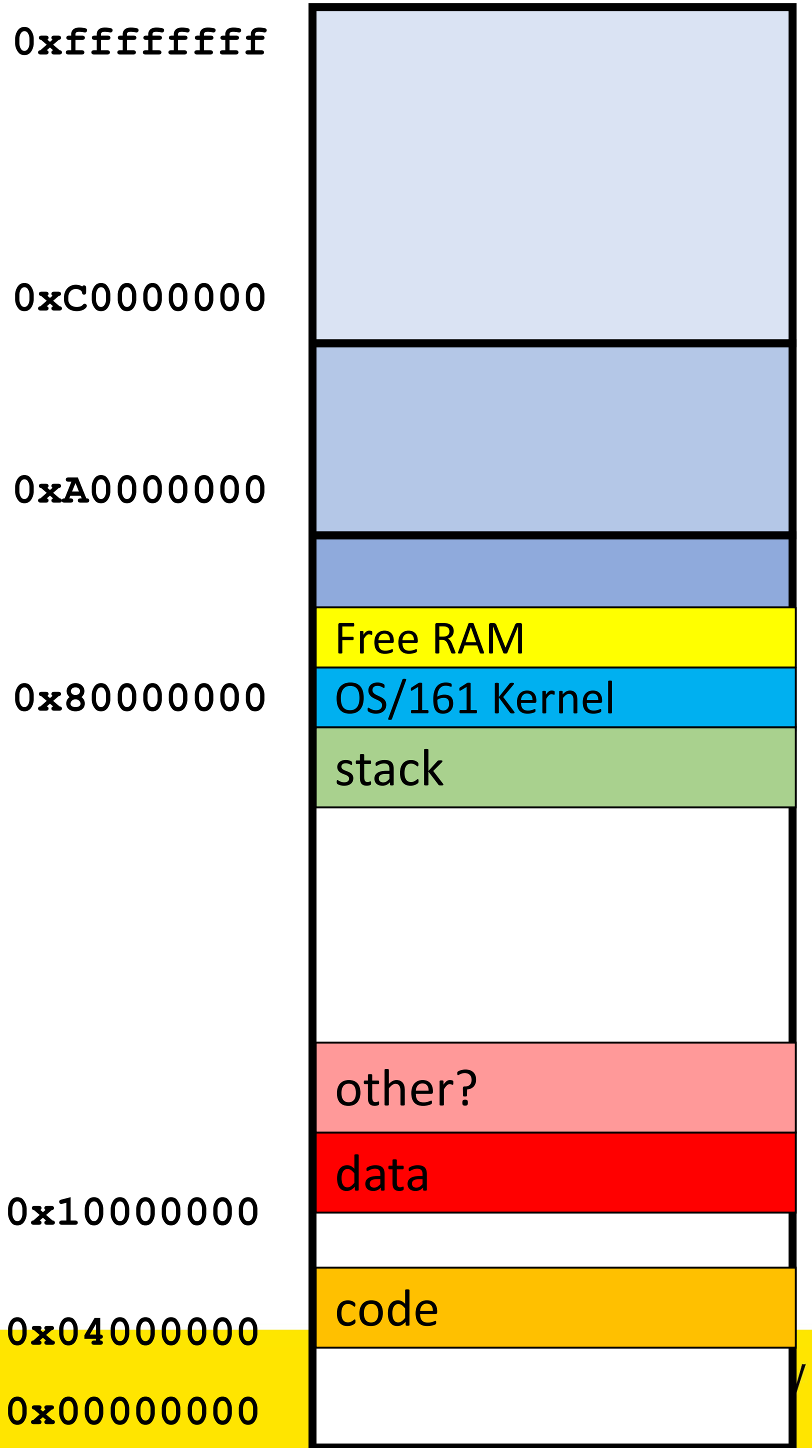
- Where is asst2 code/data (from `asst2.c`)?



