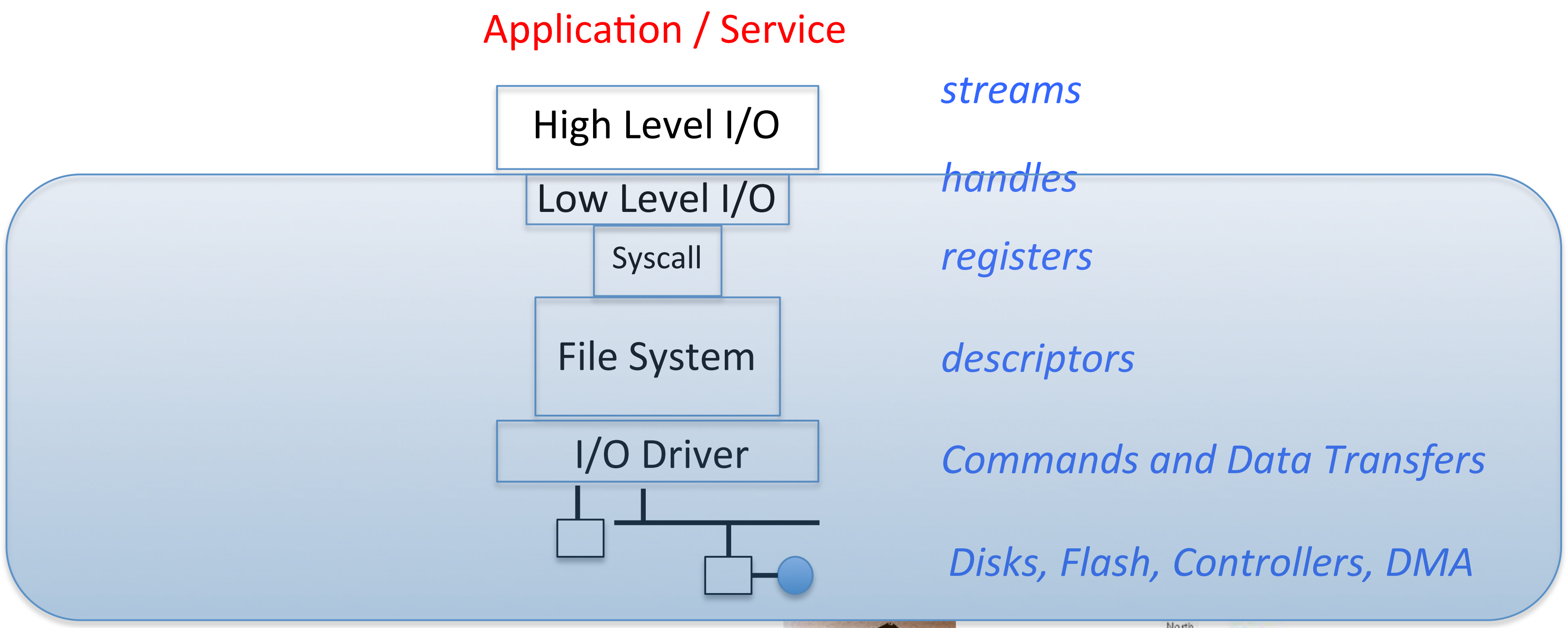
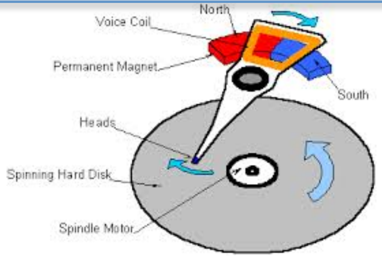
Introduction to File Systems - beneath the surface

David E. Culler
CS162 – Operating Systems and Systems Programming
Lecture 4
Sept 8, 2014

Reading: A&D 3.1-3, 11.1-2
HW0 due today
HW1: out

What's below the surface ??





