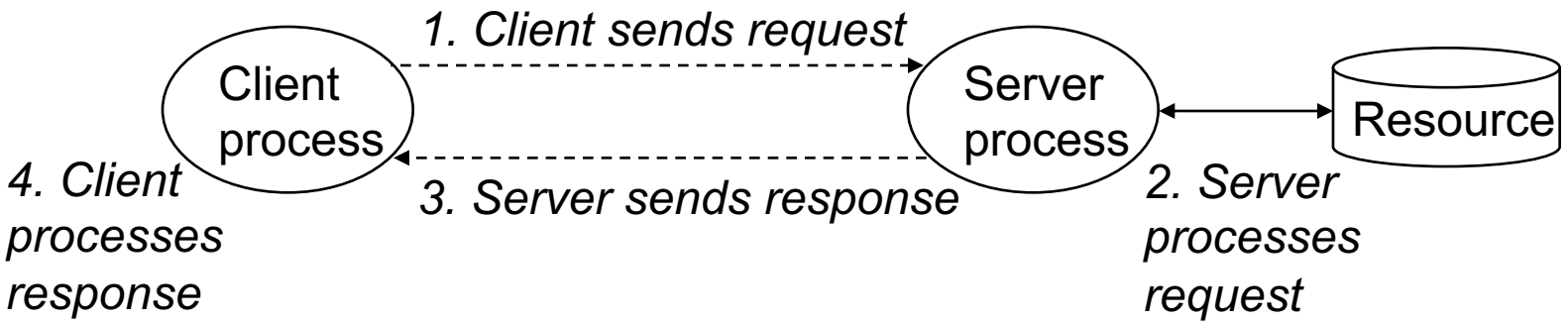
456: Network-Centric Programming

Yingying (Jennifer) Chen

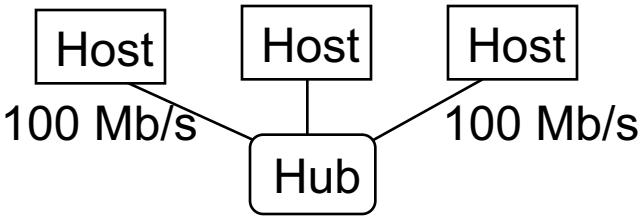
Web: https://www.winlab.rutgers.edu/~yychen/
Email: yingche@scarletmail.rutgers.edu

Department of Electrical and Computer Engineering
Rutgers University

The Client–Server Model

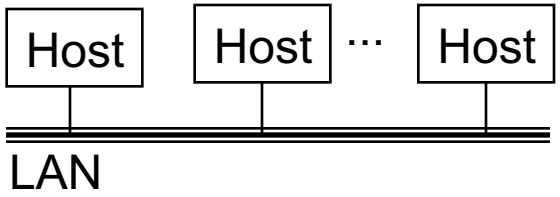


Network Structures and Elements: A Local Area Network (LAN)