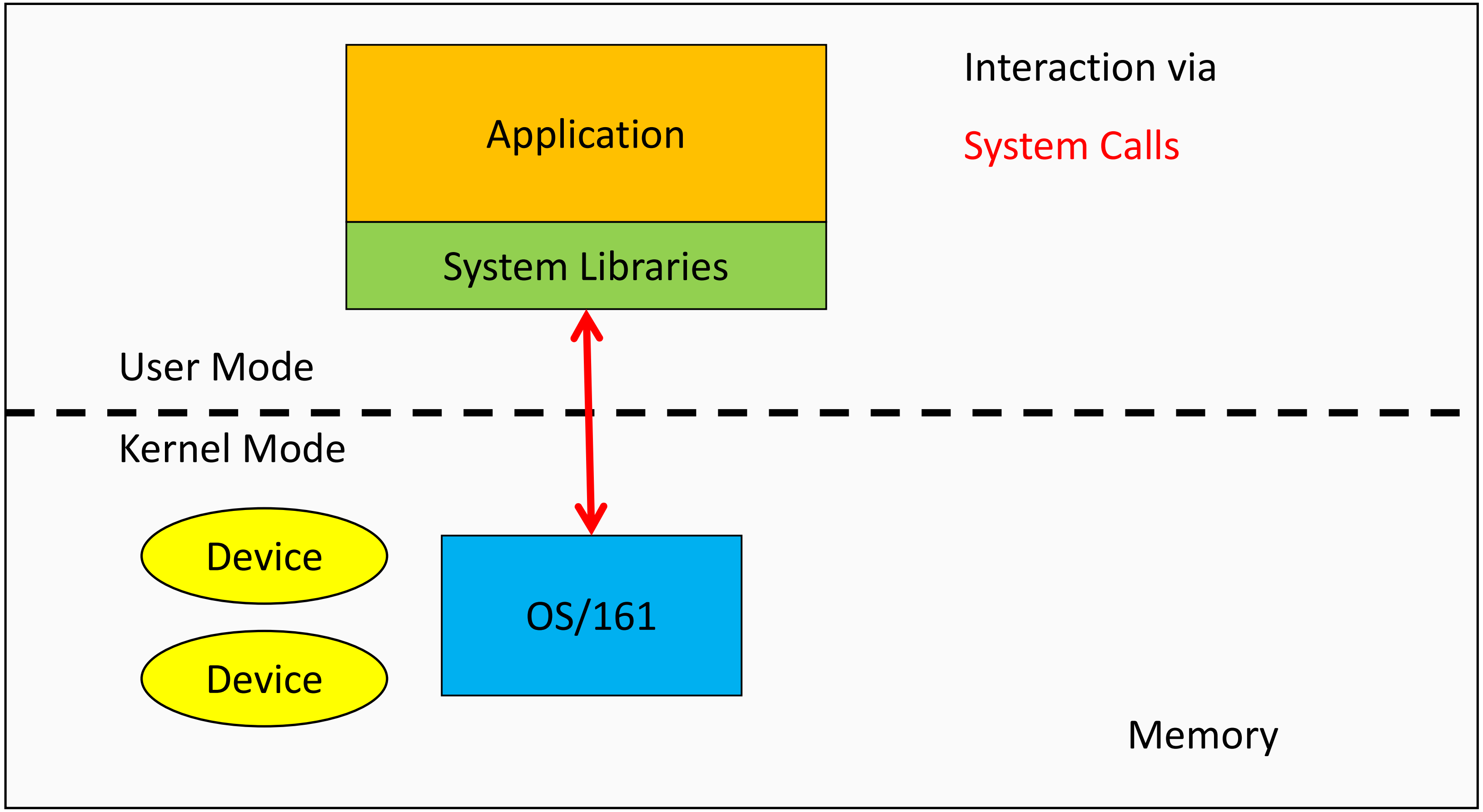
Calling open()

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

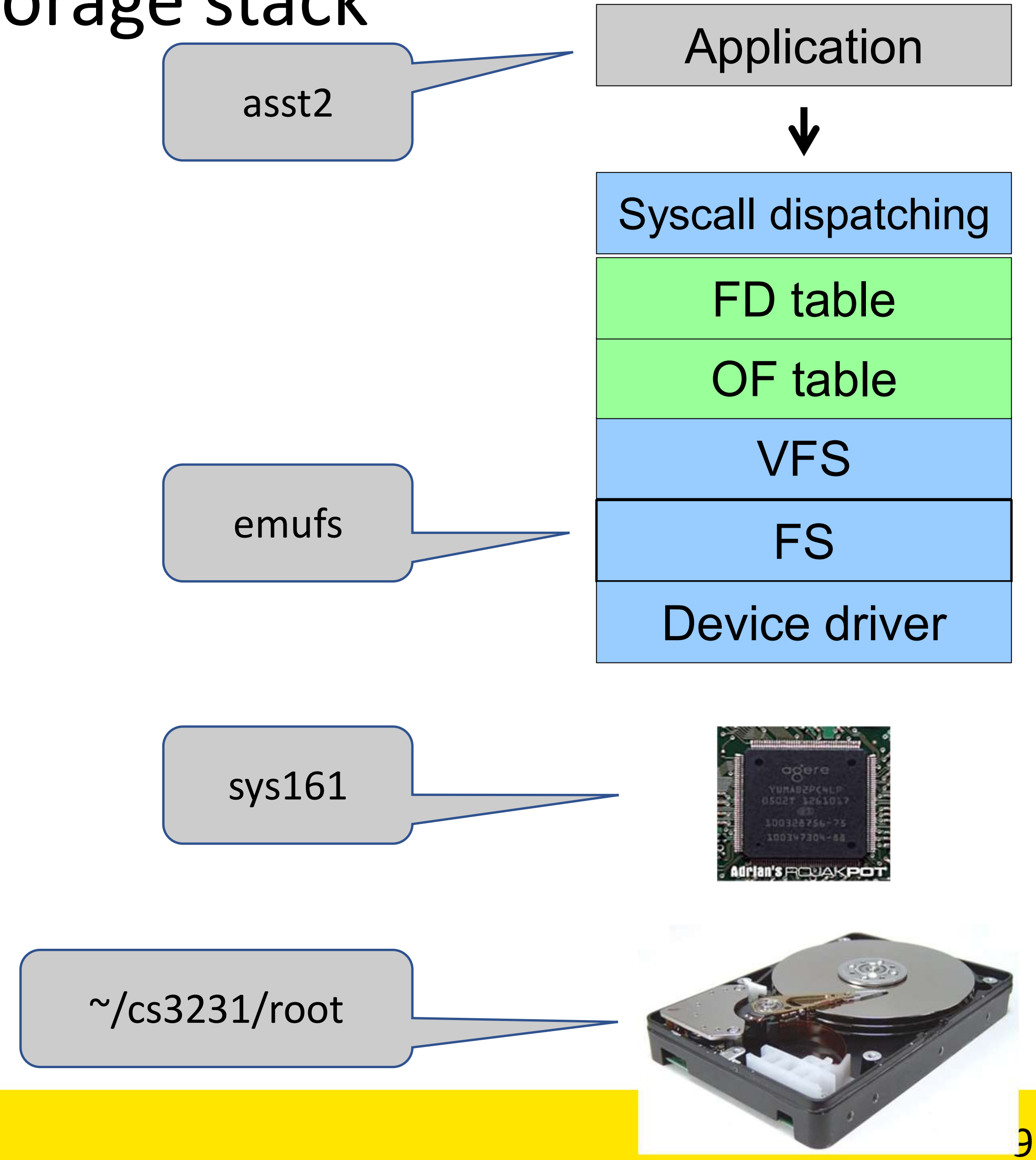
- Where is the function “open()”?