File Intro recall exercise

- What is the namespace introduced by the file system?
- Like an address space, but structured names, rather than flat addresses

C Low level I/O

- Operations on File Descriptors – as OS object representing the state of a file
  - User has a "handle" on the descriptor

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>

int open (const char *filename, int flags [, mode_t mode])
int close (int filedes)
```

Bit vector of:
- Access modes (Rd, Wr, …)
- Open Flags (Create, …)
- Operating modes (Appends, …)

Bit vector of Permission Bits:
- User|Group|Other X R|W|X

http://www.gnu.org/software/libc/manual/html_node/Opening-and-Closing-Files.html

C Low Level: standard descriptors

```
#include <unistd.h>

STDIN_FILENO -  macro has value 0
STDOUT_FILENO - macro has value 1
STDERR_FILENO - macro has value 2

int fileno (FILE *stream)

FILE * fdopen (int filedes, const char *opentype)
```

- Crossing levels: File descriptors vs. streams
- Don’t mix them!

C Low Level Operations

```
#include <unistd.h>
#include <sys/types.h>

ssize_t read (int filedes, void *buffer, size_t maxsize)
```
- returns bytes read, 0 => EOF, -1 => error
```
ssize_t write (int filedes, const void *buffer, size_t size)
 - returns bytes written

off_t lseek (int filedes, off_t offset, int whence)

