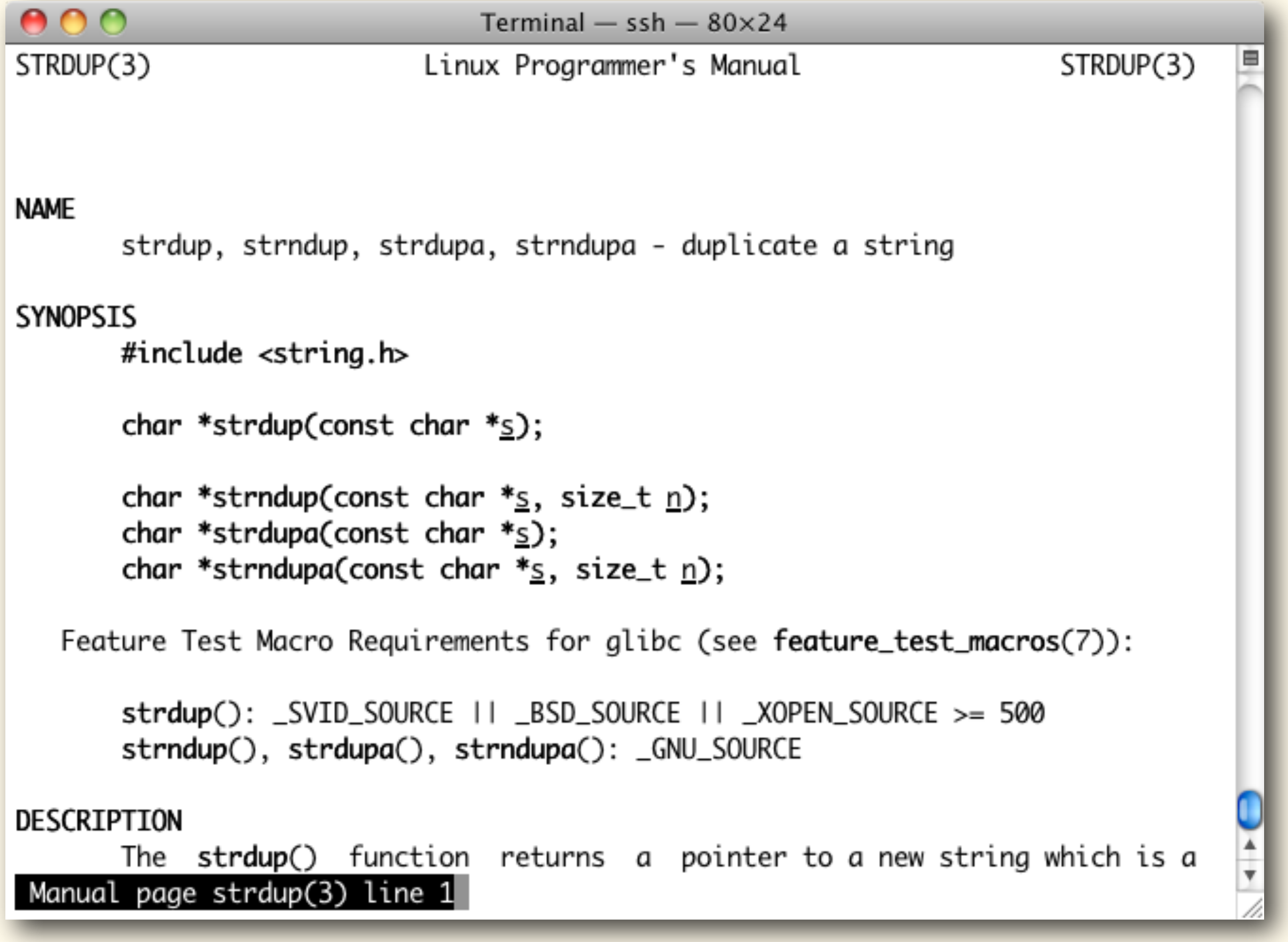
Operating Systems Structure

Computer Science 432 - Lecture 3 - Duane Bailey

February 9, 2022

Announcements

- Code Walkthroughs today & tomorrow. Zooms in calendar
- Office Hours: T1-3, F9-10:30 Hybrid on Friday
- Ideal: in-person beginning Monday.
  - Lectures in Wachenheim 114
  - Labs in Ward
  - Code Walkthroughs in Knuth
- Contact me if you are isolated



```
STRDUP(3)                  Linux Programmer's Manual                 STRDUP(3)



NAME
       strdup, strndup, strdupa, strndupa - duplicate a string

SYNOPSIS
       #include <string.h>

       char *strdup(const char *s);

