Operating System Design

Threads
Computer Networks in 10 minutes!
Network Programming

Neda Nasiriani
Fall 2018

Activity Answer Continued!

Question 5

```
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine) (void *),
                   void *arg);
```

```
void * sum(void * a){
    int * val = (int *)a;
    int result = 0;
    for (int i = 1; i <= *val; i++){
        result += i;
    }
    printf("Running thread %lu value in thread %d \n",pthread_self(),result);
    return (NULL);

}
```

Question 6

```
int main(){
    int val = 100;
    int * a = &val;
    void * (*func)(void *) = &sum;

    pthread_t id;
    int err;
    err = pthread_create(&id, NULL, func,(void *)a);
    printf("thread create ret val %d\n",err);
    pthread_join(id, NULL);
    return 0;
}
```

Passing Multiple Arguments: Q7

```
struct input{
    int a;
    int b;
};
```

```
void * sum(void * a){
    struct input * val = (struct input *)a;
    int result = val->a + val->b;
    return ((void *) &result);

}
```

```
int main(){
    struct input args;
    void * retval;
    args.a = 10;
    args.b = 20;
    pthread_t id;
    int err;

    err = pthread_create(&id, NULL, sum,(void *)&args);
    printf("thread create ret val %d\n",err);
    pthread_join(id, &retval);
    printf("thread return value is %d\n",(*(int *)retval));
    return 0;
}
```

Passing Multiple Arguments

```
void * sum(void * a){
    int * val = (int *)a;
    int result = 0;
    for (int i = 1; i <= *val; i++){
        result += i;
    }
    printf("Running thread %lu value in thread %d \n",pthread_self(),result);
    return (NULL);

}
```

Good idea?!?
Why?

Return Value – Bad Practice!

```
struct input{
    int a;
    int b;
};
```

```
void * sum(void * a){
    struct input * val = (struct input *)a;
    int result = val->a + val->b;
    return ((void *) &result);

}
```

```
int main(){
    struct input args;
    void * retval;
    args.a = 10;
    args.b = 20;
    pthread_t id;
    int err;

    err = pthread_create(&id, NULL, sum,(void *)&args);
    printf("thread create ret val %d\n",err);
    pthread_join(id, &retval);
    printf("thread return value is %d\n",(*(int *)retval));
    return 0;
}
```

```
thread create ret val 0
thread return value is 32609
```

Return Value – Good Practice!

```
struct input{
    int a;
    int b;
    int result;
};
```

```
void * sum(void * a){
    struct input * val = (struct input *)a;
    val -> result = val->a + val->b;
    return ((void *) a);

}

int main(){
    struct input args;
    void * retval;
    args.a = 10;
    args.b = 20;
    pthread_t id;
    int err;
    err = pthread_create(&id, NULL, sum,(void *)&args);
    printf("thread create ret val %d\n",err);
    pthread_join(id, &retval);
    printf("thread return value is %d\n",((struct input *)retval)->result);
    return 0;
}
```

```
thread create ret val 0
thread return value is 30
```

Multiple Threads

```
struct input{
    int a;
    int b;
    int result;
};
```

```
#define NUM_THREADS 5
struct input args[NUM_THREADS];
pthread_t ids[NUM_THREADS];
```

```
void * sum(void * a){
    struct input * val = (struct input *)a;
    val -> result = val->a + val->b;
    printf("Thread with index %d is running now\n",val->a/10);
    return ((void *) a);

}
```

```
thread 0 return val 0
thread 1 return val 0
Thread with index 0 is running now
thread 2 return val 0
Thread with index 1 is running now
Thread with index 2 is running now
thread 3 return val 0
thread 4 return val 0
thread return value is 0
thread return value is 30
Thread with index 3 is running now
Thread with index 4 is running now
thread return value is 60
thread return value is 90
thread return value is 120
```

```
int main(){
    void * retval[NUM_THREADS];
    for (int i=0; i < NUM_THREADS; i++){
        args[i].a = i * 10;
        args[i].b = i * 20;

        int err;
        err = pthread_create(&ids[i], NULL, sum,(void *)&args[i]);
        printf("thread %d return val %d\n",i,err);
    }

    for (int i = 0; i < NUM_THREADS; i++){
        pthread_join(ids[i], &retval[i]);
        printf("thread return value is %d\n",((struct input *)retval[i])->result);
    }
    return 0;
}
```

