ITEC 414
Network Programming

Dr. N. B. Gyan

Central University, Miotso. Ghana

Stand-alone Applications

- Most software developers are familiar with stand-alone application architectures, in which a single computer contains all the software components related to the graphical user interface (GUI), application service processing, and persistent data resources.
- For example, the stand-alone application architecture illustrated in Figure below consolidates the GUI, service processing, and persistent data resources within a single computer, with all peripherals attached directly.

Network Applications

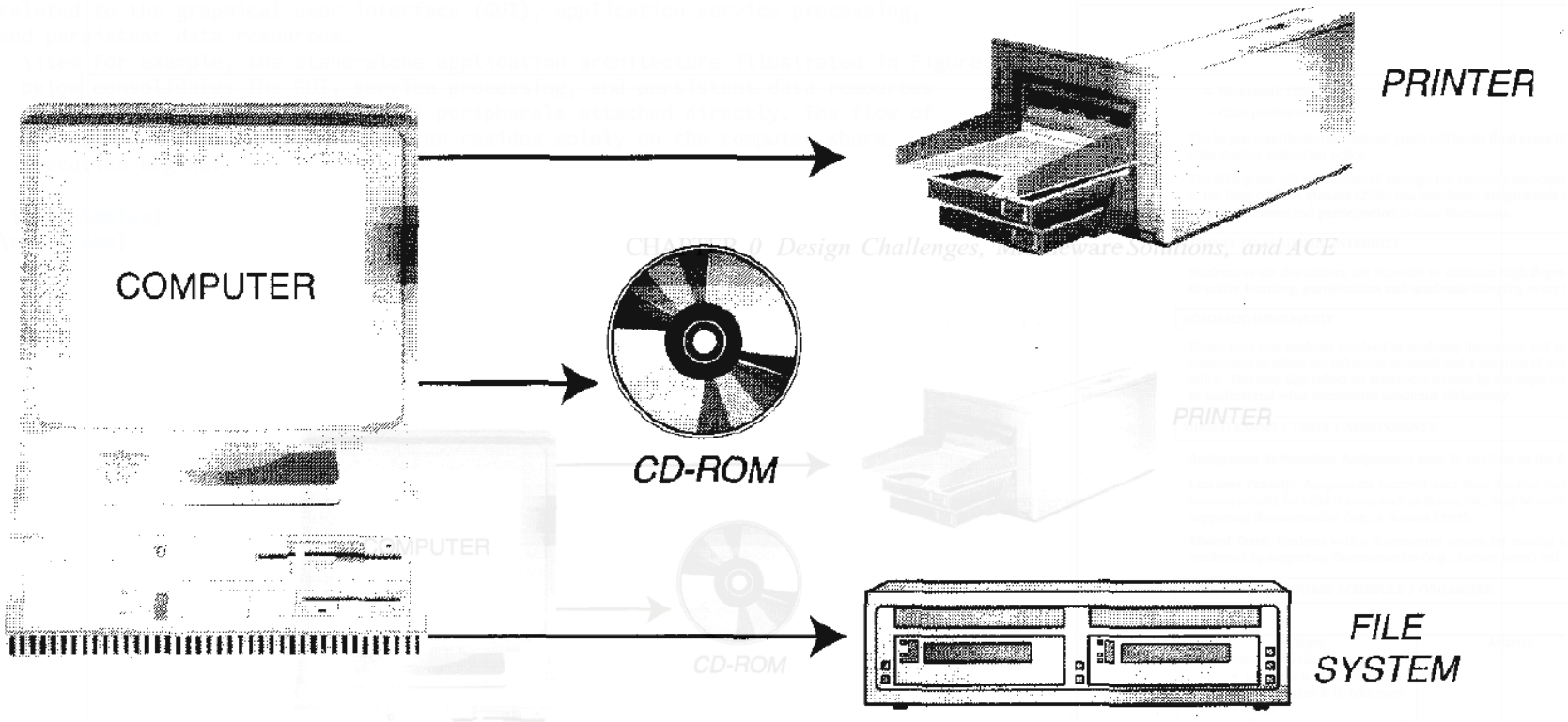


Figure 1

The flow of control in a stand-alone application resides solely on the computer where execution begins.

