

# Assignment 2

1

## 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

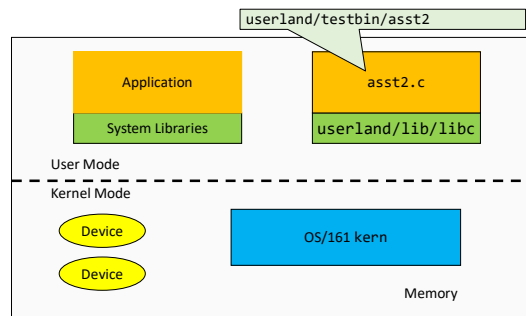
2

## 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

3

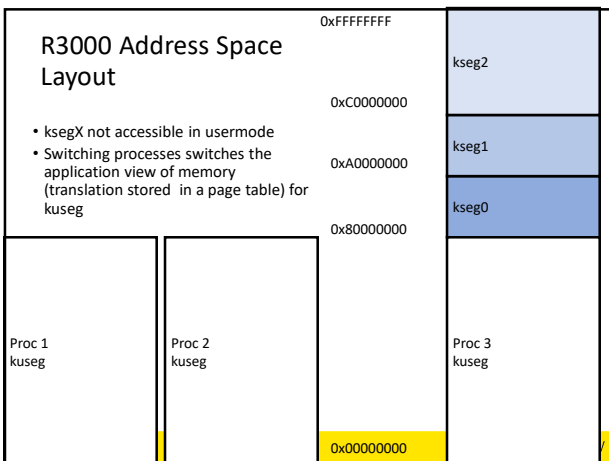
## Structure of a Computer System



4

## R3000 Address Space Layout

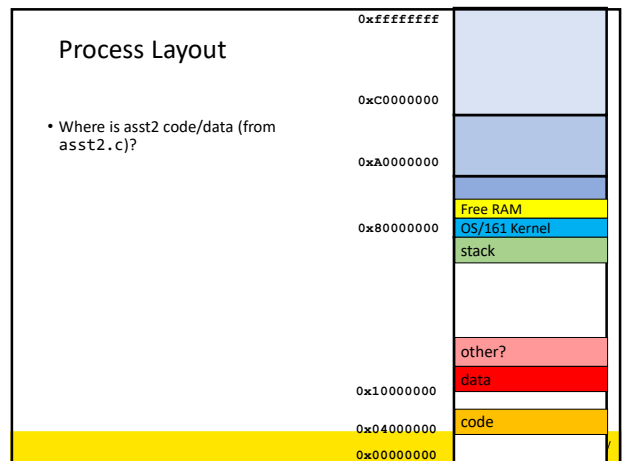
- `ksegX` not accessible in usermode
- Switching processes switches the application view of memory (translation stored in a page table) for `kuseg`



5

## Process Layout

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



6

### Calling open()

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

- Where is the function "open()?"

7

### Structure of a Computer System

8

### OS/161 storage stack

9

### 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

10

### Existing storage stack

11

### Provided Storage Stack

12

## Details

13 

13

## 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);

14 

14

## open/close

```
int open(const char *filename, int flags);
int open(const char *filename, int flags, mode_t mode);
int close(int fd);
```