       char *strndup(const char *s, size_t n);
       char *strdupa(const char *s);
       char *strndupa(const char *s, size_t n);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       strdup(): _SVID_SOURCE || _BSD_SOURCE || _XOPEN_SOURCE >= 500
       strndup(), strdupa(), strndupa(): _GNU_SOURCE

DESCRIPTION
       The  strdup()  function  returns  a  pointer to a new string which is a
 Manual page strdup(3) line 1
```

Hints for Computer System Design 1984 & 2021 — Butler Lampson

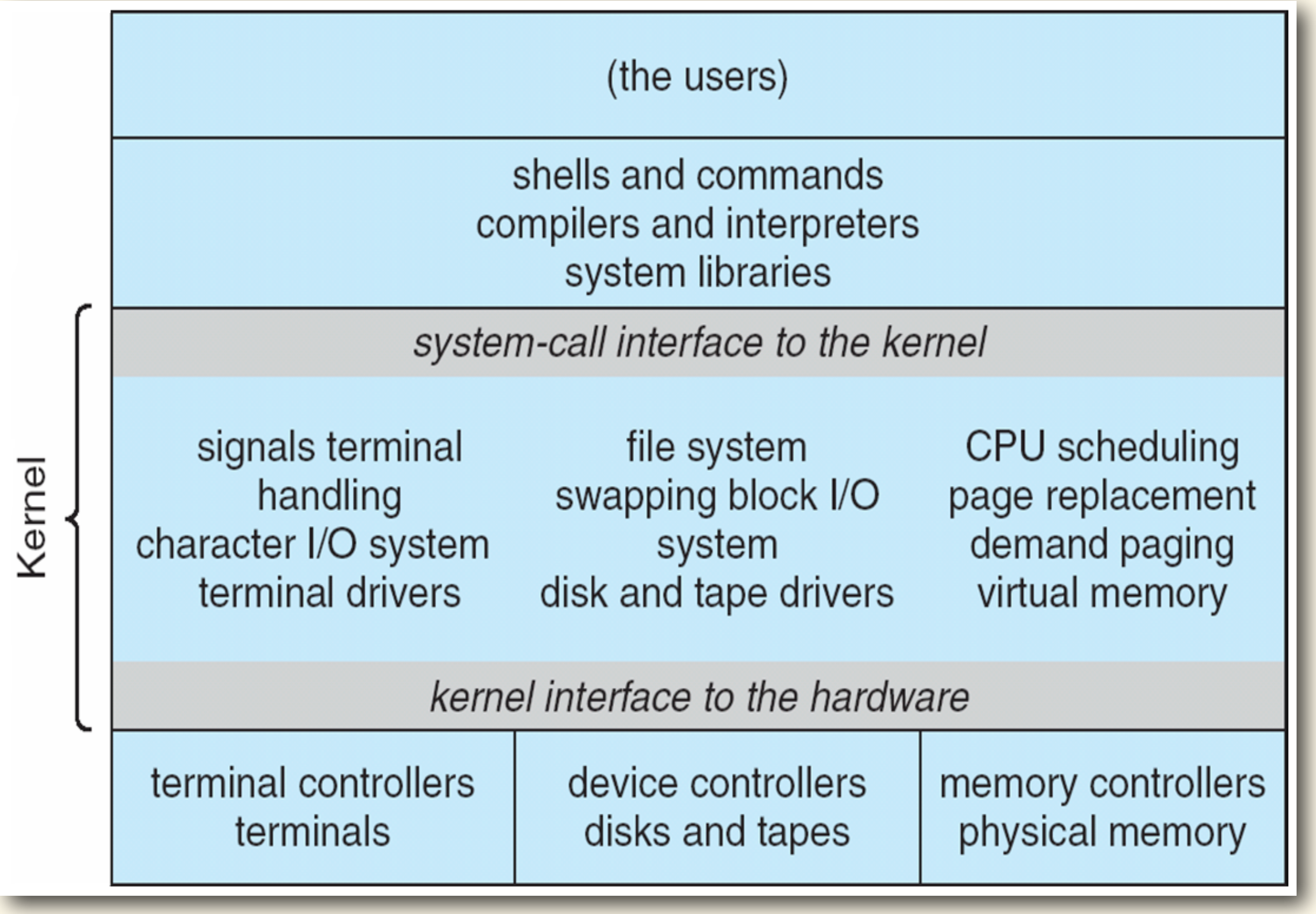
- A system designer of nearly unparalleled experience. Microsoft fellow at MIT. Turing Award winner, among many other kudos.
  - Is there a Zen of design? No.
  - Are there Rules of Thumb? Sure.
    - Systems are complex. Keep them as simple as possible.
    - "Good implementation is not impossible. It's merely hard."
    - Get it right. Make it fast. Expose power, but be flexible. Hide.
    - The client is usually most informed. Help them help themselves.
    - Stick to an interface, but plan on prototypes.
    - Share resources. Cache results. Identify hints.
    - Just do it, computing offline if possible. Delegate.
    - Handle errors. Use logs. Checkpoint if possible.