Structure of a Computer System



OS/161 storage stack



Application

Syscall dispatching

FD table

OF table

VFS

FS

Device driver

asst2

emufs

sys161

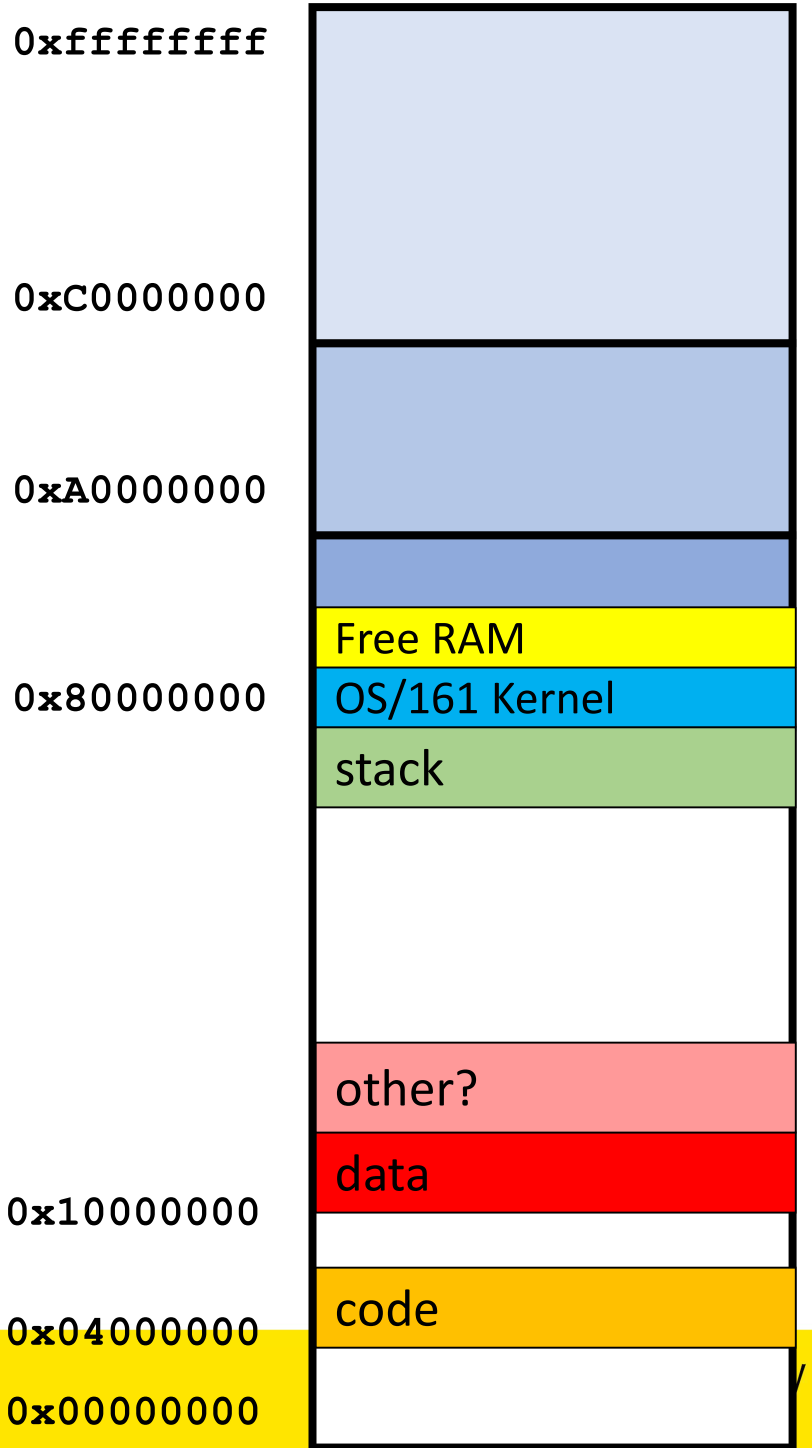
~/cs3231/root

open()?