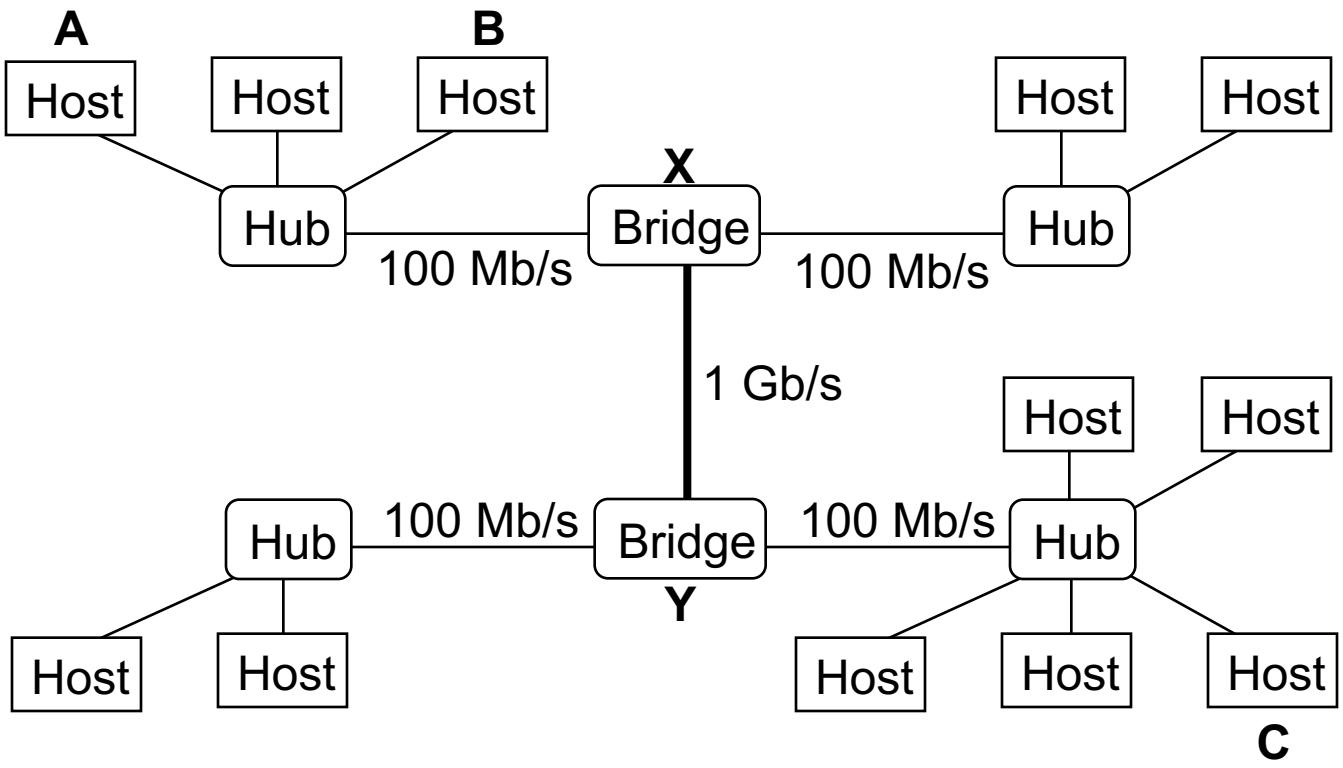


A hub slavishly copies every bit that it receives on each port to every other port.

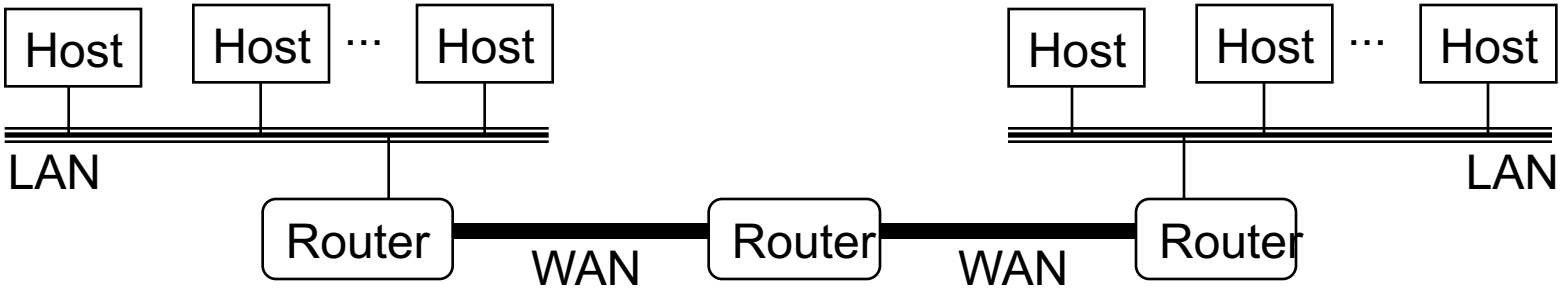
Each wire has the same maximum bit bandwidth, typically 100 Mb/s or 1 Gb/s.



