CSE 451 Autumn 2011

Module 1.5: Files / Naming and Project 0 Lessons

John Zahorjan
zahorjan@cs.washington.edu
534 Allen Center

Project 0 Math Library

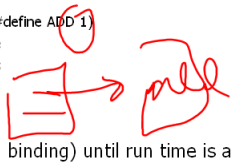
- Naming: a map from one name space to another
  - name → name
- Binding: creating a map entry for a specific name
- Resolution: looking up the map's value for a specific name

- Example: direct procedure call
- `int result = add(x1, x2);`
  - method name → memory address
  - Binding occurs at link time
  - Resolution occurs at link time

©2011 Gribble, Lazowska, Levy, Zahorjan

Math library implementation

- math_call(ADD, &a2, &result);
- There are three name spaces:
  - string → index
    - Binding: compile time (#define ADD 1)
    - Resolution: compile time
  - index → memory address
    - Binding: link time
    - Resolution: run time



- Delaying resolution (and binding) until run time is a generally useful technique

© 2011 Gribble, Lazowska, Levy, Zahorjan

Example 1: syscalls

- `syscall(__NR_fork);`
  - string → index
    - Binding happens at app compile time (#define __NR_fork 2)
    - Resolution happens at compile time
  - index → memory address
    - Binding occurs at kernel link time
    - Resolution occurs at run time (using a function pointer table)

- Why?
  - We sort of have to: can't let caller specify address for protection reasons
  - But… also lets us update kernel without having to update every application!

© 2011 Gribble, Lazowska, Levy, Zahorjan

Example 2: DLLs (Slightly Simplified)

- Dynamic Link Library
  - Link library into process address space at run time
  - Why?
    - One reason: Updating library updates all apps that use it

- `int result = add(x,y);`
  - method name → index
    - Binding happens at compile time
    - Resolution happens at compile time
  - index → memory address
    - Binding occurs at run time (when lib is loaded)
    - Resolution occurs at run time (using a function pointer table)

© 2011 Gribble, Lazowska, Levy, Zahorjan

Example 3: Sub-type Polymorphism

- class Animal defines method toString()
  - subclass Cow overrides toString()

- foo(Animal emma) {
  print emma.toString()
  }
  - To what is "emma.toString" bound, and when?

- method name → index
  - binding happens at compile time
  - resolution happens at compile time
- object → function pointer table (vtable)
  - binding occurs at run time (when object is new'ed)
  - resolution occurs at run time (emma.toString)
- vtable index → memory address
  - binding occurs at link time (when class is instantiated)
  - resolution occurs at run time

© 2011 Gribble, Lazowska, Levy, Zahorjan

Final Example: URLs

- www.cs.washington.edu/education/courses/cse451/11au/index.shtml
  - host name → IP address
    - bind time?
    - resolve time?
  - document name → returned data
    - bind time?
    - resolve time?

© 2011 Gribble, Lazowska, Levy, Zahorjan

The File Abstraction

A file is a simple array of bytes

Files are made larger by writing beyond their current end

Files are named by paths in a naming tree

System calls on files are *synchronous*

Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Naming

(almost) everything has a path name
files

directories

devices (known as *special files*)

- keyboards
- displays
- disks
- etc.

Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Uniformity

```
// opening a normal file
int file = open("/home/twd/data", O_RDWR);

// opening a device (one's terminal or window)
int device = open("/dev/tty", O_RDWR);

// either way, this works
int bytes = read(file, buffer, sizeof(buffer));
write(device, buffer, bytes);
```


Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Working Directory

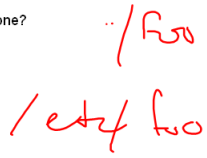
Maintained in kernel *for each process*

paths not starting from "/" start with the working directory

changed by use of the *chdir* system call

displayed (via shell) using "pwd"

- how is this done?



Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Standard File Descriptors

File number	Name	Use
0	stdin	Input
1	stdout	Normal output
2	stderr	Error output

```
main( ) {
        char buf[BUFSIZE];
        int n;
        const char* note = "Write failed\n";
        while ((n = read(0, buf, sizeof(buf))) > 0)
                if (write(1, buf, n) != n) {
                        (void)write(2, note, strlen(note));
                        exit(EXIT_FAILURE);
                }
        return(EXIT_SUCCESS);
}
```


Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Primes program

```
int nprimes;
int *prime;
int main(int argc, char *argv[]) {
   …
   for (i=1; i<nprimes; i++) {
      …
   }
   if (write(1, prime, nprimes*sizeof(int)) == -1) {
      perror("primes output");
      exit(1);
   }
   return(0);
}
```

 Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Human-Readable Output

```
int nprimes;
int *prime;
int main(int argc, char *argv[]) {
   …
   for (i=1; i<nprimes; i++) {
      …
   }
   for (i=0; i<nprimes; i++) {
      printf("%d\n", prime[i]);
   }
   return(0);
}
```

print (sth, to string)

 Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Running It

```
$ /home/twd/bin/primes 300 >/home/twd/Output
```

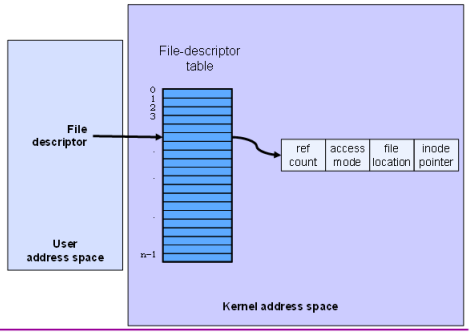
```
if (fork() == 0) {
   /* set up file descriptor 1 in the child process */
   close(1);
   if (open("/home/twd/Output", O_WRONLY) == -1) {
      perror("/home/twd/Output");
      exit(1);
   }
   execl("/home/twd/bin/primes", "primes", "300", 0);
   exit(1);
}

/* parent continues here */

while(pid != wait(0))/* ignore the return code */
   ;
```

 Copyright © 2010 Thomas W. Doeppner. All rights reserved.

File-Descriptor Table



 Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Allocation of File Descriptors

Whenever a process requests a new file descriptor, the lowest numbered file descriptor not already associated with an open file is selected; thus

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

close(0);
fd = open("file", O_RDONLY);
```

will always associate *file* with file descriptor 0 (assuming that the *open* succeeds)

 Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Redirecting Output … Twice

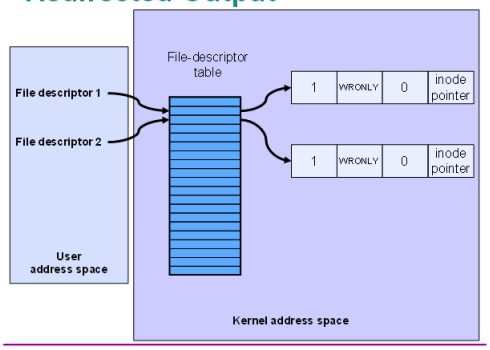
```
$ /home/twd/bin/primes 300 >/home/twd/Output 2>/home/twd/Output
```

```
if (fork() == 0) {
   /* set up file descriptors 1 and 2 in the child process */
   close(1);
   close(2);
   if (open("/home/twd/Output", O_WRONLY) == -1) {
      exit(1);
   }
   if (open("/home/twd/Output", O_WRONLY) == -1) {
      exit(1);
   }
   execl("/home/twd/bin/program", "program", 0);
   exit(1);
}

/* parent continues here */
```

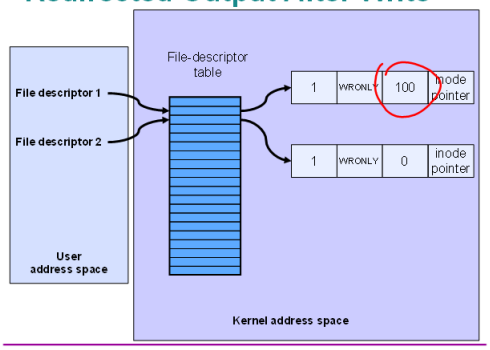
 Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Redirected Output



Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Redirected Output After Write



Copyright © 2010 Thomas W. Doeppner. All rights reserved.

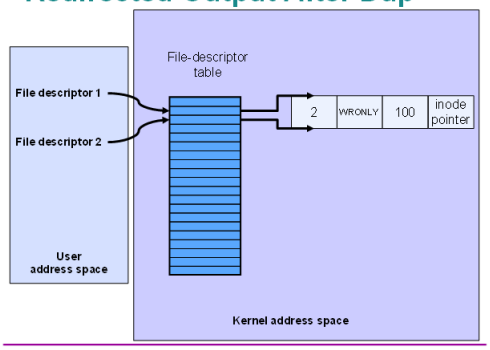
Sharing Context Information