int fsync (int fildes)          – wait for i/o to finish
void sync (void)                – wait for ALL to finish
```

- When write returns, data is on its way to disk and can be read, but it may not actually be permanent!
- ISO C: size_t is the preferred way to declare any arguments or variables that hold the size of an object.
- ssize_t return value permits use of -1 to indicate error

A little example: lowio.c

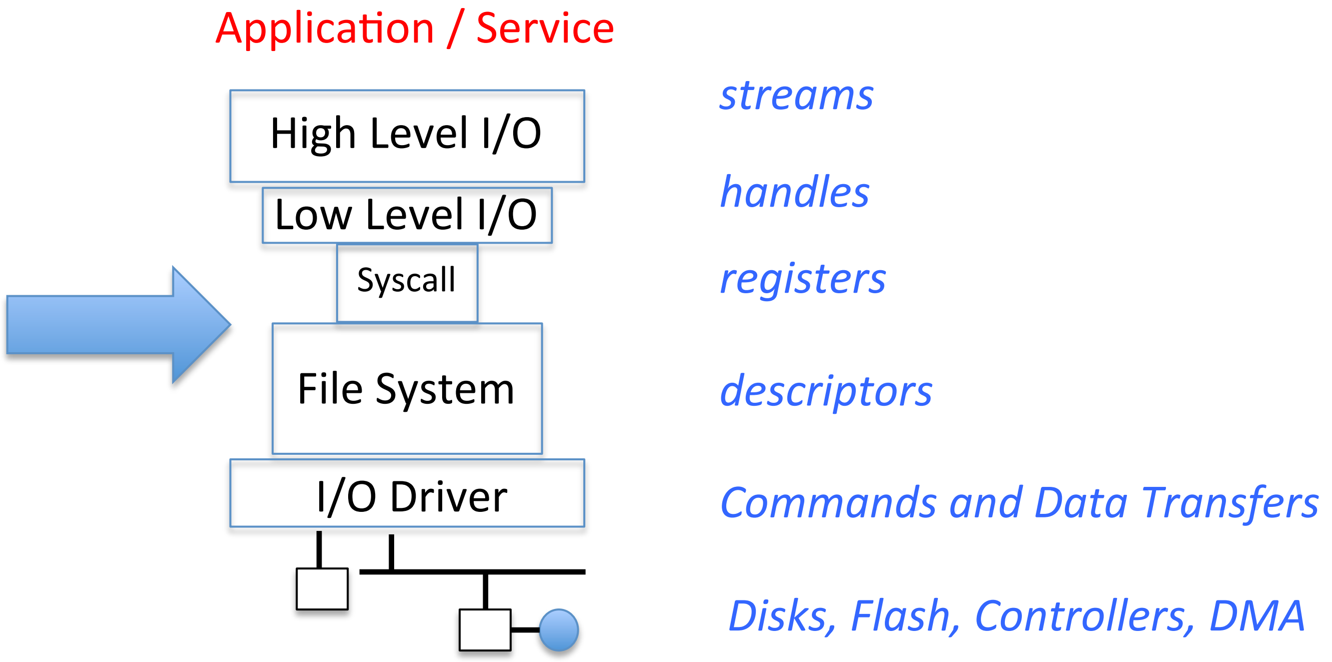