The Unix System
1974—Ritchie & Thompson

- Unix: A general purpose operating system:
  - Less than $40,000
  - Two man-years to construction
  - Successful because it met no particular need
- Realize some important things:
  - You can make do with less
  - Small levers move big rocks
  - Great ideas appear in the *beginning* of great systems
  - Great is rarely big

Typical Structure of an O/S

Unix world view



(the users)

shells and commands
compilers and interpreters
system libraries

system-call interface to the kernel

Kernel

signals terminal handling
character I/O system
terminal drivers

file system
swapping block I/O system
disk and tape drivers

CPU scheduling
page replacement
demand paging
virtual memory

kernel interface to the hardware

terminal controllers
terminals

device controllers
disks and tapes

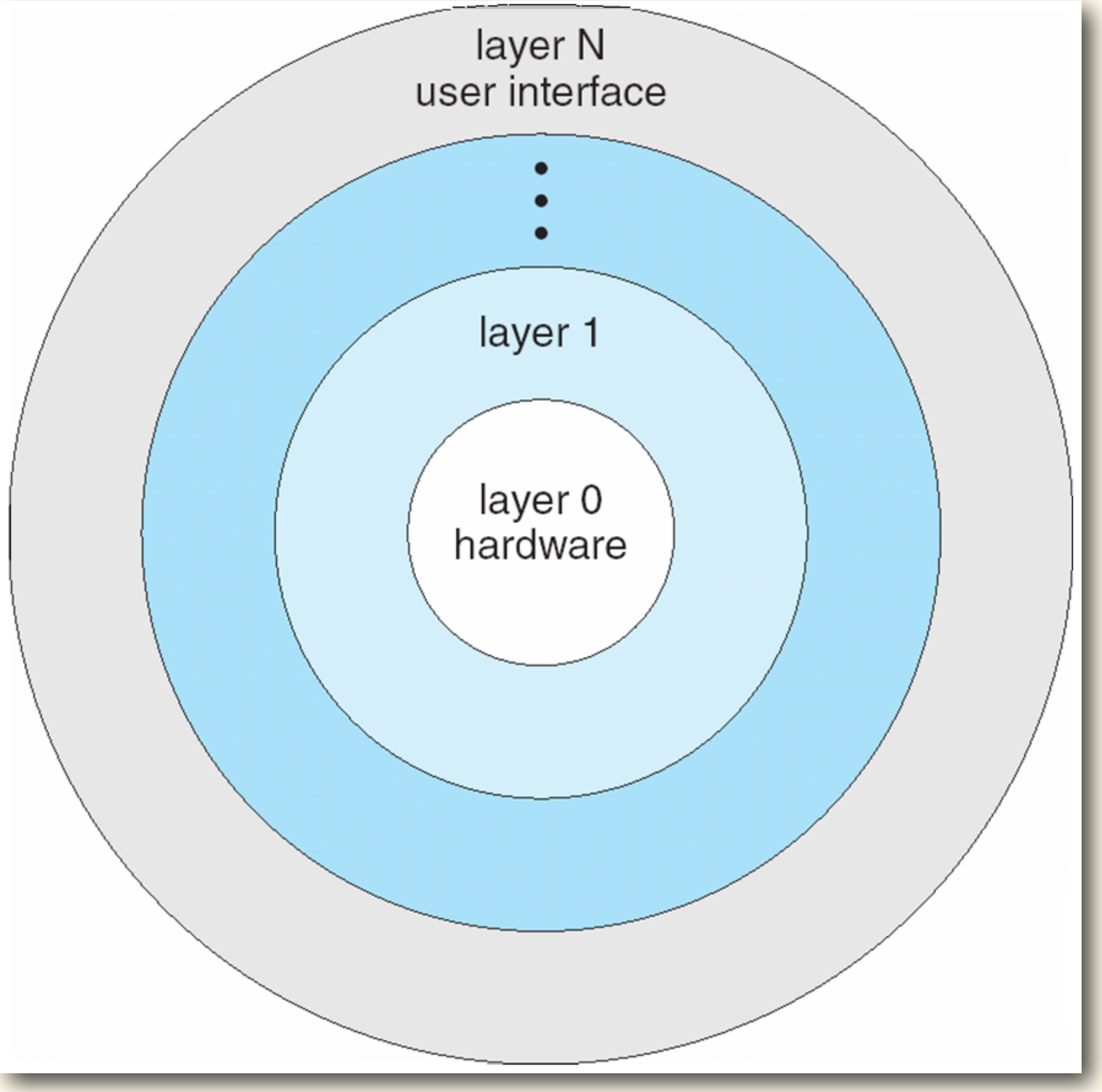
memory controllers
physical memory

O/S Services for User Support

- Command interpretation
  - Integrated (most early O/S's, aside from Unix)
    - Secure
    - Hard to modify
  - Separate (shell execution, sh, bash, etc.)
    - Anyone can extend the command set
    - Everyone can have a different view
- Program loading and process execution
  - Process control (fork, wait)
  - Loading and dynamic linking (exec, mmap, etc.)
- I/O and File support
  - Agnostic "file" descriptors
  - Layout, security, integrity (open, read, write, close, link, unlink)

