CS3733: Operating Systems

Topics: Client-Server Model and Connection-Oriented Communication (USP Chapter 18)

Instructor: Dr. Tongping Liu

Last Lecture Overview
- High-level synchronization structure: Monitor
- Pthread mutex
- Conditional variables
- Barrier
- Threading Issues

Quiz
- Difference between semaphore and mutex lock

Binary Semaphore and Mutex Lock?
- Binary Semaphore:
  - No ownership

- Mutex lock
  - Only the owner of a lock can release a lock.
  - Priority inversion safety: potentially promote a task
  - Deletion safety: a task owning a lock can’t be deleted.

Outline
- Client-server model
  - Widely used in many applications: ftp, mail, http, ssh

- Communication models
  - Connectionless vs. connection-oriented

- Network basics: TCP/IP protocols

- Connection-oriented Client-Server communication
  - Sockets & usages: socket, bind, listen, accept, connect

- UICI: A Simple Client/Server Implementation
  - Simplified interface: u_open, u_accept, u_connect

Outline
- Client-server model
  - Widely used in many applications: ftp, mail, http, ssh

- Communication models
  - Connectionless vs. connection-oriented

- Network basics: TCP/IP protocols

- Connection-oriented Client-Server communication
  - Sockets & usages: socket, bind, listen, accept, connect

- UICI: A Simple Client/Server Implementation
  - Simplified interface: u_open, u_accept, u_connect
Client-Server Model

- Applications with two distinct parts
  - server vs. client
- Server: wait for requests from clients
- Client: makes requests for services from server
- Examples:
  - Web server vs. clients: http and FireFox, Chrome etc.
  - Mail server vs. clients
  - File server vs. client: nfs

Outline

- Client-server model
  - Widely used in many applications: ftp, mail, http, ssh
- Communication models
  - Connectionless vs. connection-oriented
- Network basics: TCP/IP protocols
- Connection-oriented Client-Server communication
  - Sockets & usages: socket, bind, listen, accept, connect
- UICI: A Simple Client/Server Implementation
  - Simplified interface: u_open, u_accept, u_connect

Client-Server over Internet

- Client/Server differentiate communication channels using port numbers
- Communication endpoint
  - Host address: specify which machine
  - Port number: specify which process
  - Standard servers with well-known port numbers
    - Mail: 25; http: 80; telnet: 23; ftp: 21
- Two approaches for C/S communications
  - Connectionless: server/client embeds info. in every request
  - connection-oriented: setup a connection before sending requests

Connection-oriented Client-Server Model

- Client sets up a connection to server’s well-known port number
  - After that, communicate over the private channel
- Pros and Cons?
  - Pros: Errors can be detected, if no response or overloaded
  - Cons: Initial setup overhead

Many Clients vs. One Server

- Clients use the same passive endpoint initially
  - How does server handle each request? sequentially?
    - Server are typically powerful to handle multiple requests

Handle Concurrent Requests

- Server can fork a child process to handle each request from different clients
  - Too costly, use threads!
  - Apache, nginx: hybrid multi-process multi-threaded. Why?
Outline

- Client-server model
  - Widely used in many applications: ftp, mail, http, ssh
- Communication models
  - Connectionless vs. connection-oriented
- Network basics: TCP/IP protocols
- Connection-oriented Client-Server communication
  - Sockets & usages: socket, bind, listen, accept, connect
- UICI: A Simple Client/Server Implementation
  - Simplified interface: u_open, u_accept, u_connect

Background: Network Communication

- Layered network models
- Ethernet: local area network
- Inter-network Protocols (IP)
  - Addressing and routing etc.
- TCP/UDP protocols
  - communication ports and sockets

Commonly Used Layer Structure

- Layered structure: layer n uses layer n-1 services
- Protocols: pairs of software in send/receive nodes
  - Specify the sequence of messages for transmission
  - Specify the format, contents and meanings of data in messages

Local Area Network (LAN)

- Shared medium: Carrier Sensing Multi-Access.
  - CSMA/CD: collision detection
- Every Ethernet interface has a unique 48 bit MAC address (a.k.a. hardware address).
  - Example: C0:B3:44:17:21:17
- Addresses are assigned to vendors by a central authority (IEEE to manufacturers)

What is IP?

- Network (or Inter-Network) layer protocol
  - packet delivery service (host-to-host).
  - translation between different data-link protocols (Ethernet).
- IP provides connectionless, unreliable delivery of IP datagram.
  - Connectionless: each datagram is independent.
  - Unreliable (best effort): no guarantee for datagrams to be delivered correctly or at all.

IP Packet

- header
  - 1 byte: IP address of source
  - 1 byte: IP address of destination
  - 2 bytes: up to 64 kilobytes

- VERS: version
  - 1 byte

- HL: header length
  - 1 byte

- Service: type of service
  - 1 byte

- Fragment Length: fragment length
  - 1 byte

- Datagram ID: unique identifier for datagram
  - 2 bytes

- Flag: fragmentation
  - 1 byte

- TTL: time to live
  - 1 byte

- Protocol: protocol type
  - 1 byte

- Header Checksum: checksum of header
  - 2 bytes

- Source Address

- Destination Address

- Options (if any)

- Data

- 1 byte

- 1 byte

- 1 byte

- 1 byte
Internet Addresses

- **Logical addresses**
  - 32 bits (IPv4, 4 billion, population in early 80s)
  - Includes a network ID and a host ID.
    - Network ID is assigned to an organization by the authority
    - Host IDs are assigned locally by a system administrator

- **Different classes of addresses**
  - **Class A**
    - 128 possible network IDs and over 4 million host IDs per network ID
  - **Class B**
    - 16K possible network IDs and 64K host IDs per network ID
  - **Class C**
    - Over 2 million possible network IDs and about 256 host IDs per network ID