A Campus Network



A Small Internet

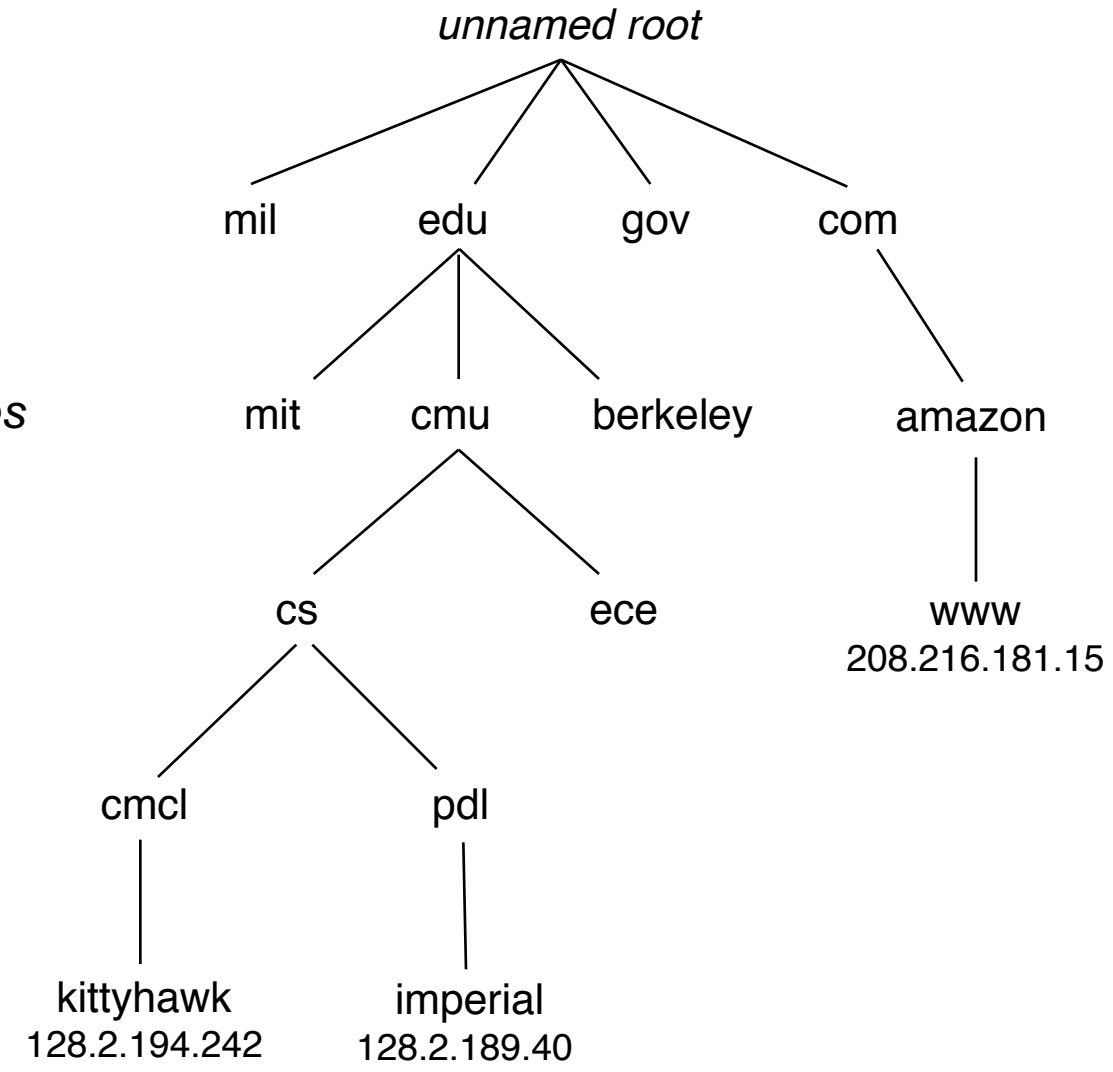


Naming and Addressing(DNS)