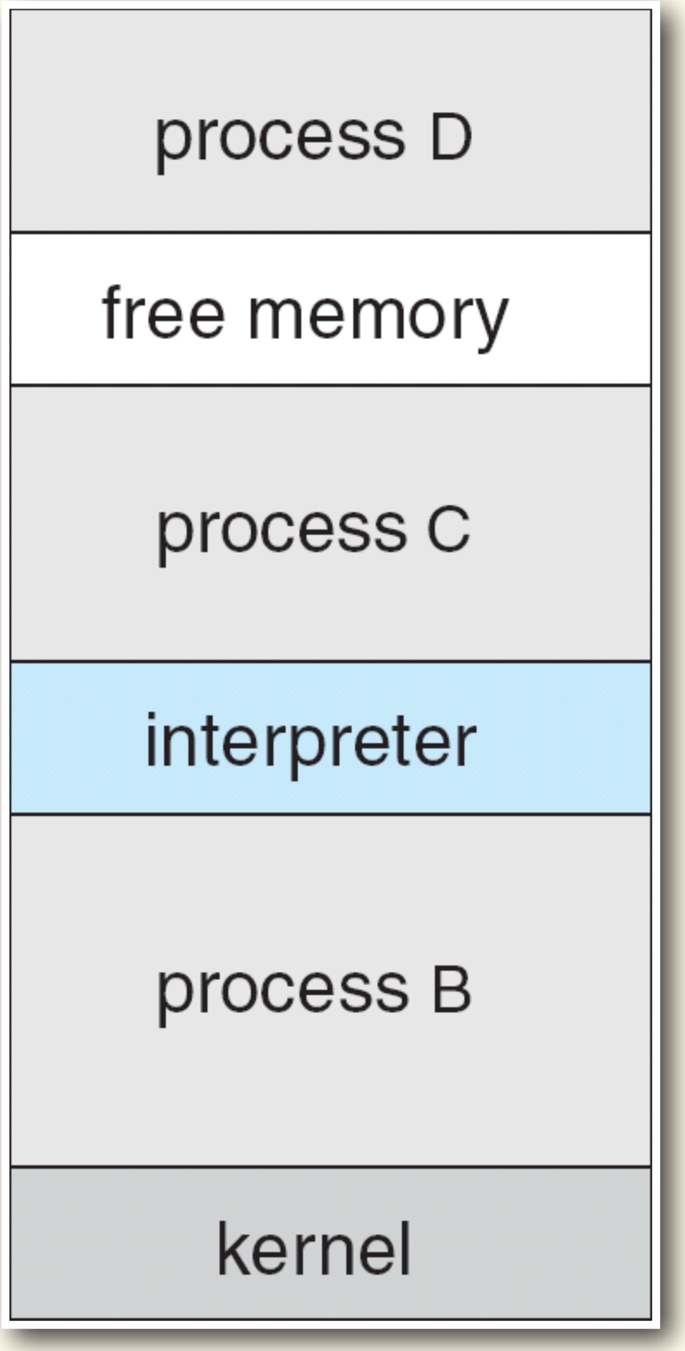
Multiple Personalities

- In layered, or ring-architecture O/S (older DEC systems) privilege is escalated in a series of abstraction layers
- Outer layers provide abstract user services
- Middle layers support administrative services (logical devices, loader services, etc.)
- Inner layers manage interface to hardware (center)
- Unix provides a 3-layer system



Focus: O/S Split Personality

- Most (but not all) operating systems support at least one privileged mode of execution, supporting this view:
  - Most applications are not privileged (Word, grep, shells/interpreters)
    - They can only access their own memory, ie.
    - They cannot access anyone else's memory
  - Privileged accesses typically reside in the kernel:
    - The kernel can do anything, anywhere
  - Users must ask the kernel to perform privileged operations on their behalf
    - The kernel is then responsible for limiting access, protecting the machine

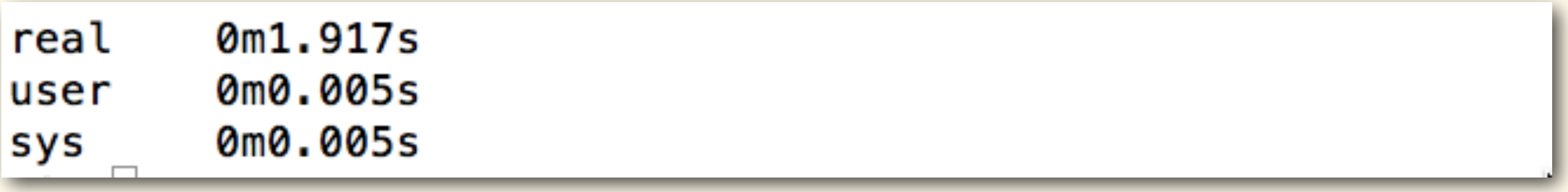


Hardware Support for O/S

- Timers & clocks
- Special instructions: halt (m68k); int, syscall (x86); ecall (RISC-V); context switch (VAX)
- Memory protection
- Limited access to I/O control memory or instructions
- Protected modes of execution (RISC-V: 3 modes, x86: 4, m68k: 2)
- Mechanisms for raising or lowering protection
  - Anything that changes the code segment (interrupts, etc.)
- Synchronization primitives (load-reserved, store-conditional, etc.)
- Threading support
- Virtualization