```
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
  char buf[1000];
  int     fd = open("lowio.c", O_RDONLY, S_IRUSR | S_IWUSR);
  ssize_t rd = read(fd, buf, sizeof(buf));
  int    err = close(fd);
  ssize_t wr = write(STDOUT_FILENO, buf, rd);
}
```

And lots more !

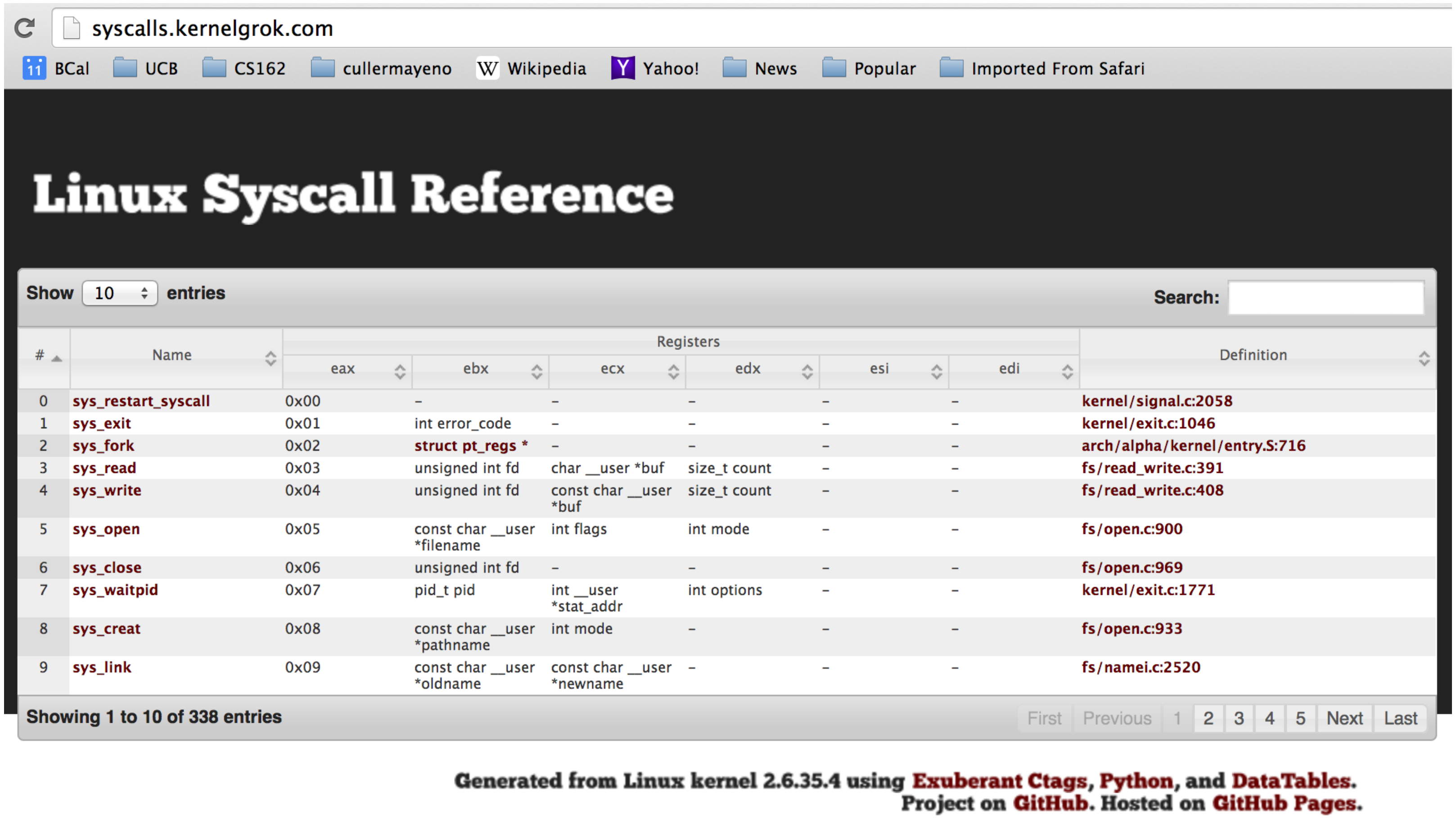
- TTYs versus files
- Memory mapped files
- File Locking
- Asynchronous I/O
- Generic I/O Control Operations
- Duplicating descriptors