Discussion: Activity Operating System Design

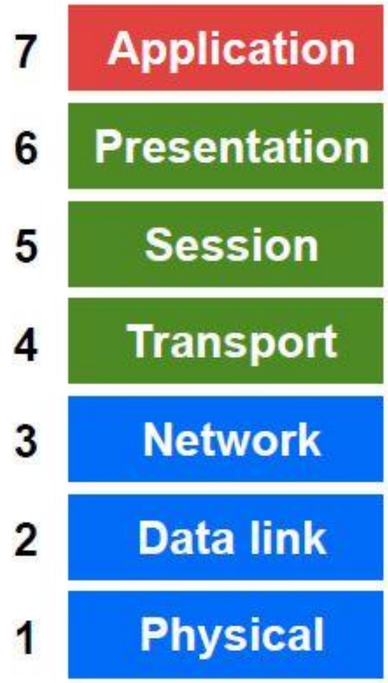
Quiz 03!

Computer Network

- How does two computers communicate?
- Internet is connecting millions of users all over the globe, who are connecting using different devices
- How does it work?!?
- The key to Internet success is its abstract design and protocols based on those design paradigms (guidelines)
- This architecture which is a layered architecture allows for communication between different nodes as long as they follow the same protocols
  - Remember your last lab on pipes
  - Pipes are an implementation of message passing
  - How did you send an integer and string using the pipe?
- The idea of an abstraction
  - To have a unifying model
  - To encapsulate this model in an object which provides an interface for other layers
  - To hide the details of how the object is implemented from the users of the object.

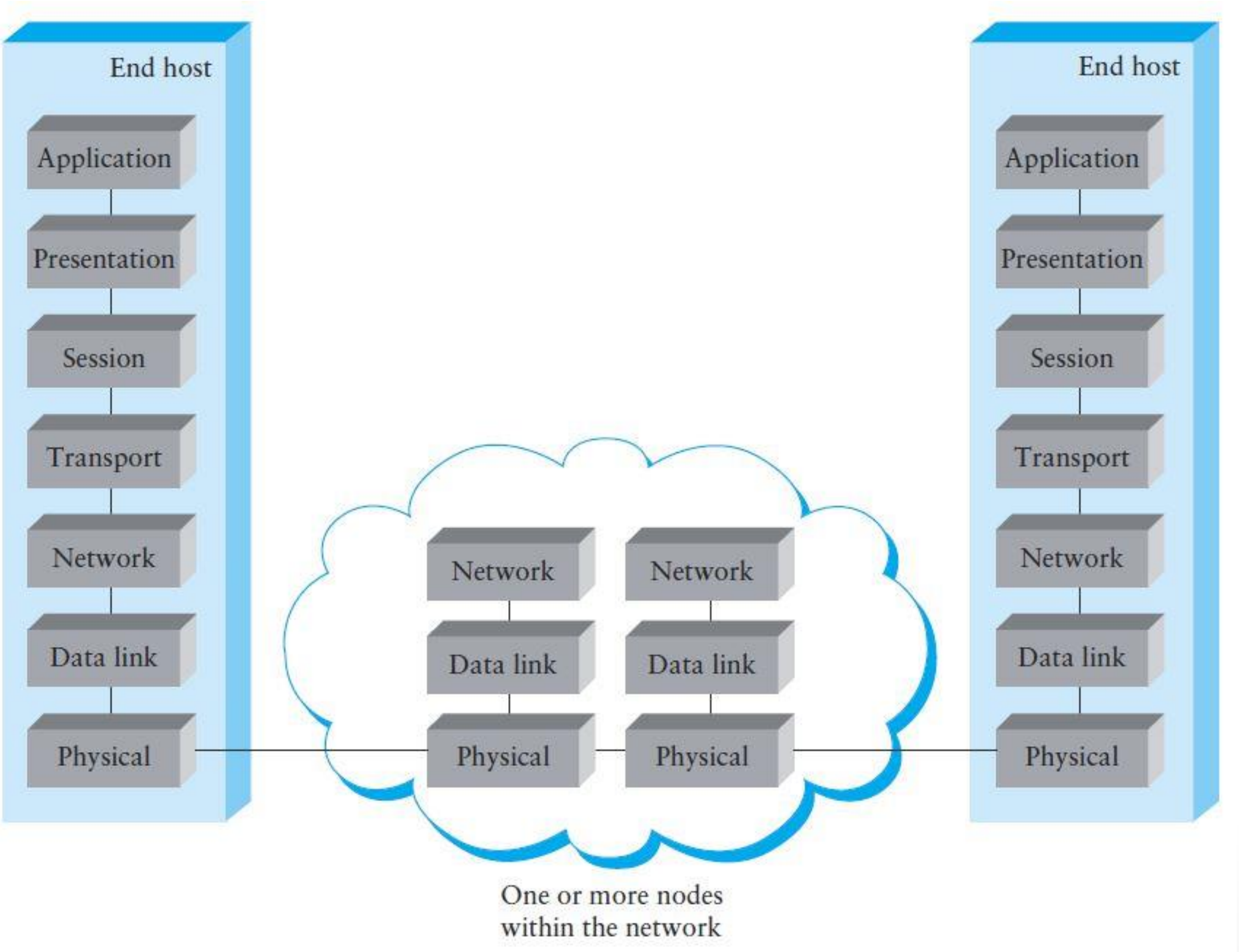
Open Systems Interconnection (OSI) Model