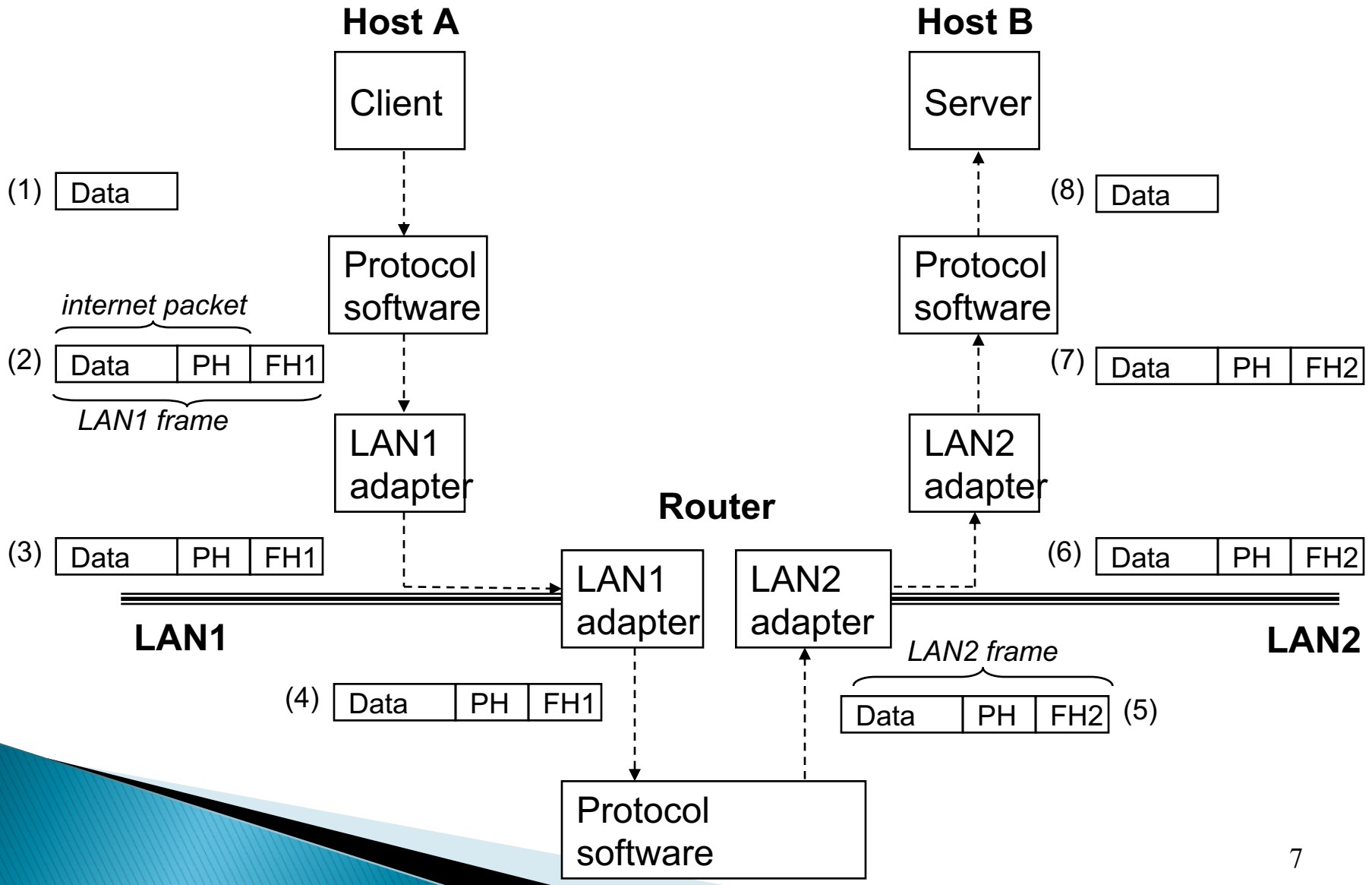
First-level domain names

Second-level domain names

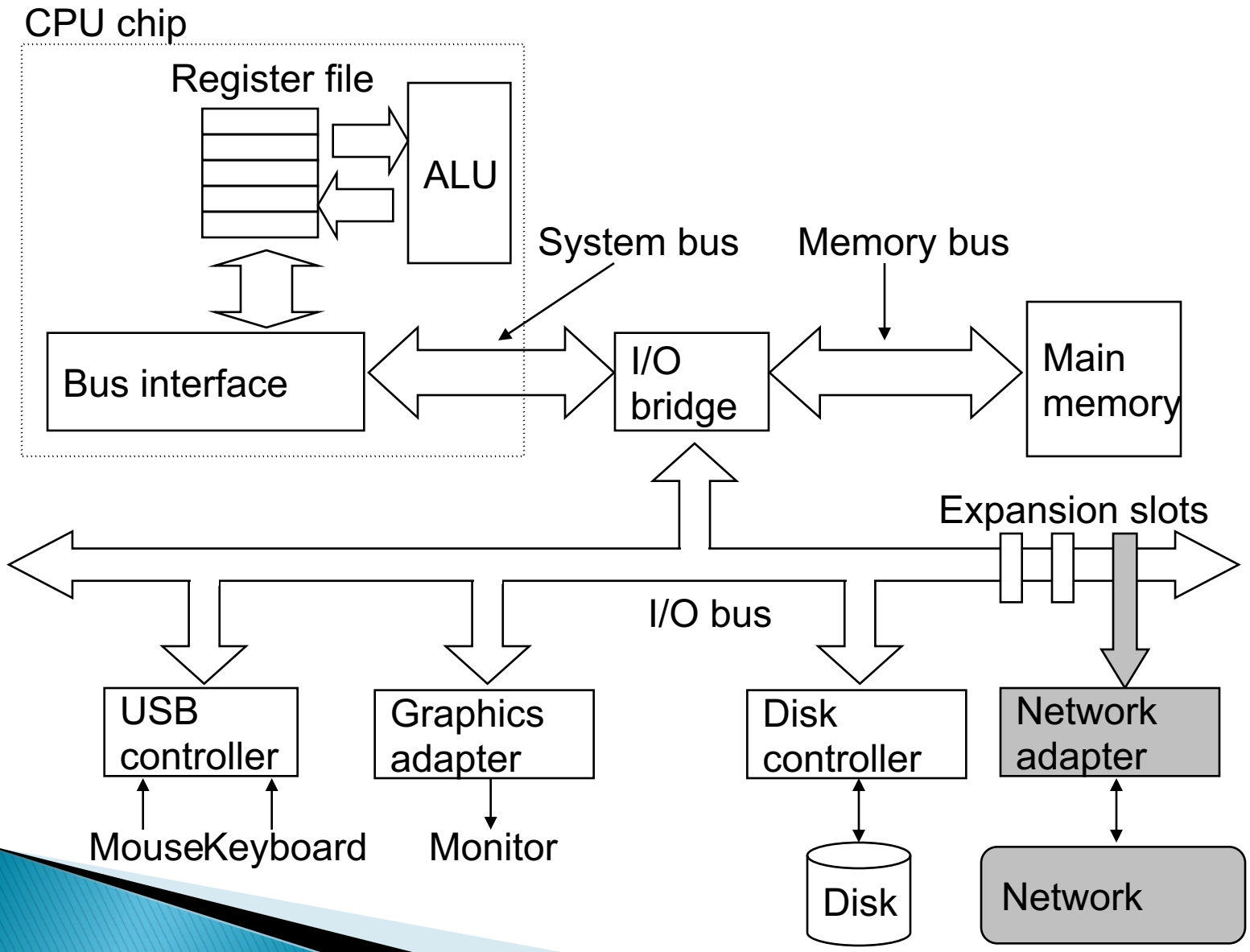
Third-level domain names



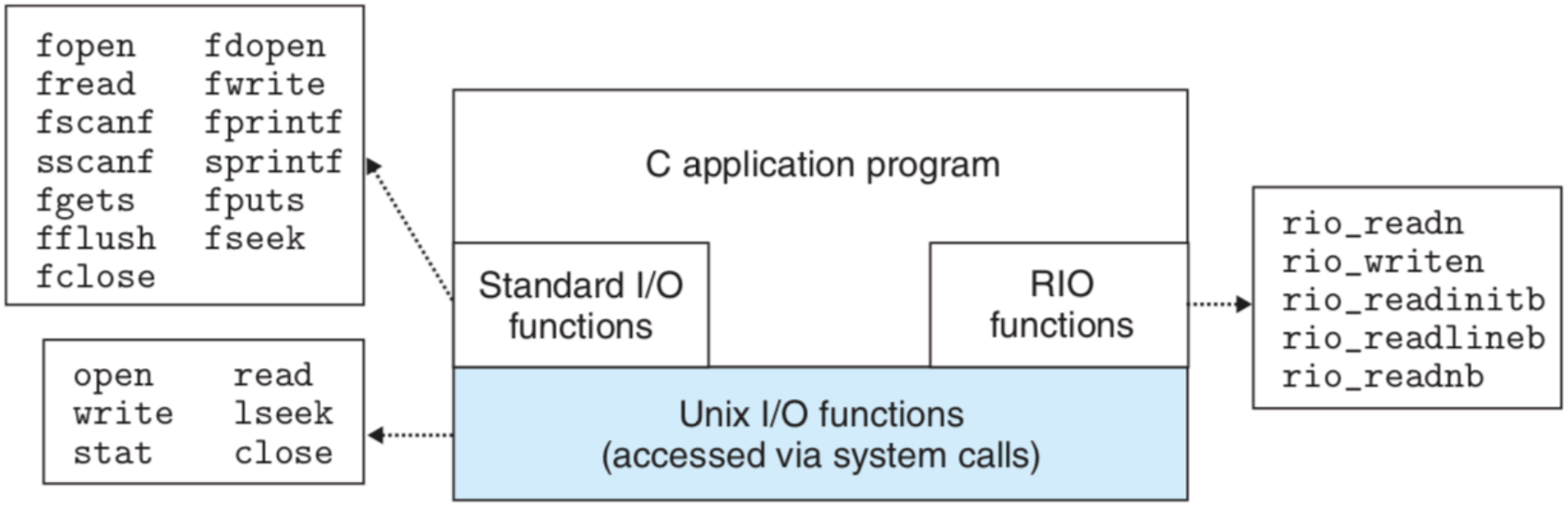
Network Transmission of Packets: Encapsulation



Uniform File-based I/O Model



Standard I/O, Unix I/O and RIO



Standard I/O Library

- First low-level Unix (and Windows) I/O: numeric file descriptor (file handle)
- Or Unix, now ANSI C
- Handles
  - buffer allocation: read into large buffer, dump to OS in fixed units
  - performs I/O in optimal-sized chunks

Standard I/O Library

- Portable C I/O library
  - first for Unix, now ANSI C – runs on Windows etc.
- Usually, much more efficient than system calls (e.g., read(), write())
  - if data is accessed in small chunks
  - Buffering leads to fewer system calls