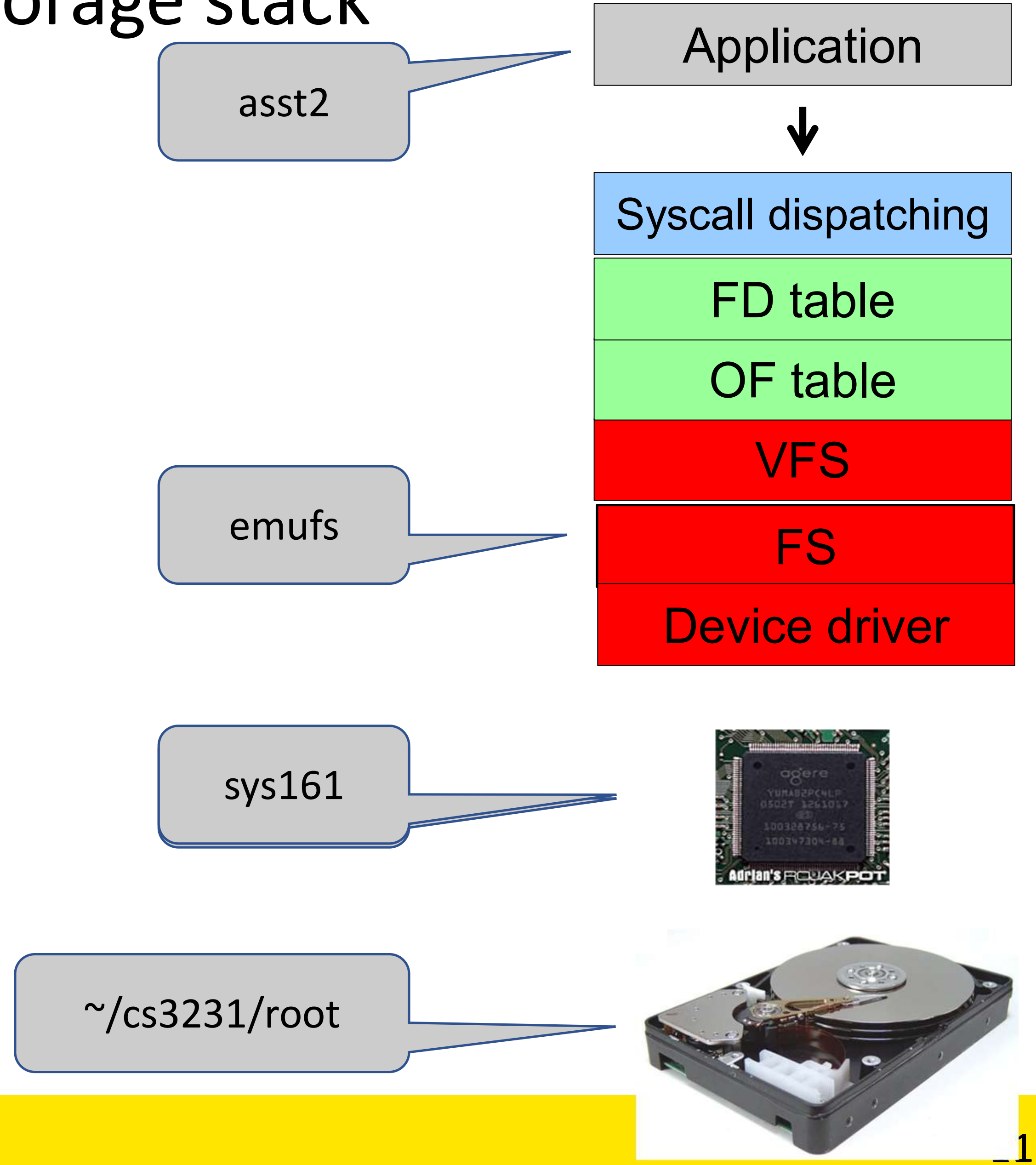
```
int open(const char *filename,
          int flags, ...);
```

