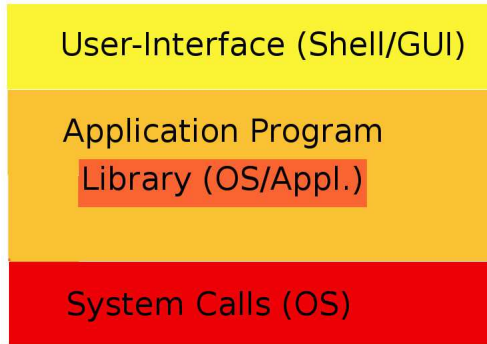


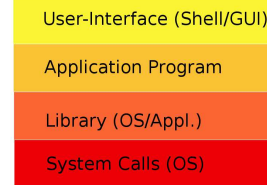
## Library: Static Linking (1)

library functions incorporated into program (at compile time)



## Dynamic Linking

library functions load on demand into program (default, at run-time)



- + saves main memory, smaller size of executable
- + cases saving performance: already in mem, less likely paged out
- + simplifies security patches
  - errors are devastating
  - compatibility with library has to be ensured  $\leadsto$  *dependency hell*

## Library: Static Linking (2)

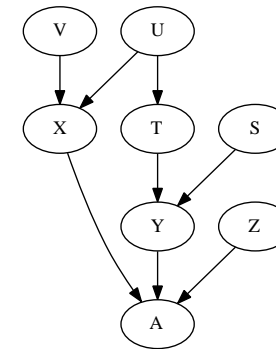
- + applications are more robust (run *standalone*)
- + cases saving performance: many calls to library routines
- + errors in new library version limited to corresp. programs
  - program size bigger
  - changes in library require recompiling program

static linking is enabled by

```
gcc -static ...
```

## The Dependency Hell (with Dynamic Linking)

also called DLL-hell in Windows, JAR-hell in Java,...



$\leadsto$  package manager

## Case Study: A C-Program

## Case Study: Which Shared Libraries Are Involved?

```
$ ldd filetest
filetest:
        libc.so.6 => /lib/libc.so.6 (0x2807c000)
```

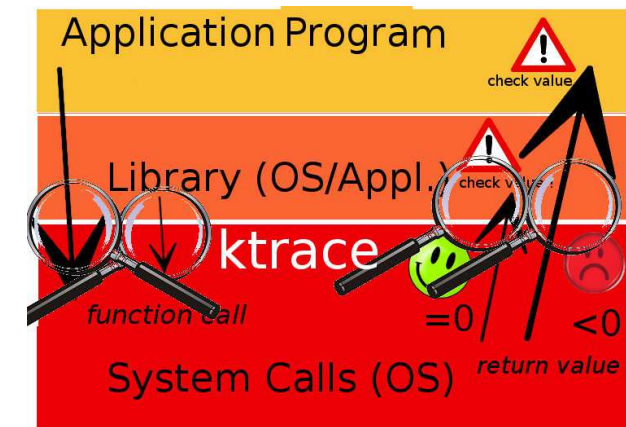
```
#include <stdio.h>

int main(int argc, char **argv)
{
    FILE *f;

    if (argc<=1) return 1;
    f=fopen(argv[1],"w");
    if (!f) return 1;
    fprintf(f,"hello world\n");
    fclose(f);

    return 0;
}
```

## Watching this Call: ktrace (BSD)



(Linux users use strace)

**ktrace / kdump**

```

ktrace ./testprog myfile
kdump -tcn | less
...
16114 testprog CALL  open(0xbfbfe8f7,0x601,0x1b6)
16114 testprog NAMI  "myfile"
16114 testprog RET   open 3
...
16114 testprog CALL  write(0x3,0x804b000,0xc)
16114 testprog RET   write 12/0xc
16114 testprog CALL  close(0x3)
16114 testprog RET   close 0
16114 testprog CALL  exit(0)

```

```

-----
Files          Creating a Channel      creat()
                                   open()
                                   close()
                                   Input/Output      read()
                                   write()
                                   Random Access      lseek()
                                   Channel Duplication  dup()
                                   Aliasing and Removing  link()
Files          unlink()
File Status    stat()
Access Control access()
                                   chmod()
                                   chown()
                                   umask()
Device Control ioctl()

```

**Examples of System Calls**

```

-----
Processes     Process Creation      exec()
                                   fork()
                                   Termination      wait()
                                   exit()
                                   Process Owner and Group  getuid()
                                   geteuid()
                                   getgid()
                                   getegid()
                                   Process Identity      getpid()
                                   getppid()
                                   Process Control      signal()
                                   kill()
                                   alarm()
                                   Change Working Directory  chdir()
-----

```

IPC	Pipelines	pipe()
	Messages	msgget() msgsnd() msgrcv() msgctl()
	Semaphores	semget() semop()
	Shared Memory	shmget() shmat() shmdt()
-----		

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

int main(int argc, char **argv)
{
    int fd; /* file descriptor */
    if (argc<=1) return 1;
    fd=open(argv[1],O_CREAT|O_WRONLY,S_IRUSR|S_IWUSR);
    if (fd<0)
    {
        perror("open");
        return 1;
    }
    if (write(fd,"hello world\n",12)<0)
    {
        perror("write()");
        return 1;
    }
}
```

**C-Program with System Calls**

```
    }
    if (close(fd)<0)
    {
        perror("close()");
        return 1;
    }

    return 0;
}
```

## Commands/Functions/Kernel: The uname Command (1)

Which kernel name?