Stream

- A stream of type FILE is an abstraction for a file descriptor and a stream buffer.
- stdio library manipulates streams
- Associate stream with file
- Work for files, but also interprocess communications
- Stream is also known as the file pointer

Stream Operation

- fopen() returns pointer to FILE object (*file pointer*)
  - file descriptor
  - pointer to buffer
- fclose()
  - Close stream
  - Broke the connection to the corresponding file
- fread() / fwrite() / fseek() / ftell() library functions

Standard I/O and System Calls

- Standard I/O
  - fopen
    - returns pointer to FILE structure (file pointer)
    - pointer to buffer
  - fclose
  - fread
  - Fwrite
  - fseek / ftell
  - fgetc / fputc
  - fgets / fputs
- C Function / Macro for every system call
  - open
  - close
  - read
  - write
  - lseek

Low-level File I/O Functions

- C Function / Macro for every system call
  - open
  - read
  - write
  - lseek
  - close
- Unbuffered I/O
- Every read or write call invokes system call
- Open files are identified by *file descriptor* (nonnegative integer)
- Shells follow this convention
  - 0 = STDIN_FILENO
  - 1 = STDOUT_FILENO
  - 2 = STDERR_FILENO

Standard I/O: fopen()

- FILE* fp=fopen(const char* pathname, const char* mode)
  - Returns file pointer (to buffer)
  - Mode
    - “r” – read-only mode
    - “w” – write mode, overwrite if file exist, otherwise create a new file
    - “a” – append access, write at the end, otherwise create a new file
    - “r+” – read and write mode
    - “w+” – read and write mode, create a new one if exist
    - “a+” – open or create file for both reading and appending

System Calls: open()