- 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

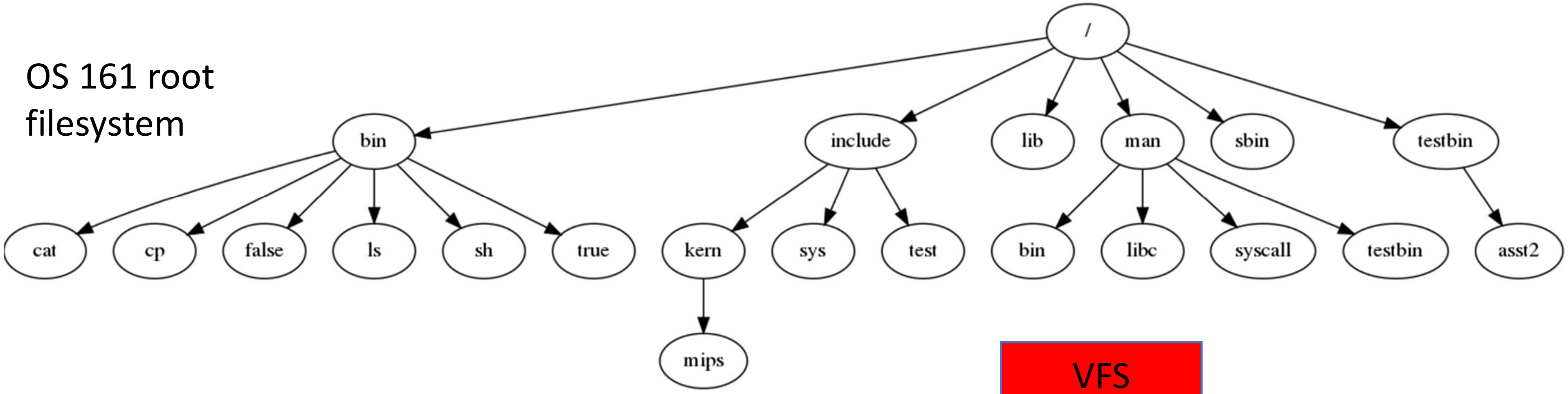
asst2

emufs

sys161

~/cs3231/root

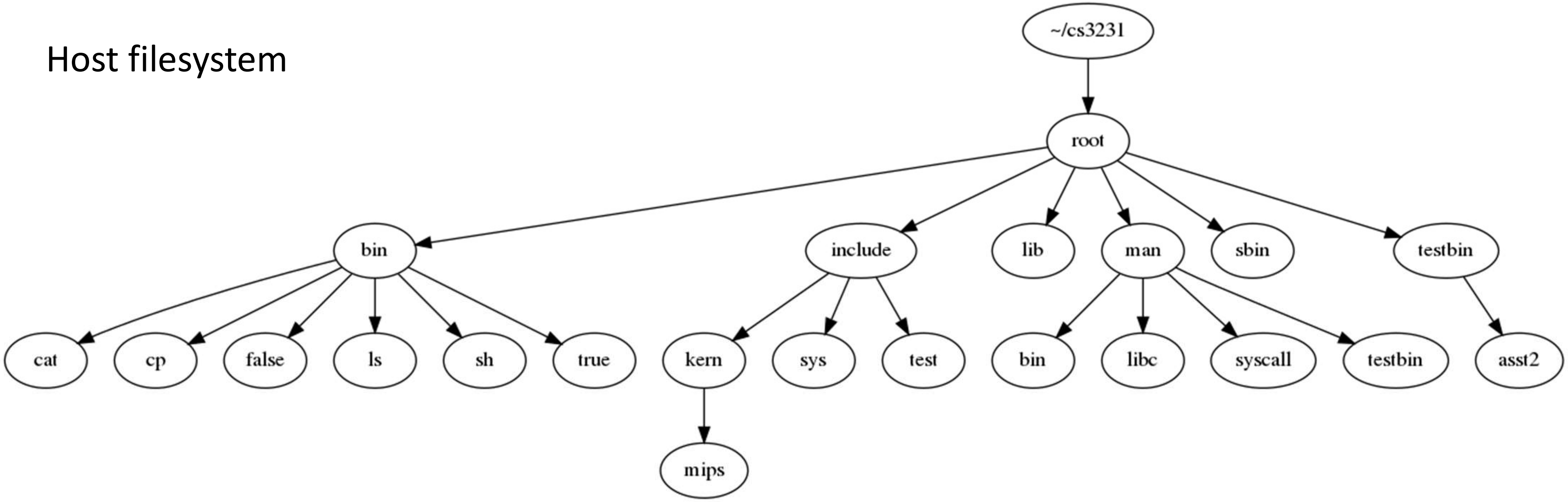
OS 161 root filesystem



Provided Storage Stack

VFS
emufs
driver
sys161

Host filesystem



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);
```

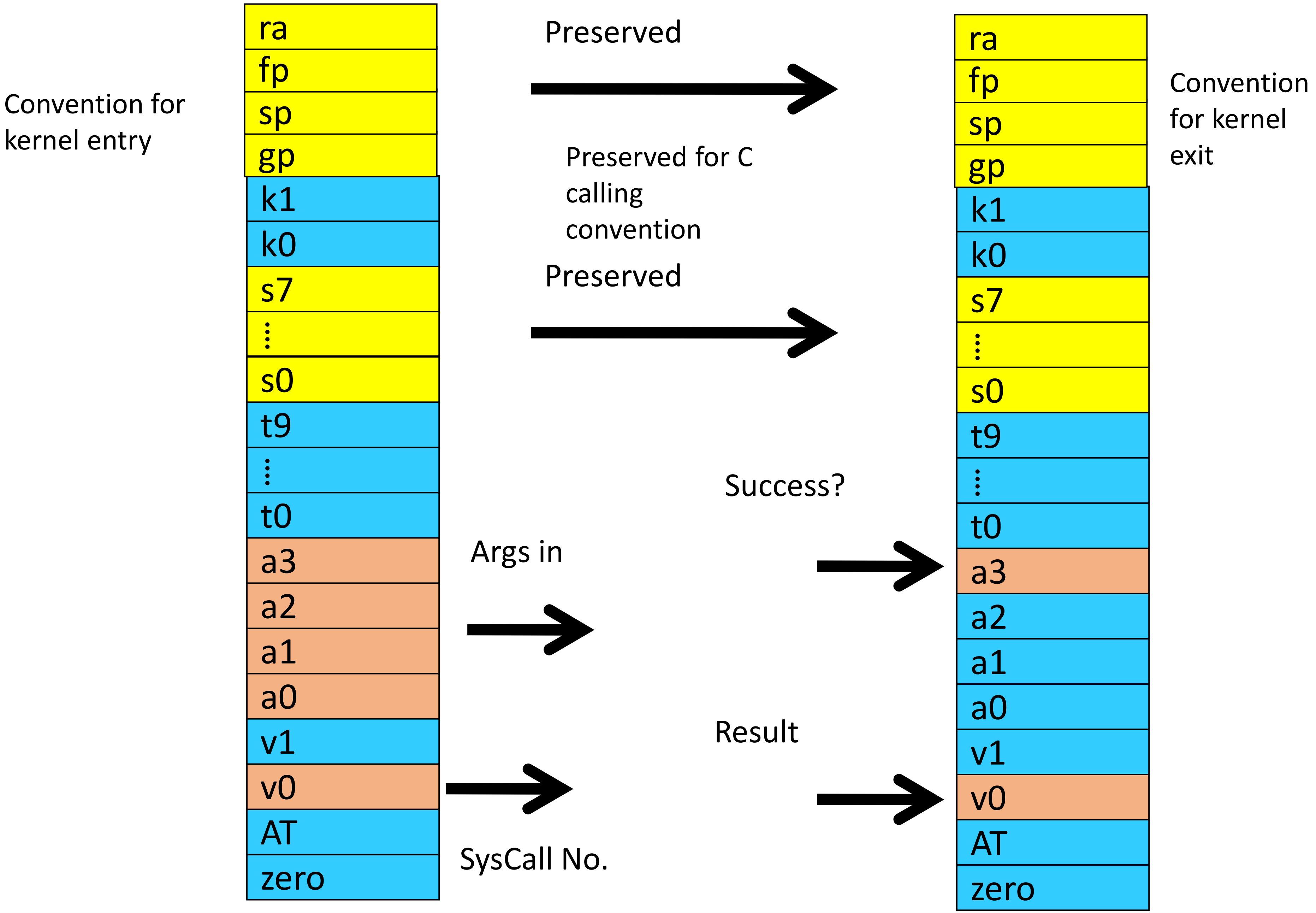
Description