15 

15

## Read/write

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

16 

16

## dup2

```
int dup2(int oldfd, int newfd);
```

17 

17

## lseek

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

18 

18

fork

```
pid_t fork(void);
```

### Argument passing

```
#include <unistd.h>
```

```
int reboot(int code);
```

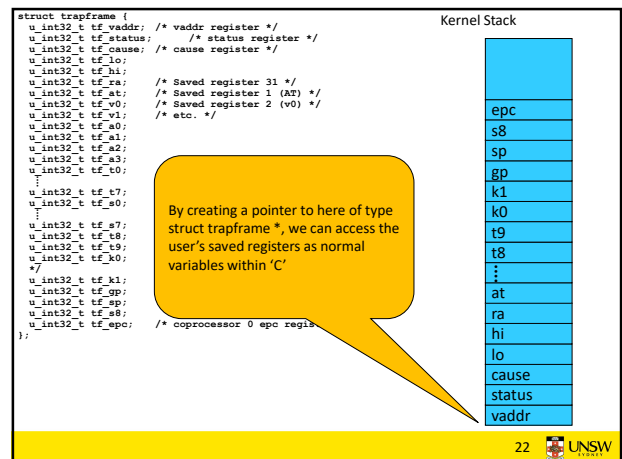
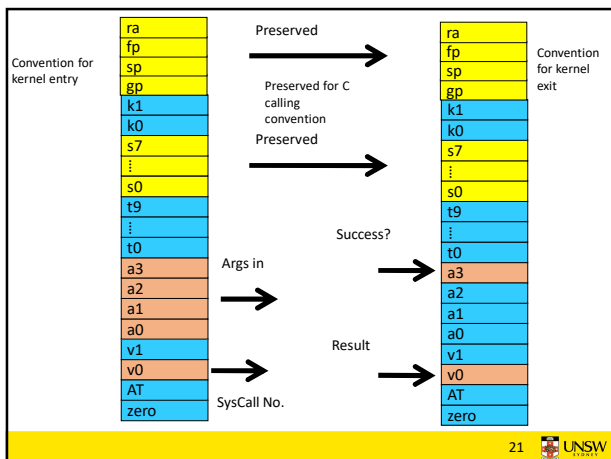
#### Description

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.



```
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);
```

25 UNSW

25

## 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);
```

26 UNSW

26

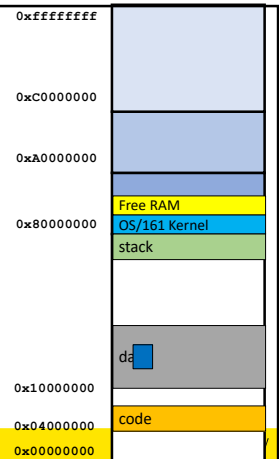


27 UNSW

27

## Pointers

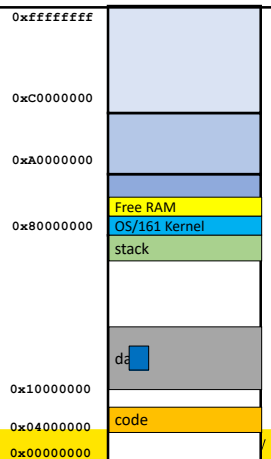
- What about the first argument to open()
  - It's a string?
- What are the problems with accessing a string (i.e. user-specified region of memory)?



28

## 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);
```



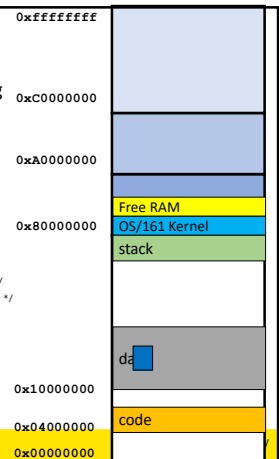
29

## 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_base; /* user-supplied pointer */
        void *iov_kbase; /* kernel-supplied pointer */
    };
    size_t iov_len; /* Length of data */
};
```



30

## 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?

31 UNSW

31

## UIO

```
/* Source/destination. */
enum uio_seg {
    UIO_USERSPACE, /* 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 */
};
```

32 UNSW

32

## 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->uio_ubase = buf;
    iov->uio_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();
}
```

33 UNSW

33

## System call implementation

- |                |                     |
|----------------|---------------------|
| 1. sys_open()  | 1. vfs_open()       |
| 2. sys_close() | • copyinstr()       |
| 3. sys_read()  | 2. vfs_close()      |
| 4. sys_write() | 3. VOP_READ()       |
| 5. sys_lseek() | 4. VOP_WRITE()      |
| 6. sys_dup2()  | 5. VOP_ISSEEKABLE() |
|                | 6. VOP_STAT()       |

34 UNSW

34