```
int dup2 (int old, int new)
int dup (int old)
```

What's below the surface ??



SYSCALL



Linux Syscall Reference

Show 10 entries Search:

#	Name	Registers						Definition
		eax	ebx	ecx	edx	esi	edi	
0	sys_restart_syscall	0x00	-	-	-	-	-	kernel/signal.c:2058
1	sys_exit	0x01	int error_code	-	-	-	-	kernel/exit.c:1046
2	sys_fork	0x02	struct pt_regs *	-	-	-	-	arch/alpha/kernel/entry.S:716
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count	-	-	fs/read_write.c:391
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count	-	-	fs/read_write.c:408
5	sys_open	0x05	const char __user *filename	int flags	int mode	-	-	fs/open.c:900
6	sys_close	0x06	unsigned int fd	-	-	-	-	fs/open.c:969
7	sys_waitpid	0x07	pid_t pid	int __user *stat_addr	int options	-	-	kernel/exit.c:1771
8	sys_creat	0x08	const char __user *pathname	int mode	-	-	-	fs/open.c:933
9	sys_link	0x09	const char __user *oldname	const char __user *newname	-	-	-	fs/namei.c:2520

Showing 1 to 10 of 338 entries

First Previous 1 2 3 4 5 Next Last

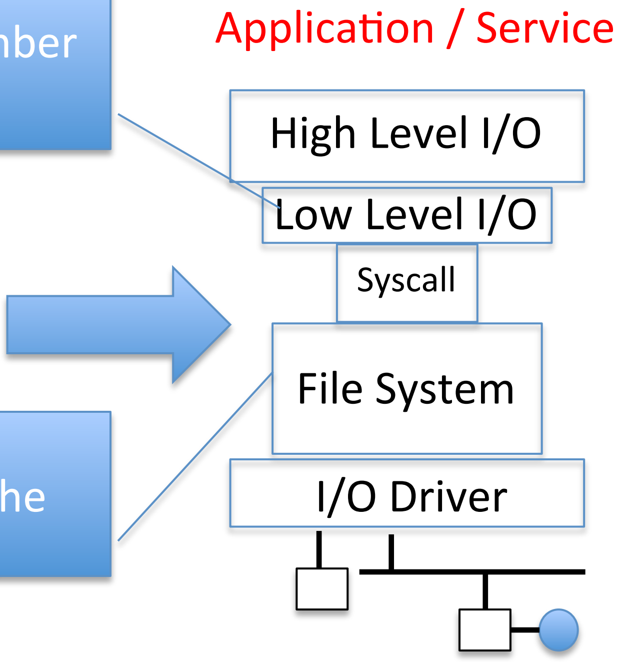
Generated from Linux kernel 2.6.35.4 using Exuberant Ctags, Python, and DataTables.
Project on GitHub. Hosted on GitHub Pages.

- Low level lib parameters are set up in registers and syscall instruction is issued

What's below the surface ??

File descriptor number
- an int

File Descriptors
- a struct with all the info about the files



streams

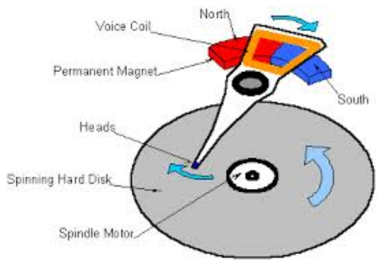
handles

registers

descriptors

Commands and Data Transfers

Disks, Flash, Controllers, DMA

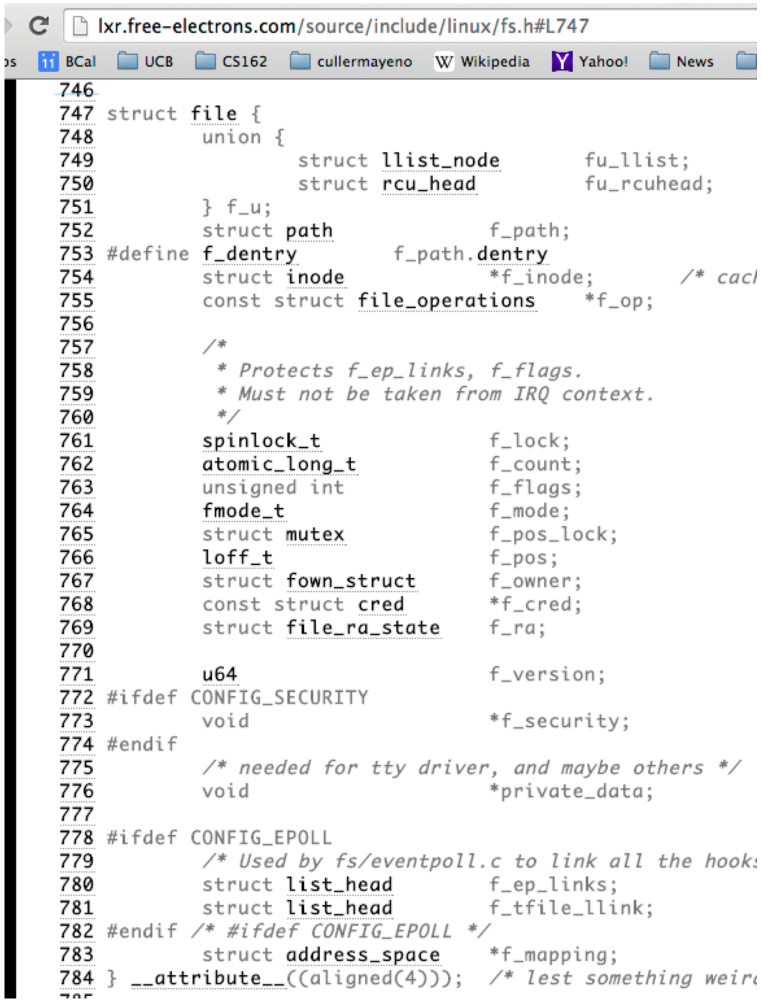


Another: lowio-std.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define BUFSIZE 1024

int main(int argc, char *argv[])
{
  char buf[BUFSIZE];
  ssize_t writelen = write(STDOUT_FILENO, "I am a process.\n", 16);

  ssize_t readlen  = read(STDIN_FILENO, buf, BUFSIZE);

  ssize_t strlen   = snprintf(buf, BUFSIZE,"Got %zd chars\n", readlen);

  writelen = strlen < BUFSIZE ? strlen : BUFSIZE;
  write(STDOUT_FILENO, buf, writelen);

  exit(0);
}
```