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.



Convention for kernel entry

ra
fp
sp
gp
k1
k0
s7
⋮
s0
t9
⋮
t0
a3
a2
a1
a0
v1
v0
AT
zero

Preserved

Preserved for C calling convention

Preserved

Args in

SysCall No.

Success?

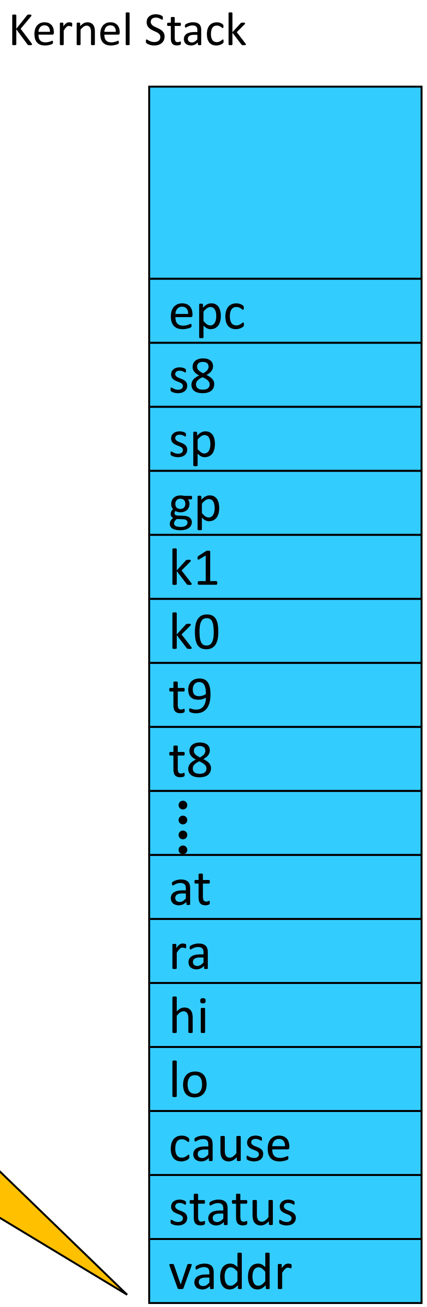
Result

ra
fp
sp
gp
k1
k0
s7
⋮
s0
t9
⋮
t0
a3
a2
a1
a0
v1
v0
AT
zero

Convention for kernel exit