---

**Internet Addresses (cont.)**

<table>
<thead>
<tr>
<th>Class A</th>
<th>Class B</th>
<th>Class C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0.0.0 to 127.255.255.255</td>
<td>128.0.0.0 to 191.255.255.255</td>
<td>192.0.0.0 to 223.255.255.255</td>
</tr>
</tbody>
</table>

---

IP Solves the Routing Problem

- Decide the route for each packet
  - Necessary in MANs and WANs
- Update knowledge of the network
  - Adaptive/dynamic routing is usually used: traffic patterns, topological changes
- Routing decision
  - Hop-by-hop, with period update and distribution of traffic data, e.g., the distance-vector, dynamic, distributed algorithm

---

TCP: Transmission Control Protocol

- **TCP is connection-oriented.**
  - 3-way handshake used for connection setup
  - Acknowledge each message (piggyback)

---

Outline

- Client-server model
  - Widely used in many applications: ftp, mail, http, ssh
- Communication models
  - Connectionless vs. connection-oriented
- Network basics: TCP/IP protocols
- Connection-oriented Client-Server communication
  - Sockets & usages: socket, bind, listen, accept, connect
  - UICI: A Simple Client/Server Implementation
    - Simplified interface: u_open, u_accept, u_connect
Socket Abstraction

- A socket must be bound to a local port
- Provide endpoints for communication between processes
- Socket pair - (local IP address, local port, foreign IP address, foreign port) uniquely identifies a communication channel

TCP Socket Primitives

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
<td>Create a new communication endpoint</td>
</tr>
<tr>
<td>Bind</td>
<td>Attach a local address to a socket</td>
</tr>
<tr>
<td>Listen</td>
<td>Announce willingness to accept connections</td>
</tr>
<tr>
<td>Accept</td>
<td>Block caller until a connection request arrives</td>
</tr>
<tr>
<td>Connect</td>
<td>Actively attempt to establish a connection</td>
</tr>
<tr>
<td>Send</td>
<td>Send some data over the connection</td>
</tr>
<tr>
<td>Recv</td>
<td>Receive some data over the connection</td>
</tr>
<tr>
<td>Close</td>
<td>Release the connection</td>
</tr>
</tbody>
</table>

POSIX Socket System Calls

```c
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
int bind(int s, const sockaddr *address, size_t address_len);
int listen(int s, int backlog);
int accept(int s, struct sockaddr *address, int *address_len);
int connect(int s, struct sockaddr *address, int *address_len);
```

**Domain Options:**
- AF_UNIX or AF_INET

**Type Options:**
- SOCK_STREAM for connection-oriented TCP
- SOCK_DGRAM for connectionless UDP

**Other Parameters:**
- `address`: contains info about the family, port, and machine
- `address_len`: size of the structure used for the address

POSIX Socket System Calls (cont.)

```c
int bind(int s, const struct sockaddr *address, size_t address_len);
  s: the file descriptor returned by socket
  address: contains info about the family, port, and machine
  sa_family_t sin_family;  // family
  in_port_t sin_port;  // 16 bits port number
  struct_in_addr sin_addr;  // has to be filled on client side
  address_len: size of the structure used for the address

int listen(int s, int backlog);
  s: the fd of the socket used to accept incoming requests
  backlog: number of outstanding requests (unaccepted)
```

POSIX Socket System Calls (cont.)

```c
int accept(int s, struct sockaddr *address, int *address_len);
  s: the fd of the socket to accept requests
  address: information of remote host making request
  address_len: size of the structure used for the address
```

POSIX Socket System Calls (cont.)

```c
int connect(int s, struct sockaddr *address, int *address_len);
  s: the fd of the socket to make requests
  address: information of remote host accepting request, needs to specify both IP/host_name and port number
  address_len: size of the structure used for the address
```

POSIX Socket System Calls (cont.)

- All functions return -1 and set errno
- Under Linux
  ```bash
c c -o server *.c
  ```
- Under Solaris, need to include socket library
  ```bash
c c -o server *.c -lsocket
  ```
Client-Server Using TCP Sockets

- Server side performs first with following actions
  - Step 1: Create socket
  - Step 2: bind server address & port (# known to clients)
  - Step 3: start listen for a connection request
  - Step 4: accept a connection request
    - create another socket for private communication with the clients
- Client: create a socket with (IP/host-name, port#), and then try to connect to the server

An Example: Client

- try to connect to the server

An Example: Server

- listen for connection request
- bind/associate socket with address, port
- accept a request
- enter into a loop
  - wait for a message from client
  - send a response back to the client (optional)

Outline

- Client-server model
  - Widely used in many applications: ftp, mail, http, ssh
- Communication models
  - Connectionless vs. connection-oriented
- Network basics: TCP/IP protocols
- Connection-oriented Client-Server communication
  - Sockets & usages: socket, bind, listen, accept, connect
- UICI: A Simple Client/Server Implementation
  - Simplified interface: u_open, u_accept, u_connect

Programming with UDP/IP sockets

1. Create the socket
2. Identify the socket (name it)
3. On the server, wait for a message
4. On the client, send a message
5. Send a response back to the client (optional)
6. Close the socket

No need to setup the channel
UICI: Universal Internet Communication Interface

- Abstract the essentials of network communication while hiding the details of programming
- not part of any UNIX standard

Outline

- Client-server model
  - Widely used in many applications: ftp, mail, http, ssh
- Communication models
  - Connectionless vs. connection-oriented
- Network basics: TCP/IP protocols
- Connection-oriented Client-Server communication
  - Sockets & usages: socket, bind, listen, accept, connect
- UICI: A Simple Client/Server Implementation
  - Simplified interface: u_open, u_accept, u_connect