Network Applications

- In contrast, networked application architectures divide the application system into services that can be shared and reused by multiple applications (see Figure 2).
- To maximize effectiveness and usefulness, services are distributed among multiple computing devices connected by a network, as shown in Figure below.

Network Applications

- Common network services provided to clients in such environments include distributed naming, network file systems, routing table management, logging, printing, e-mail, remote login, file transfer, Web-based e-commerce services, payment processing, customer relationship management, help desk systems, MP3 exchange, streaming media, instant messaging, and community chat rooms.

Network Applications

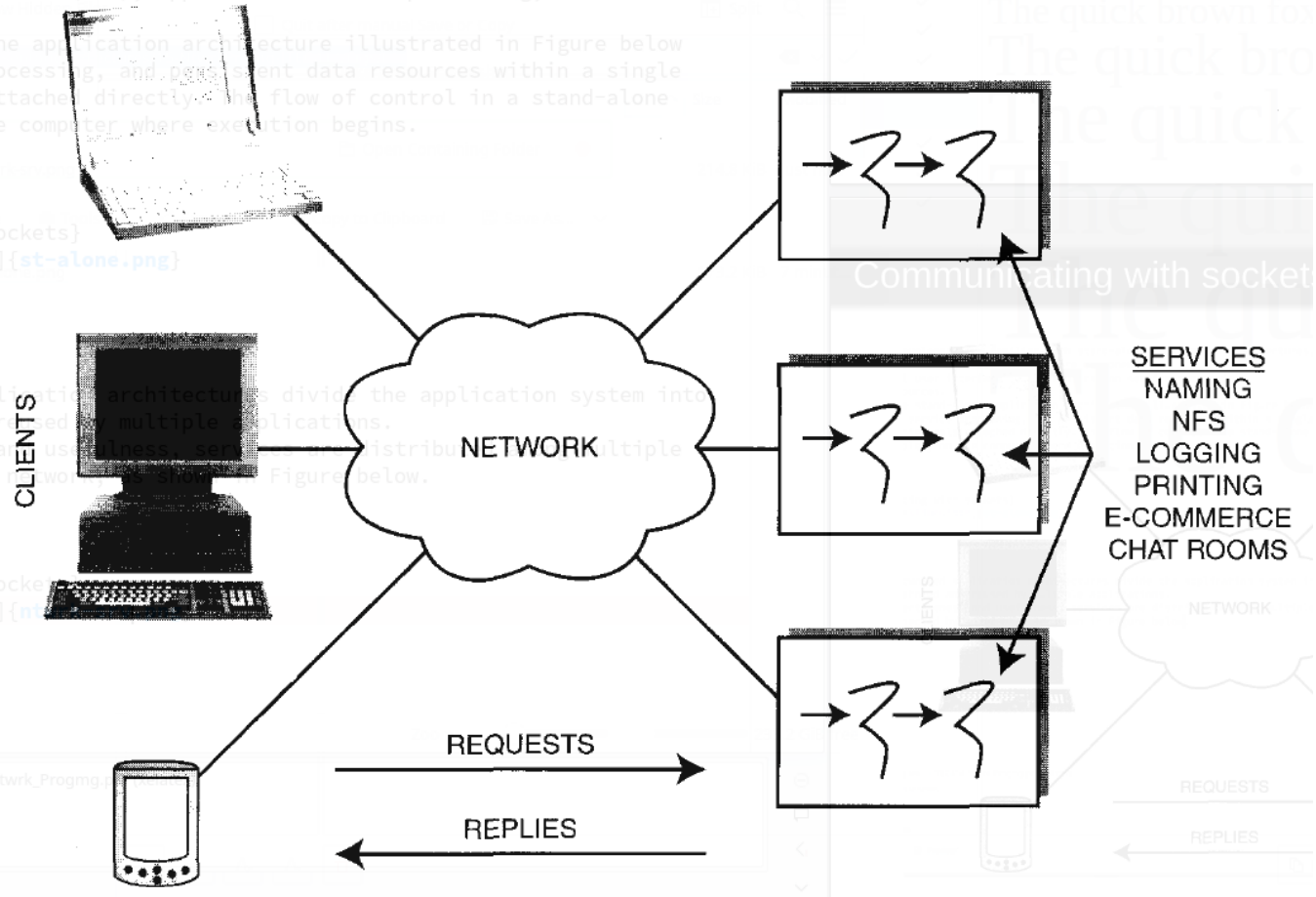


Figure 2

Network Applications

- The networked application architecture shown in Figure 2 partitions the interactive GUI, instruction processing, and persistent data resources among a number of independent hosts in a network.
- At run time, the flow of control in a networked application resides on one or more of the hosts.

Network Applications

- All the system components communicate cooperatively, transferring data and execution control between them as needed. Interoperability between separate components can be achieved as long as compatible communication protocols are used, even if the underlying networks, operating systems, hardware, and programming languages are heterogeneous

Benefits of Networked Applications

This delegation of networked application service responsibilities across multiple hosts can yield the following benefits:

- **Enhanced connectivity and collaboration** disseminates information rapidly to more potential users. This connectivity avoids the need for manual information transfer and duplicate entry.

Benefits of Networked Applications

- **Improved performance and scalability** allows system configurations to be changed readily and robustly to align computing resources with current and forecasted system demand.
- **Reduced costs** by allowing users and applications to share expensive peripherals and software, such as sophisticated database management systems.

Developing Networked Applications

Your job as a developer of networked applications is to understand the services that your applications will provide and the environment(s) available to provide them, and then

1. Design mechanisms that services will use to communicate, both between themselves and with clients.
2. Decide which architectures and service arrangements will make the most effective use of available environments.

Developing Networked Applications

3. Implement these solutions using techniques and tools that eliminate complexity and yield correct, extensible, high-performance, low-maintenance software to achieve your business's goals.