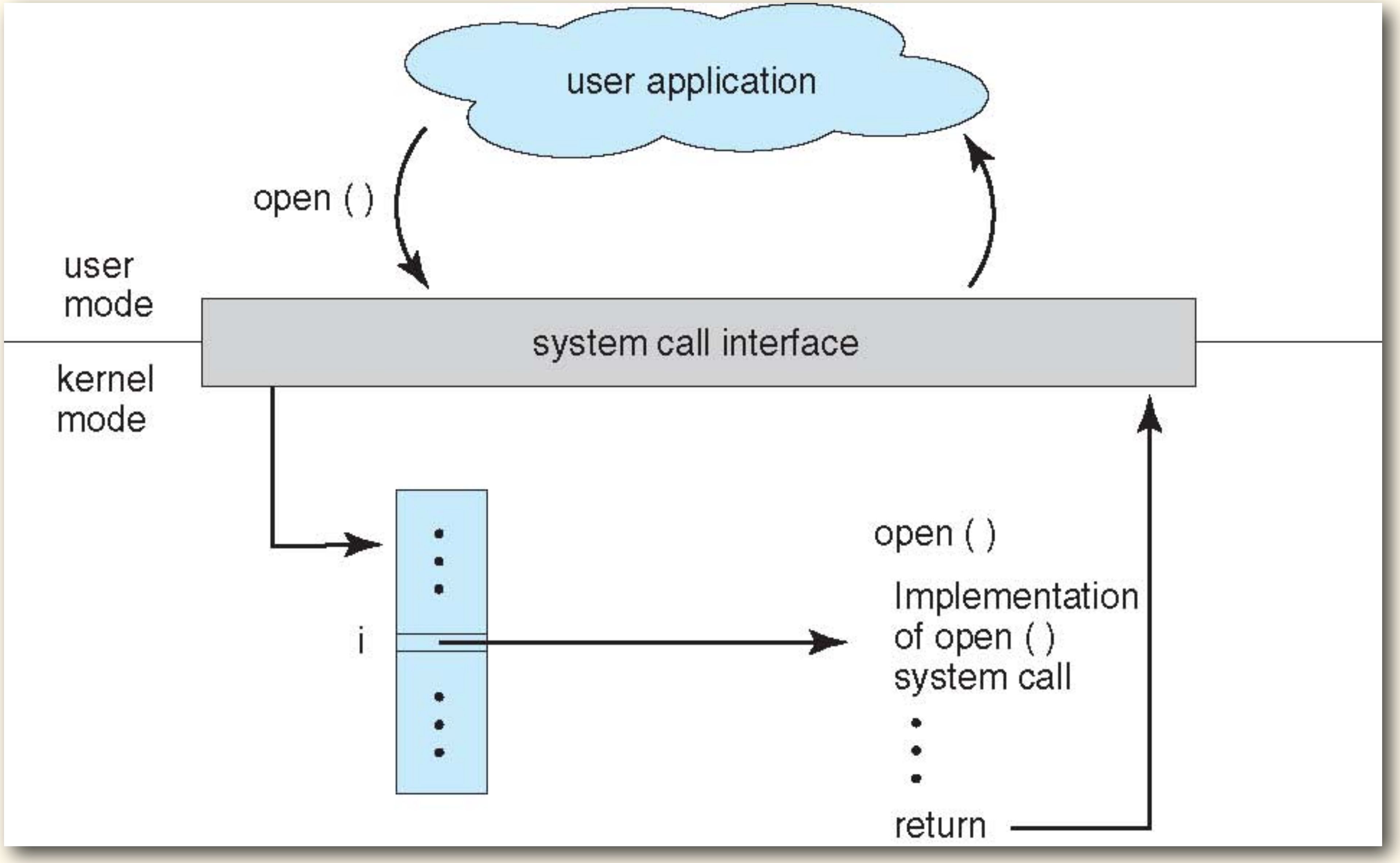
The time(1) Command

- Describes the amount of time consumed by a program.
  - The real (elapsed wall-clock) time
  - The user time — actual time the program was running as user
  - The system time — actual time the program was running as kernel