- OSI model is the standard proposed for computer networks
- Partitions the network functionality into seven layers
- Reference model for a protocol graph.
- Physical layer handles the transmission of *raw bits* over a communications link (wireless, fiber, coax)
- The data link layer then collects a stream of bits into a larger aggregate called a *frame*.
  - Network adaptors, along with device drivers running in the node's OS, typically implement the data link level.
- The network layer handles routing *packets* among nodes within a packet-switched network.
- The **Transport** layer then implements a process-to-process channel. Here, the unit of data exchanged is commonly called a *message* rather than a packet or a frame.



Hui Zhang, 15-441 Networking, Fall 2007, School of computer science, CMU.

OSI Model on Different Nodes

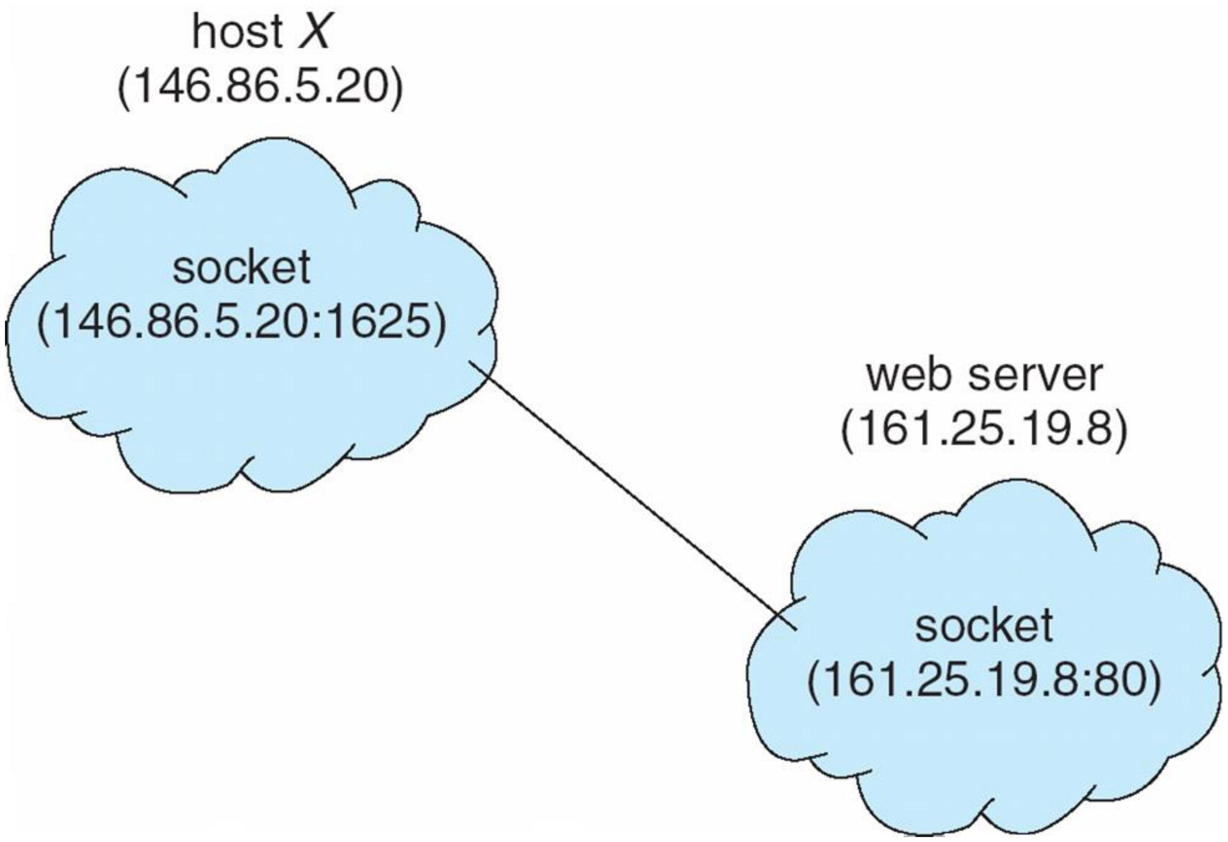


Computer Networks, Peterson and Davie

Sockets

- How can two processes on two different machines talk to each other on the web?
- A **socket** is defined as an endpoint for communication
- Concatenation of IP address and **port** – a number included at start of message packet to differentiate network services on a host
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- Communication consists between a pair of sockets
- All ports below 1024 are ***well known***, used for standard services
- Special IP address 127.0.0.1 (**loopback**) to refer to system on which process is running

Socket Communication