Internal OS File Descriptor

- Internal Data Structure describing everything about the file
  - Where it resides
  - Its status
  - How to access it



```
struct file {
        union {
                struct llist_node       fu_llist;
                struct rcu_head         fu_rcuhead;
        } f_u;
        struct path             f_path;
#define f_dentry        f_path.dentry
        struct inode            *f_inode;       /* cac
        const struct file_operations    *f_op;

        /*
         * Protects f_ep_links, f_flags.
         * Must not be taken from IRQ context.
         */
        spinlock_t              f_lock;
        atomic_long_t           f_count;
        unsigned int            f_flags;
        fmode_t                 f_mode;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;

        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        /* needed for tty driver, and maybe others */
        void                    *private_data;

#ifdef CONFIG_EPOLL
        /* Used by fs/eventpoll.c to link all the hook
        struct list_head        f_ep_links;
        struct list_head        f_tfile_llink;
#endif /* #ifdef CONFIG_EPOLL */
        struct address_space    *f_mapping;
} __attribute__((aligned(4)));  /* lest something weir
```

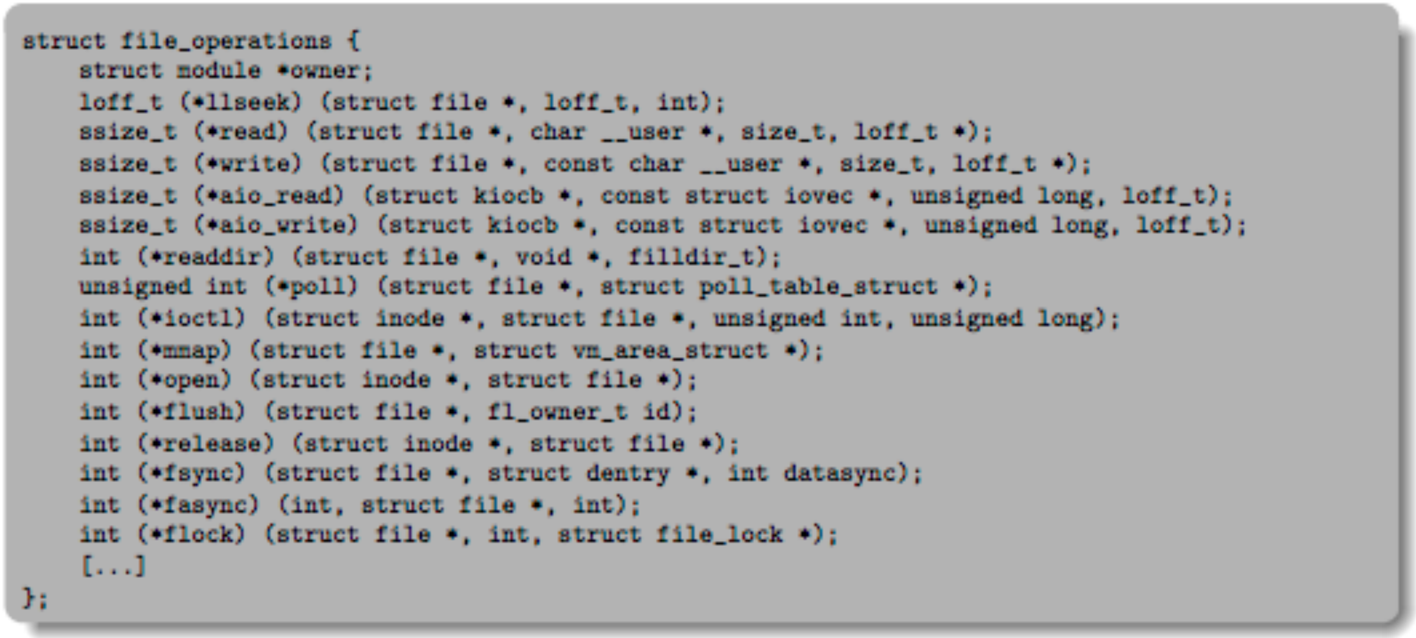
File System: from syscall to driver

In fs/read_write.c

```
ssize_t vfs_read(struct file *file, char __user *buf, size_t count, loff_t *pos)
{
  ssize_t ret;
  if (!(file->f_mode & FMODE_READ)) return -EBADF;
  if (!file->f_op || (!file->f_op->read && !file->f_op->aio_read))
    return -EINVAL;
  if (unlikely(!access_ok(VERIFY_WRITE, buf, count))) return -EFAULT;
  ret = rw_verify_area(READ, file, pos, count);
  if (ret >= 0) {
    count = ret;
    if (file->f_op->read)
      ret = file->f_op->read(file, buf, count, pos);
    else
      ret = do_sync_read(file, buf, count, pos);
    if (ret > 0) {
      fsnotify_access(file->f_path.dentry);
      add_rchar(current, ret);
    }
    inc_syscr(current);
  }
  return ret;
}
```

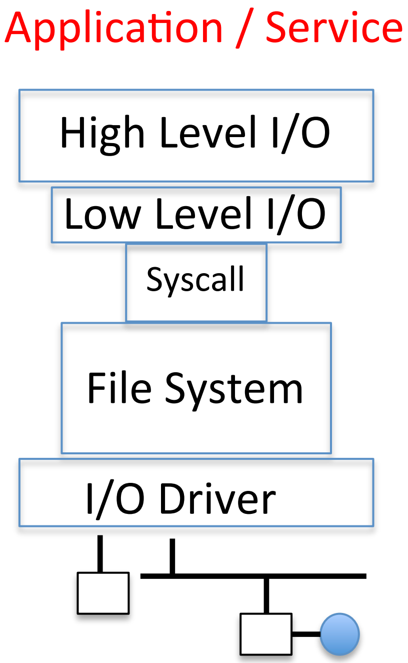
Low Level Driver

- Associated with particular hardware device
- Registers / Unregisters itself with the kernel
- Handler functions for each of the file operations



```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, struct dentry *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*flock) (struct file *, int, struct file_lock *);
    [...]
};
```

So what happens when you fgetc?