This is not easy to do.

Introduction to sockets

- One way of achieving the requirements above is by use of network **sockets**.
- A network socket is an endpoint of an inter-process communication flow across a computer network.
- Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Introduction to sockets

- Today, most communication between computers is based on the internet protocol; therefore most network sockets are internet sockets.
- To start a network socket communication, you make a call to the `socket()` system routine.
- It returns the socket descriptor, and you communicate through it using the specialized `send()` and `recv()` socket calls.

Introduction to sockets

- There are all kinds of sockets. There are DARPA Internet addresses (Internet Sockets), path names on a local node (**Unix Sockets**), **CCITT X.25 addresses** (X.25 Sockets that you can safely ignore), and probably many others depending on which Unix flavour you run.
- In this course we will focusing on the first: **Internet Sockets**.

Two Types of Internet Sockets

- One is “Stream Sockets”; the other is “Datagram Sockets”, which may hereafter be referred to as `SOCK_STREAM` and `SOCK_DGRAM`, respectively.
- Datagram sockets are sometimes called “*connectionless sockets*”.
- Stream sockets are reliable two-way connected communication streams. If you output two items into the socket in the order “1, 2”, they will arrive in the order “1, 2” at the opposite end.

Stream Sockets

What uses stream sockets?

- `telnet`
- Web browsers use the **Hypertext Transfer Protocol** (HTTP) which uses stream sockets to get pages.

How do stream sockets achieve this high level of data transmission quality?

- They use a protocol called “The Transmission Control Protocol”, otherwise known as “TCP”.

Stream Sockets

- TCP makes sure your data arrives sequentially and error-free.
- Often tied to IP to be TCP/IP. IP deals primarily with Internet routing and is not generally responsible for data integrity.

Datagram Sockets

What about Datagram sockets? What is the deal, here, anyway? Why are they unreliable?

- If you send a datagram, it may arrive. It *may* arrive out of order. If it arrives, the data within the packet will be error-free.
- Datagram sockets also use IP for routing, but they don't use TCP; they use the "User Datagram Protocol", or "UDP".

Datagram Sockets

Why are they called connectionless?

- because you don't have to maintain an open connection as you do with stream sockets. You just build a packet, slap an IP header on it with destination information, and send it out. No connection needed.
- They are generally used either when a TCP stack is unavailable or when a few dropped packets here and there don't mean the end of the Universe.