Connection Types

- Two types of connection (transport layer)
  - **Connection-oriented** (**TCP**)
  - **Connectionless** (**UDP**)

TCP Connections

- Service
  - OSI Transport Layer
- Reliable byte stream (interpreted by application)
- 16-bit port space allows multiple connections on a single host
- Connection-oriented

– Set up connection before communicating

– Tear down connection when done

TCP Service

- Reliable Data Transfer

– Guarantees delivery of all data

– Exactly once if no catastrophic failures

• Sequenced Data Transfer

– Guarantees in-order delivery of data

– If A sends M1 followed by M2 to B, B never receives M2 before M1

• Regulated Data Flow

– Monitors network and adjusts transmission appropriately

– Prevents senders from wasting bandwidth

– Reduces global congestion problems

• Data Transmission

– Full-Duplex byte stream

www.cs.uluc.edu/class/fa07/cs438

TCP Connection Establishment

- Connection oriented (streams)
  - sd = socket(PF_INET, SOCK_STREAM, 0);
- For the internet (PF_INET) this corresponds to TCP
- socket() returns a socket descriptor, an int similar to a file descriptor
- Use connect() on a socket that was previously created using socket():
- err = connect(int sd, struct sockaddr*, socklen_t addrlen);
- •Remote address and port are in

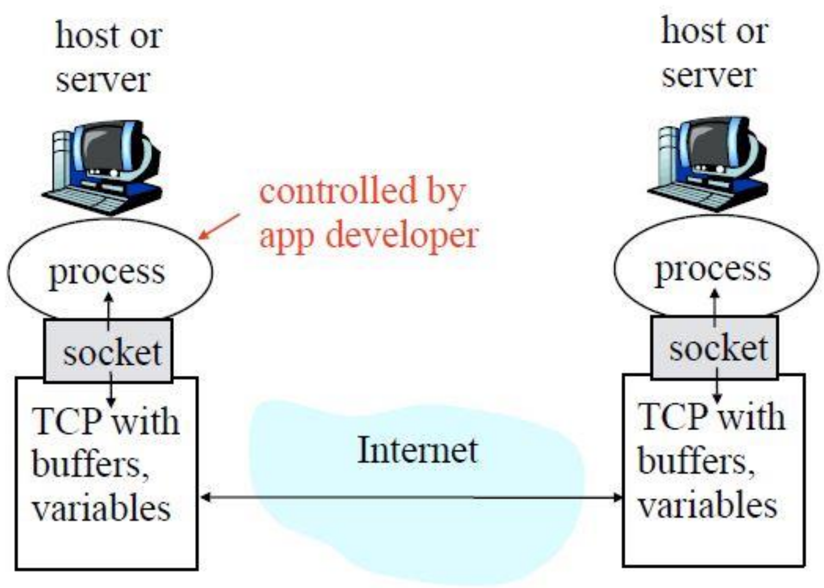
struct sockaddr:

```
    struct sockaddr_in {
    u_short sa_family;
    u_short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};
```