```
isl-c-01:~$ uname -s
FreeBSD
```

Which hostname?

```
isl-c-01:~$ uname -n
isl-c-01
```

Which operating system release?

```
isl-c-01:~$ uname -r
9.0-STABLE
```

## UNIX is Written in C

...so there is a `uname()` C function ...

## The uname Command (2)

Which kernel version?

```
isl-c-01:~$ uname -v
FreeBSD 9.0-STABLE #0: Fri Jan 27 20:10:29 CET 2012
root@isl-c-01:/usr/obj/usr/src/sys/ISL-L-01
```

Which machine hardware?

```
isl-l-01:~$ uname -m
amd64
```

All of the uname information:

```
isl-c-01:~$ uname -a
FreeBSD isl-c-01.htw-saarland.de
9.0-STABLE FreeBSD 9.0-STABLE
#0: Fri Jan 27 20:10:29 CET 2012
root@isl-l-01:/usr/obj/usr/src/sys/ISL-L-01
amd64
```

```
UNAME(3)      FreeBSD Library Functions Manual      UNAME(3)

NAME
    uname -- get system identification

LIBRARY
    Standard C Library (libc, -lc)

SYNOPSIS
    #include <sys/utsname.h>

    int
    uname(struct utsname *name);

DESCRIPTION
    The uname() function stores NUL-terminated strings of
    information identifying the current system into the structure
    referenced by name.

    The utsname structure is defined in the <sys/utsname.h> header file,
```

and contains the following members:

```

sysname      Name of the operating system implementation.

nodename     Network name of this machine.

release     Release level of the operating system.

version     Version level of the operating system.

machine     Machine hardware platform.

```

#### RETURN VALUES

The uname() function returns the value 0 if successful; otherwise the value -1 is returned and the global variable errno is set to indicate the error.

## A uname C Program

### struct utsname

from /usr/include/sys/utsname.h

```

#ifndef SYS_NMLN
#define SYS_NMLN 256 /* User can override. */
#endif

struct utsname {
    char    sysname[SYS_NMLN]; /* Name of this OS. */
    char    nodename[SYS_NMLN]; /* Name of network node. */
    char    release[SYS_NMLN]; /* Release level. */
    char    version[SYS_NMLN]; /* Version level. */
    char    machine[SYS_NMLN]; /* Hardware type. */
};

```

```

#include <sys/utsname.h>
#include <stdio.h>
int main()
{
    struct utsname buffer;
    if (uname(&buffer)==0)
    {
        puts("system information:");
        printf("sysname: %s\n",buffer.sysname);
        printf("nodename: %s\n",buffer.nodename);
        printf("release: %s\n",buffer.release);
        printf("version: %s\n",buffer.version);
        printf("machine: %s\n",buffer.machine);
    }
    else
        perror("uname");
    return 0;
}

```

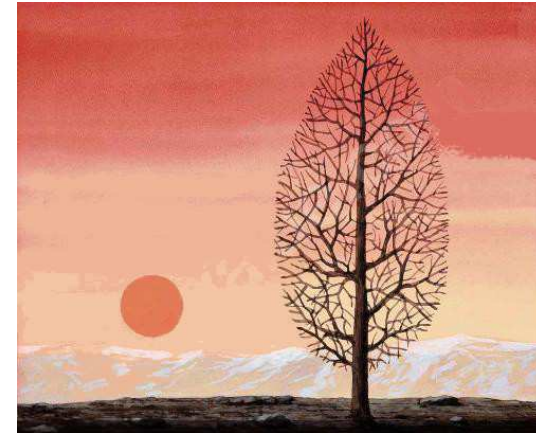
## ktrace of uname C Program

```

60550 ktrace  CALL  execve(0xbfbfe367,0xbfbfe22c,0xbfbfe234)
60550 ktrace  NAMI  "./uname-read"
60550 ktrace  NAMI  "/libexec/ld-elf.so.1"
60550 uname-read RET  execve 0
...
60550 uname-read CALL  open(0x28073040,0,0)
60550 uname-read NAMI  "/lib/libc.so.6"
60550 uname-read RET  open 3
...
60550 uname-read CALL  __sysctl(0xbfbfdc80,0x2,0xbfbfdce0,0xbfbfdc7c,0,0)
60550 uname-read RET  __sysctl 0
...
60550 uname-read CALL  exit(0)

```

## Files, Directories and Inodes



## sysctl

Management Information Base (sort of Registry of a UNIX System)  
for kernel state