- int fd=open(const char* pathname, int oflag, mode_t mode)
  - oflags
    - One is mandatory: O_RDONLY, O_WRONLY, or O_RDWR
    - Optional O_APPEND, O_CREAT, O_SYNC
  - mode
    - Permissions for a new created file
    - Can be ignored if no new file is created
      - S_IRWXU 00700 user (file owner) has read, write and execute permission
      - S_IRUSR 00400 user has read permission
      - S_IWUSR 00200 user has write permission
      - S_IXUSR 00100 user has execute permission

Standard I/O:
Reading and Writing a Stream

- fread()
  - size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
  - Read *nmemb* items of size *size* from *stream* and store them at the location pointed to by *ptr*
- fwrite()
  - size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
  - Write *nmemb* items of size *size* to *stream* from the location pointed to by *ptr*
- Useful for reading/writing arrays and structs
- Returns number of items written

System calls: read() and write()

- read()
  - Read from a file descriptor
  - int read(int fd, void *buf, int count)
  - Read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*
- write()
  - Write to a file descriptor
  - int write(int fd, void *buf, int count)
  - Write writes up to *count* bytes to the file referenced by the file descriptor *fd* from the buffer starting at *buf*.

fclose() / close()

- fclose(FILE* fp)
  - Close stream
  - Break the connection to the corresponding file
- close(int filedescriptor)
  - Cleans up kernel data structures
  - Files automatically closed if program ends

Character input/output

- int fgetc (FILE *fp)
  - Read the next character as an unsigned char from the stream and returns its value, converted to an int.
  - If an end-of-file condition or read error occurs, return EOF.
- int fputc (int c, FILE *fp)
  - Converts the character c to type unsigned char, and writes it to the stream.
  - If succeed, The character is returned.
  - If a write error occurs, return EOF.

File Pointers and Seeking

- The system remembers the position in a (real) file through a file pointer (32-bit offset), which automatically advances when you read or write
- File Positioning
  - Query or modify the position of the pointer
  - `long ftell(FILE *fp);`
  - `int fseek(FILE *fp, long offset, int whence); // SEEK_SET, SEEK_CUR, SEEK_END`
  - `void rewind(FILE *fp);`

Standard I/O: ftell()

- int ftell (FILE *fp)
  - Obtain the current value of the file position indicator for the stream pointed to by stream.
  - Return the current file position of the stream
  - Return -1 if fail

Standard I/O: fseek()

- int fseek (FILE *fp, long int offset, int whence)
  - Change the file position of the stream
  - whence can be SEEK_SET, SEEK_CUR, or SEEK_END
    - SEEK_SET specifies the offset relative to the beginning of the file
    - SEEK_CUR specifies the offset relative to the current file position
    - SEEK_END specifies the offset relative to the end of the file
  - Return zero if successful
  - Return -1 if fail

*SEEK_SET, SEEK_CUR, or SEEK_END are also used in lseek function

Standard I/O: rewind()

- `void rewind(FILE *fp)`
  - Position the stream at the beginning of the file.
  - equivalent to calling fseek with an offset argument of 0 and a whence argument of SEEK_SET.

System Calls: lseek()

- off_t lseek (int filedes, off_t offset,
  int whence)
  - Change the file position of the file with descriptor filedes.
  - Underlying primitive for the fseek, ftell and rewind functions, which operate on streams instead of file descriptors.

Error Handling

- Most system call library functions return -1 (or sometimes 0) when an error occurs
  - Must be checked after every function call
- The global errno variable contains an error number that describes why the function failed
  - Errno only valid if an error has occurred
- Use strerr() or perror() functions to convert error numbers to meaningful strings

strerror()

- char * strerror (int errnum)
  - Maps the error code to a descriptive error message string.
  - Return a pointer to error message string.
  - The value *errnum* normally comes from the variable errno.

perror()

- void perror (const char *message)
  - Prints an error message to the stream stderr
  - If message is null, prints the error message corresponding to errno
  - If the message in non-null, prefixes the error string corresponding to errno with this message

Error Handling

```
#include      <errno.h>
#include      "ourhdr.h"

int
main(int argc, char *argv[])
{
    /* … some library call here */
   fprintf(stderr, "EACCES: %s\n", strerror(EACCES));

   /* another example – perror directly prints to stderr */
   errno = ENOENT;
   perror(argv[0]);  \\

   exit(0);
}
```

Make this an concrete example

EACCES: error access or permission denied
ENOENT: no such file or directory

Examples

- files
  - Write a string to a file, and read the file and display the string.
- fgetc
  - Read from the txt file using fgetc() and display it.
- erron
  - Try two ways (i.e., *strerror()* and *perror()*) to print out error string.