streams

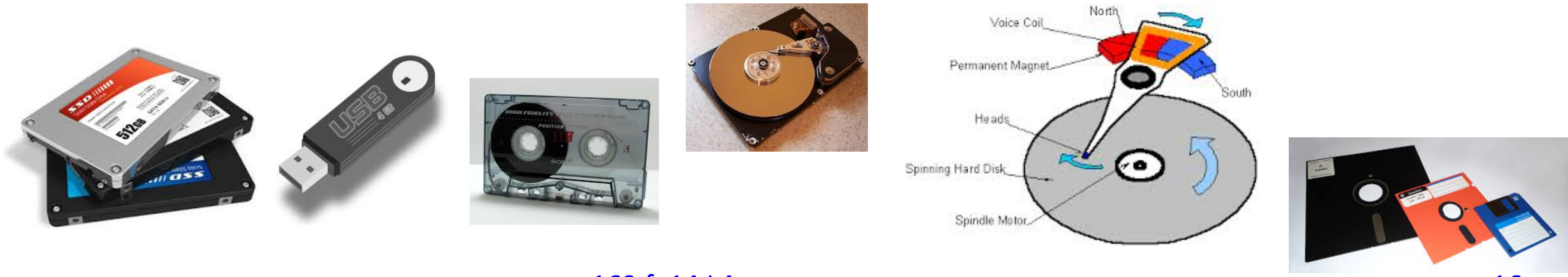
handles

registers

descriptors

Commands and Data Transfers

Disks, Flash, Controllers, DMA



Breather

Question

- Process is an instance of a program executing.
  - – The fundamental OS responsibility
- Processes do their work by processing and calling file system operations
- Are their any operations on processes themselves?
- exit ?

pid.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define BUFSIZE 1024
int main(int argc, char *argv[])
{
  int c;

  pid_t pid = getpid();   /* get current processes PID */

  printf("My pid: %d\n", pid);

  c = fgetc(stdin);
  exit(0);
}
```

ps anyone?

Can a process create a process ?

- Yes
- Fork creates a copy of process

fork1.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define BUFSIZE 1024
int main(int argc, char *argv[])
{
  char buf[BUFSIZE];
  size_t readlen, writelen, slen;
  pid_t cpid, mypid;
  pid_t pid = getpid();          /* get current processes PID */
  printf("Parent pid: %d\n", pid);
  cpid = fork();
  if (cpid > 0) {                /* Parent Process */
    mypid = getpid();
    printf("[%d] parent of [%d]\n", mypid, cpid);
  }  else if (cpid == 0) {      /* Child Process */
    mypid = getpid();
    printf("[%d] child\n", mypid);
  } else {
    perror("Fork failed");
    exit(1);
  }
  exit(0);
}
```