```
$ /home/twd/bin/primes 300 >/home/twd/Output 2>&1
```

```
if (fork() == 0) {
   /* set up file descriptors 1 and 2 in the child process */
   close(1);
   close(2);
   if (open("/home/twd/Output", O_WRONLY) == -1) {
      exit(1);
   }
   dup(1); /* set up file descriptor 2 as a duplicate of 1 */
   execl("/home/twd/bin/program", "program", 0);
   exit(1);
}
/* parent continues here */
```


Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Redirected Output After Dup



Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Fork and File Descriptors

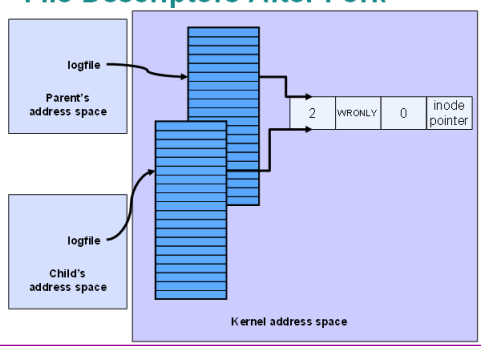
```
int logfile = open("log", O_WRONLY);
if (fork() == 0) {
   /* child process computes something, then does: */
   write(logfile, LogEntry, strlen(LogEntry));
   …
   exit(0);
}

/* parent process computes something, then does: */

write(logfile, LogEntry, strlen(LogEntry));
…
```

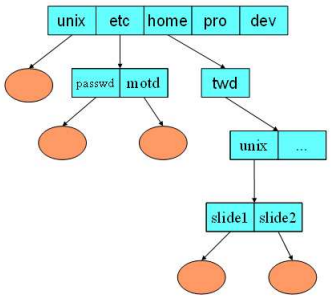

Copyright © 2010 Thomas W. Doeppner. All rights reserved.

File Descriptors After Fork



Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Directories



Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Directory Representation

Component Name	Inode Number

directory entry

.	1
..	1
unix	117
etc	4
home	18
pro	36
dev	93

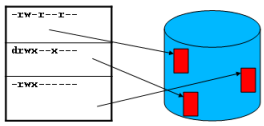
string → inode index

Copyright © 2010 Thomas W. Doeppner. All rights reserved.

File (Contents) Representation

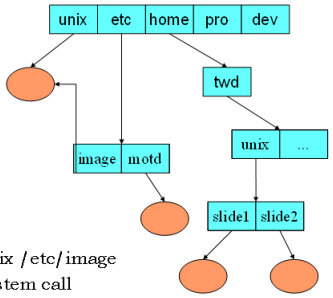
- (Regular) file inodes:
  - maintain file meta-data (e.g., permissions)
  - describe where on disk the file contents reside

inode index → disk address



Copyright © 2010 Thomas W. Doeppner. All rights reserved.

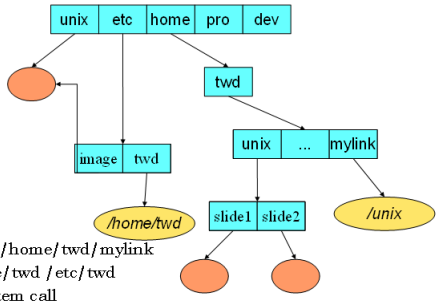
Modifying the Name Space: Hard Links



% ln /unix /etc/image
# link system call

Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Modifying the Name Space: Soft Links



% ln –s /unix /home/twd/mylink
% ln –s /home/twd /etc/twd
# symlink system call

Copyright © 2010 Thomas W. Doeppner. All rights reserved.

Summary

- File names maps to inode indices
  - That's the information that is in directories
- Inodes contain metadata
  - e.g., type, permissions, modified time, owner, …
- Inodes also describe where file contents are on disk
- There are operations to:
  - act on the name space
    - open (create), mv, cp, ln
  - act on the metadata
    - chmod, chown, touch, …
  - act on the contents
    - write, truncate, …

Copyright © 2010 Thomas W. Doeppner. All rights reserved.