```
struct trapframe {
  u_int32_t tf_vaddr;  /* vaddr register */
  u_int32_t tf_status;      /* status register */
  u_int32_t tf_cause;  /* cause register */
  u_int32_t tf_lo;
  u_int32_t tf_hi;
  u_int32_t tf_ra;     /* Saved register 31 */
  u_int32_t tf_at;     /* Saved register 1 (AT) */
  u_int32_t tf_v0;     /* Saved register 2 (v0) */
  u_int32_t tf_v1;     /* etc. */
  u_int32_t tf_a0;
  u_int32_t tf_a1;
  u_int32_t tf_a2;
  u_int32_t tf_a3;
  u_int32_t tf_t0;
     ⋮
  u_int32_t tf_t7;
  u_int32_t tf_s0;
     ⋮
  u_int32_t tf_s7;
  u_int32_t tf_t8;
  u_int32_t tf_t9;
  u_int32_t tf_k0;
  */
  u_int32_t tf_k1;
  u_int32_t tf_gp;
  u_int32_t tf_sp;
  u_int32_t tf_s8;
  u_int32_t tf_epc;    /* coprocessor 0 epc regis
};
```

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

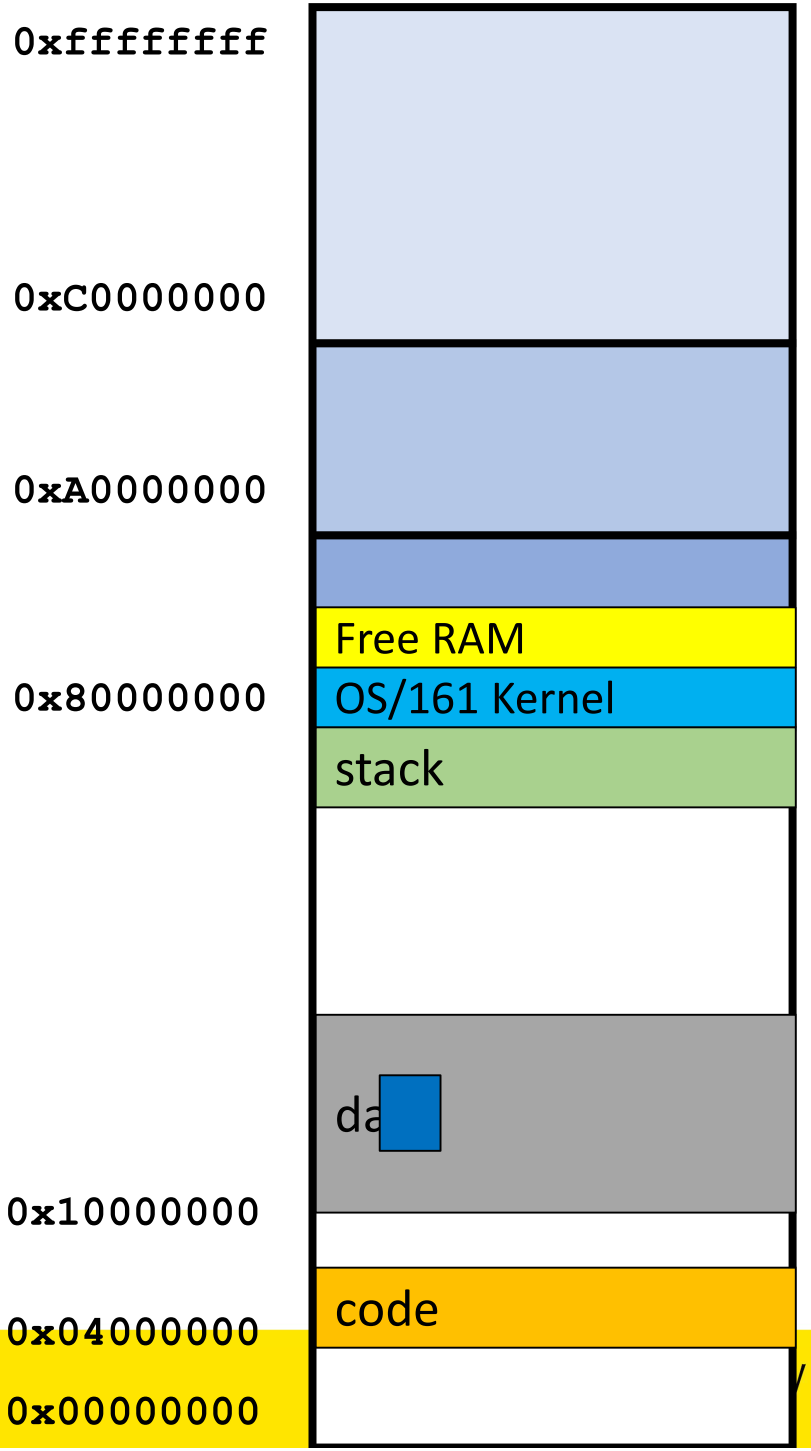
```
int open(const char *filename, int flags);
int open(const char *filename, int flags, mode_t mode);
int close(int fd);
ssize_t read(int fd, void *buf, size_t buflen);
ssize_t write(int fd, const void *buf, size_t nbytes);
int dup2(int oldfd, int newfd);
off_t lseek(int fd, off_t pos, int whence);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

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);
```

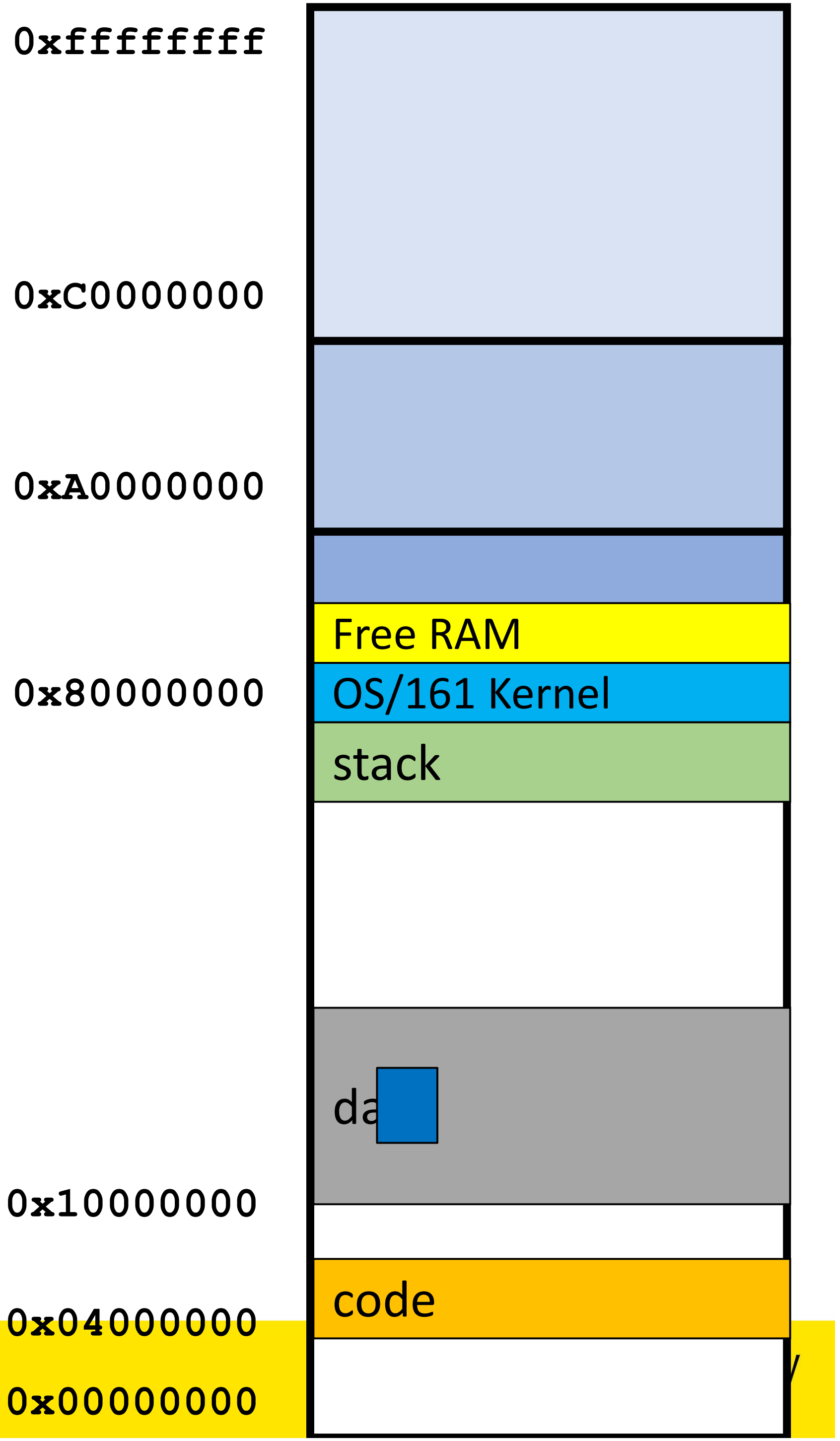
Pointers

- 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);
```