```
$ sysctl kern.ostype
kern.ostype: FreeBSD
```

```
$ sysctl kern.osrelease
kern.osrelease: 6.3-RELEASE-p2
```

```
$ sysctl hw.model
hw.model: Intel(R) Pentium(R) 4 CPU 1.80GHz
```

## Files (1)

- abstraction from hardware (Device, Head, Cylinder, Sector)
- call file by a *file name*



/etc/passwd at head 2, cyl 17, sector 73

## Files (2)

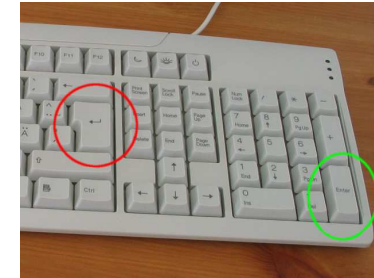
programs need a *reference* to a file

they cannot use the file name on each write operation

- system level: *file descriptor* = `int`
- C library level: *file pointer* = `FILE *`
- C++ library level: *file stream class* = `fstream`
- JAVA : *file stream class* = `FileReader/Writer`

## What *is* a *newline*?

seems like a dull question ...



## Text Files

all UNIX tools understand plain text files in ASCII encoding

only contains printable characters, see `isprint(3)`, `iswprint(3)`

separator is *newline*

usually last line is terminated by a *newline*

otherwise some tools may fail or give unexpected results

the command `wc -l` counts *newline* characters

## This is a *newline* (1)

depends on the operating system

- **LF** UNIX-family:  
GNU/Linux, BSD, Solaris, . . . , Mac OS X, BeOS, Amiga, RISC OS,  
. . .
- **CR+LF** DOS-family:  
DEC RT-11, CP/M, MP/M, DOS, OS/2, Microsoft Windows
- **CR** Apple-family:  
Commodore machines, Apple II family and Mac OS up to version 9

[Wikipedia: Newline]



### This is a *newline* (2)

depends on the character set

for example Unicode:

LF: Line Feed, U+000A

CR: Carriage Return, U+000D

CR+LF: CR followed by LF, U+000D followed by U+000A

NEL: Next Line, U+0085

FF: Form Feed, U+000C

LS: Line Separator, U+2028

PS: Paragraph Separator, U+2029

### Standard File Descriptors

By default, a process has three open file descriptors.

Descriptor	FILE *	meaning	device
0	stdin	standard input	keyboard
1	stdout	standard output	terminal
2	stderr	standard error	terminal

These can be redirected on the shell:

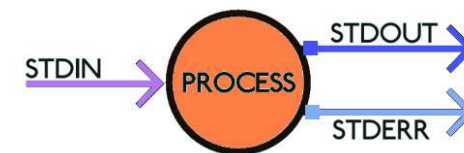
<	redirects the standard input
>	redirects the standard output
>>	redirects the standard output, appends to file
2>	redirects the standard error

### iconv: Converting between Encodings

```
iconv -f utf-8 -t iso8859-1 file.utf8 >file.latin1
```

- -f encoding *from*
- -t encoding *to*
- -c discard encoding errors

### Processes and Stdin/Stdout (default)



processes

- read from `stdin`
- write to `stdout` *and* `stderr`

## Processes and Stdin/Stdout

example: sort reads stdin, writes stdout, EOF is [Strg-d]:

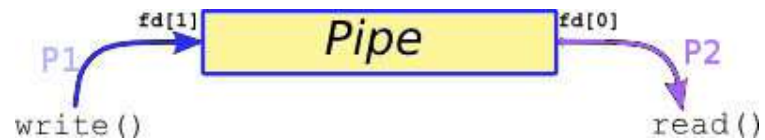
```
$ sort
zhn
rfa
qqm
[press Ctrl-d aka Strg-d]
qqm
rfa
zhn
```

## Pipes: Example

```
$ cat german_soccer
Bayer 04 Leverkusen;56
Borussia Dortmund;64
Eintracht Frankfurt;46
FC Bayern München;84
FC Schalke 04;49
Sport-Club Freiburg;45

$ sort -t";" -k2 -n <german_soccer
Sport-Club Freiburg;45
Eintracht Frankfurt;46
FC Schalke 04;49
Bayer 04 Leverkusen;56
Borussia Dortmund;64
FC Bayern München;84
```

## Pipes: Connect Output of Process 1 to Input of Process 2



Syntax:

```
P1 | P2
```

## Pipes: Example (continued)

```
$ sort -t";" -k2 -n <german_soccer | cut -d";" -f1
Sport-Club Freiburg
Eintracht Frankfurt
FC Schalke 04
Bayer 04 Leverkusen
Borussia Dortmund
FC Bayern München
```

**Pipes: xargs transforms stdin to command line args**

delete all links (simple version, see manual find(1))

example:

```
find . -type l | xargs rm
```