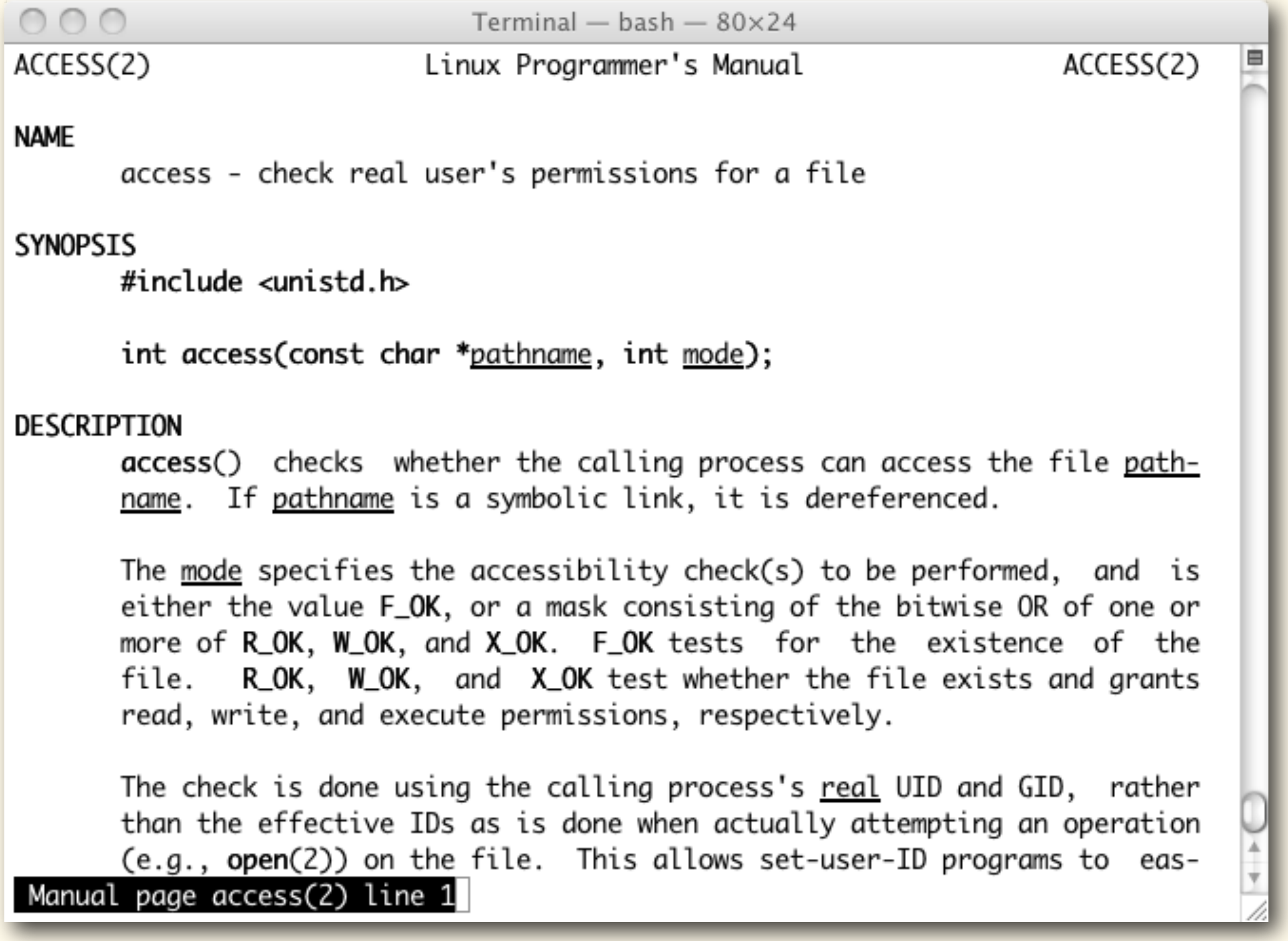
```
real	0m1.917s
user	0m0.005s
sys	0m0.005s
```

The Unix System Call



System Calls

- All system calls are documented in Section 2 of the Unix manual:
  - To get a short overview of system calls: man 2 intro
  - To get a list of system calls: man 2 syscalls



```
ACCESS(2)                  Linux Programmer's Manual                 ACCESS(2)

NAME
       access - check real user's permissions for a file

SYNOPSIS
       #include <unistd.h>

       int access(const char *pathname, int mode);

DESCRIPTION
       access()  checks  whether the calling process can access the file path-
       name.  If pathname is a symbolic link, it is dereferenced.

       The mode specifies the accessibility check(s) to be performed,  and  is
       either the value F_OK, or a mask consisting of the bitwise OR of one or
       more of R_OK, W_OK, and X_OK.  F_OK tests  for  the  existence  of  the
       file.   R_OK,  W_OK,  and  X_OK test whether the file exists and grants
       read, write, and execute permissions, respectively.

       The check is done using the calling process's real UID and GID,  rather
       than the effective IDs as is done when actually attempting an operation
       (e.g., open(2)) on the file.  This allows set-user-ID programs to  eas-
 Manual page access(2) line 1
```