Examples

- copy
  - Copy stdin to stdout.
- systemcalls
  - Read from a txt file using system calls, and display the number of bytes were read, and the content.

Example: files lecture_2_code/files

```
#include <stdio.h>
#include <string.h>
//write a string to a file, and read the file and display the string.
int main () {
   FILE *fp;
   char c[] = "Network Centric Programming";
   char buffer[100];

   /* Open file for both reading and writing */
   fp = fopen("file.txt", "w+");

   /* Write data to the file */
   fwrite(c, sizeof(c), 1, fp);
```

Example: files lecture_2_code/files

```
  /* Seek to the beginning of the file */
   fseek(fp, 0, SEEK_SET);

   /* Read and display data */
   fread(buffer, sizeof(c), 1, fp);
   printf("%s\n", buffer);
   fclose(fp);

   return(0);
}
```

```
% make
% ./main
```

Example: fgetc lecture_2_code/fgetc

```
#include <stdio.h>
// read from file.txt
int main () {
   FILE *fp;
   int c;
   fp = fopen("file.txt","r"); //read the file
   while(1) {
      c = fgetc(fp);
      if( feof(fp) ) {    // feof() is used to find the end of a file
         break ;
      }
      printf("%c", c); // print out each char read from file.txt
   }
   fclose(fp);
   return(0);
}
```

```
% make
% ./main
```

Example: erron lecture_2_code/errno

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
int main () {
   FILE *fp;
   /* now let's try to open the file, which does not exist in this directory.
*/

   // try two ways to print out error string.
   fp = fopen("file.txt", "r");
   if( fp == NULL ) {
            printf("Error: %s\n", strerror(errno));
            printf("----------\n");
```

Example: erron lecture_2_code/errno

```
        perror("Error: ");
        return(-1);
    }
    fclose(fp);

    return(0);
}
```

```
% make
% ./main
```

Example: copy lecture_2_code/copy

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

//copy stdin to stdout

int main() {
  char buf[10000];
  int n=0;
  while((n=read(STDIN_FILENO, buf, sizeof(buf))) != 0) {
    write(STDOUT_FILENO, buf, n);
  }
  return 0;
}
```

```
% make
% ./main
```

Example: systemcalls lecture_2_code/systemcalls

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
//read from a txt file using system calls, and display the number of bytes were read, and the content.
int main() {
```

Example: systemcalls lecture_2_code/systemcalls

```
char *c;
int fd, sz;

c = (char *) calloc(100, sizeof(char)); // allocate 100 char-size buffer

fd = open("file.txt", O_RDONLY, 0);
if (fd < 0) { perror("r1"); exit(1); }

sz = read(fd, c, 20);
printf("Returned that %d bytes were read.\n", sz);
c[sz] = '\0';
printf("Those bytes are as follows: %s\n", c);
}
```

```
% make
% ./main
```

Reading Homework

- CSAPP: Computer Systems – A Programmer's Perspective
  - Reading Sec10.1 – 10.3,10.8; Sec 11.1–11.2.
- UNP: Unix Network Programming
  - Reading Sec 1.1

Warm-up Assignment

Warming up exercise

3 weeks

Due on 2/18

Problem

- Write a program foo that counts the occurance of strings in a file
  - % echo “Mike Ed Tony Ed Emily Ed” > file.txt
  - % ./foo file.txt Ed
  - 3

  - % echo “aaab” > file.txt
    % ./foo file.txt aa B
    2
    1

Why?

- Set up and get to know your programming environment
- Let us assess how prepared you are for the course
- Remember how to use
  - argc, argv
  - malloc, free
  - Pointers
  - fopen, fseek, ftell
  - …
- Test error handling
- Get to know the manpages !

Complete details …

- Available on Canvas tonight