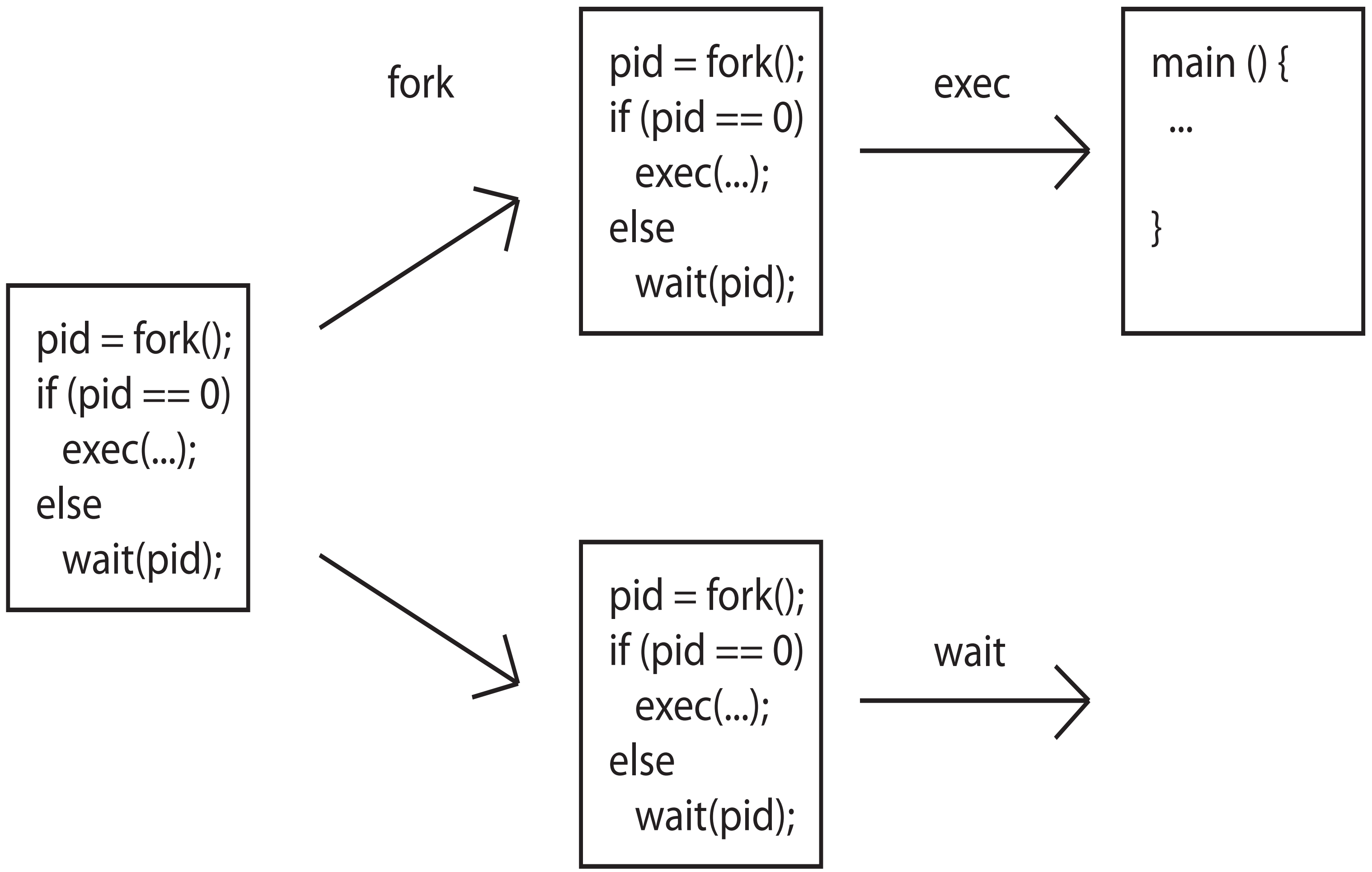
UNIX Process Management

- UNIX fork – system call to create a copy of the current process, and start it running
  - No arguments!
- UNIX exec – system call to *change the program* being run by the current process
- UNIX wait – system call to wait for a process to finish
- UNIX signal – system call to send a notification to another process

fork2.c

```
…
cpid = fork();
if (cpid > 0) {                    /* Parent Process */
  mypid = getpid();
  printf("[%d] parent of [%d]\n", mypid, cpid);
  tcpid = wait(&status);
  printf("[%d] bye %d\n", mypid, tcpid);
}  else if (cpid == 0) {           /* Child Process */
  mypid = getpid();
  printf("[%d] child\n", mypid);
}
…
```

UNIX Process Management



Shell

- A shell is a job control system
  – Allows programmer to create and manage a set of programs to do some task
  – Windows, MacOS, Linux all have shells

HW1

- Example: to compile a C program

```
cc –c sourcefile1.c
cc –c sourcefile2.c
ln –o program sourcefile1.o sourcefile2.o
./program
```

Signals – infloop.c

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>

#include <unistd.h>
#include <signal.h>

void signal_callback_handler(int signum)
{
  printf("Caught signal %d - phew!\n",signum);
  exit(1);
}

int main() {
  signal(SIGINT, signal_callback_handler);

  while (1) {}
}
```

Got top?

Process races: fork.c

```
if (cpid > 0) {
   mypid = getpid();
   printf("[%d] parent of [%d]\n", mypid, cpid);
   for (i=0; i<100; i++) {
     printf("[%d] parent: %d\n", mypid, i);
     //       sleep(1);
   }
 }  else if (cpid == 0) {
   mypid = getpid();
   printf("[%d] child\n", mypid);
   for (i=0; i>-100; i--) {
     printf("[%d] child: %d\n", mypid, i);
     //       sleep(1);
   }
 }
```

BIG OS Concepts so far

- Processes
- Address Space
- Protection
- Dual Mode
- Interrupt handlers (including syscall and trap)
- File System
  – Integrates processes, users, cwd, protection
- Key Layers: OS Lib, Syscall, Subsystem, Driver
  – User handler on OS descriptors
- Process control
  – fork, wait, signal --- exec

Code for this lecture

- http://cs162.eecs.berkeley.edu/static/lectures/code04/fork.c
- http://cs162.eecs.berkeley.edu/static/lectures/code04/fork1.c
- http://cs162.eecs.berkeley.edu/static/lectures/code04/fork2.c
- http://cs162.eecs.berkeley.edu/static/lectures/code04/infloop.c
- http://cs162.eecs.berkeley.edu/static/lectures/code04/lowio-std.c
- http://cs162.eecs.berkeley.edu/static/lectures/code04/lowio.c
- http://cs162.eecs.berkeley.edu/static/lectures/code04/pid.c