Datagram Sockets

Why are they called connectionless?

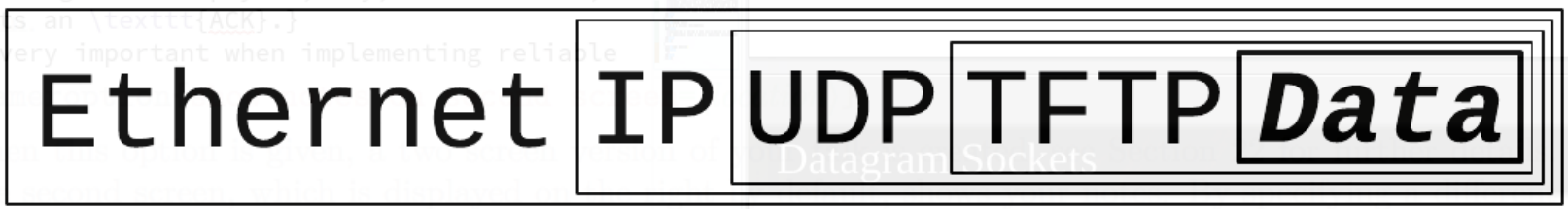
- Sample applications: `tftp` (trivial file transfer protocol, a little brother to FTP), `dhcpcd` (a DHCP client), multiplayer games, streaming audio, video conferencing, etc.

Datagram Sockets

Why would you use an unreliable underlying protocol?

- Two reasons: speed and speed. It's way faster to fire-and-forget than it is to keep track of what has arrived safely and make sure it's in order and all that.
- If you're sending chat messages, TCP is great; if you're sending 40 positional updates per second of the players in the world, maybe it doesn't matter so much if one or two get dropped, and UDP is a good choice.

Data Encapsulation



Data Encapsulation

- Basically, it says this: a packet is created, the packet is wrapped (“encapsulated”) in a header (and rarely a footer) by the first protocol (say, the TFTP protocol), then the whole thing (TFTP header included) is encapsulated again by the next protocol (say, UDP), then again by the next (IP), then again by the final protocol on the hardware (physical) layer (say, Ethernet).

Data Encapsulation

- When another computer receives the packet, the hardware strips the Ethernet header, the kernel strips the IP and UDP headers, the TFTP program strips the TFTP header, and it finally has the data.

Layered Network Model Revisited

- This Network Model describes a system of network functionality that has many advantages over other models.
- For instance, you can write sockets programs that are exactly the same without caring how the data is physically transmitted (serial, thin Ethernet, AUI, whatever) because programs on lower levels deal with it for you.

Layered Network Model Revisited

- The actual network hardware and topology is transparent to the socket programmer.
- It consists of the following (top to bottom): Application Layer $\rightarrow$ Presentation Layer $\rightarrow$ Session Layer $\rightarrow$ Transport Layer $\rightarrow$ Network Layer $\rightarrow$ Data Link Layer $\rightarrow$ Physical Layer

Layered Network Model Revisited

- The Physical Layer is the hardware (serial, Ethernet, etc.).
- The Application Layer is just about as far from the physical layer as you can imagine—it's the place where users interact with the network.

Layered Network Model Revisited

- A layered model more consistent with Unix might be:
  - Application Layer (`telnet`, `ftp`, etc.)
  - Host-to-Host Transport Layer (TCP, UDP)
  - Internet Layer (IP and routing) (Note: The router strips the packet to the IP header, consults its routing table, etc).
  - Network Access Layer (Ethernet, wi-fi, or whatever)

Layered Network Model

- Much of the practical use of these layers are often hidden from the programmer.
- All you have to do for stream sockets is `send()` the data out.
- All you have to do for datagram sockets is encapsulate the packet in the method of your choosing and `sendto()` it out.
- The kernel builds the Transport Layer and Internet Layer on for you and the hardware does the Network Access Layer.

Tools/software

For linux users you don’t have to do anything. :-)

For windows users:

→ Install Cygwin: `https://cygwin.com/`