0xffffffff

0xC0000000

0xA0000000

Free RAM

OS/161 Kernel

0x80000000

stack

da

0x10000000

code

0x04000000

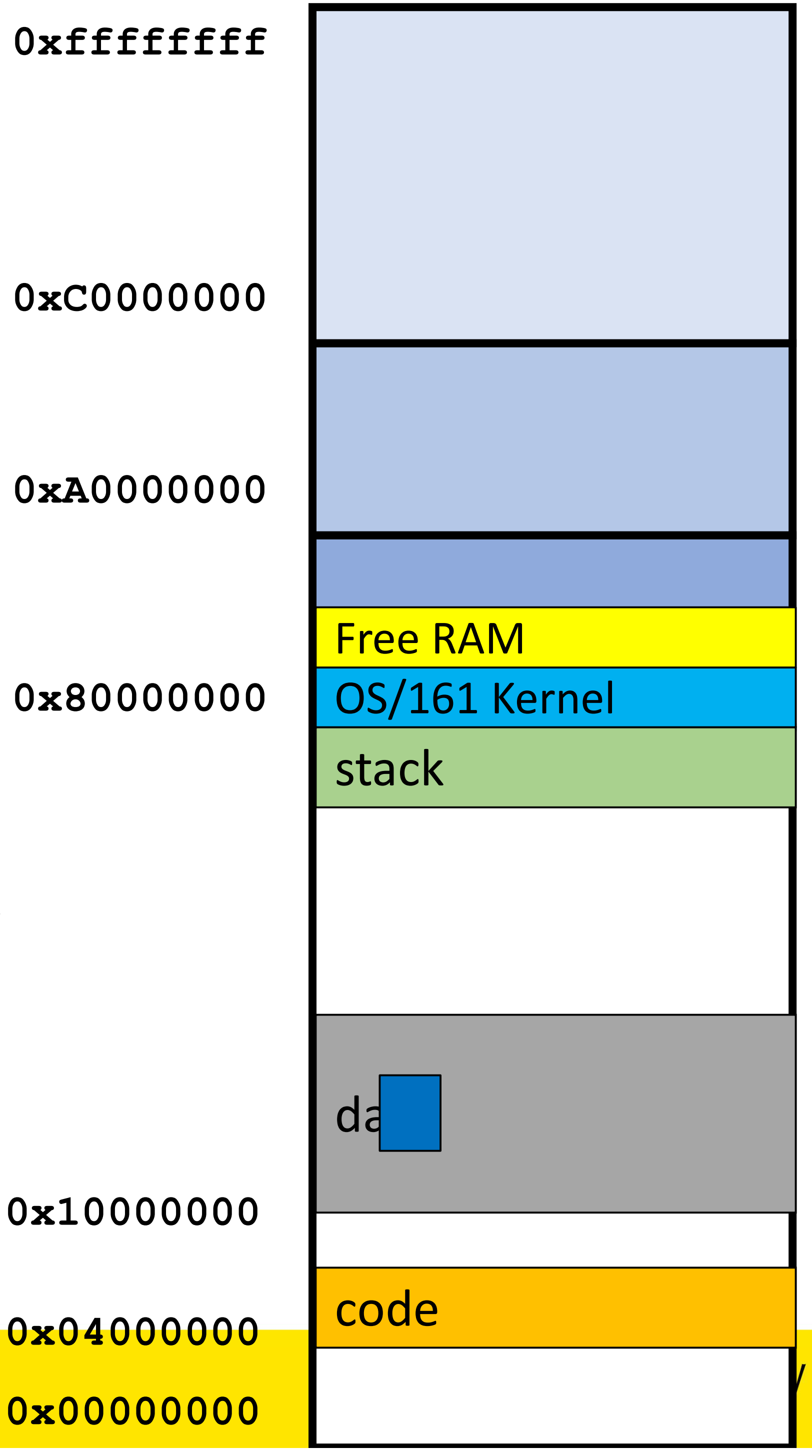
0x00000000

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?

UIO

```
/* 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

```
uio_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()