www.cs.uluc.edu/class/fa07/cs438

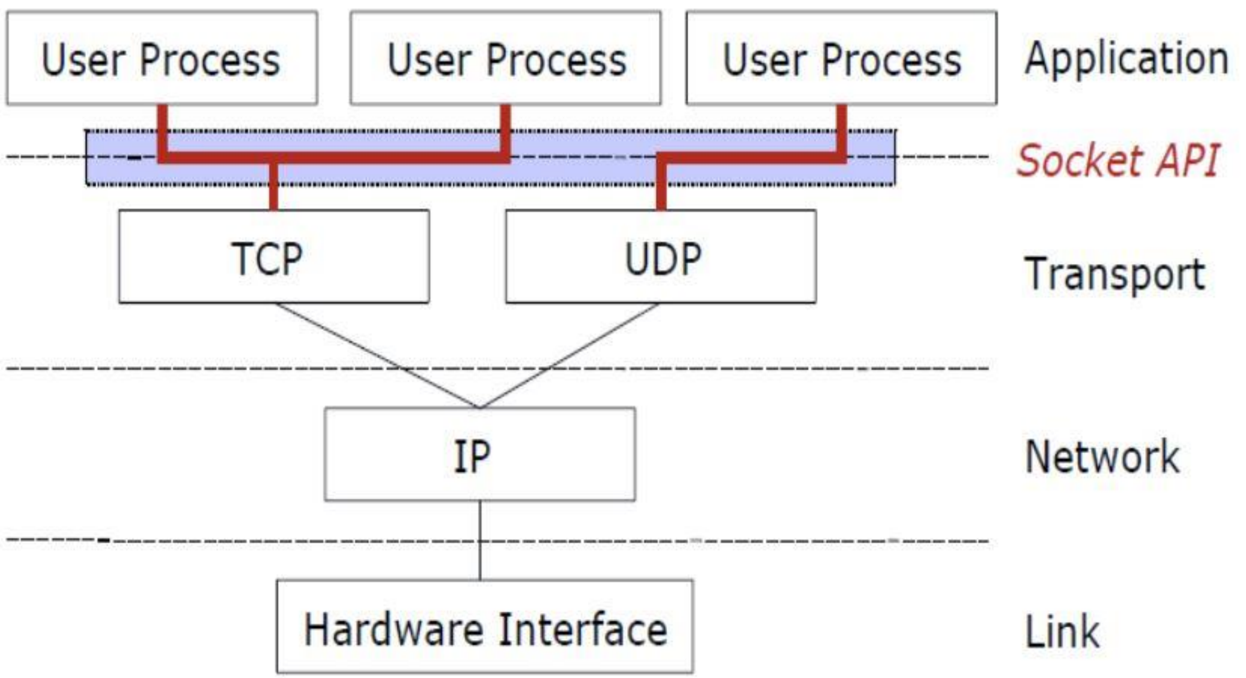
Sample TCP communication

- Transport Control Protocol (TCP)



www.cs.uluc.edu/class/fa07/cs438

TCP connection from OSI P.O.V.



www.cs.uluc.edu/class/fa07/cs438

TCP: Client

- socket() create the socket descriptor
- connect() connect to the remote server.
- read(),write() communicate with the server
- close() end communication by closing socket descriptor

TCP: Server

- socket() create the socket descriptor
- bind() associate the local address
- listen() wait for incoming connections from clients
- accept() accept incoming connection
- read(),write() communicate with client
- close() close the socket descriptor