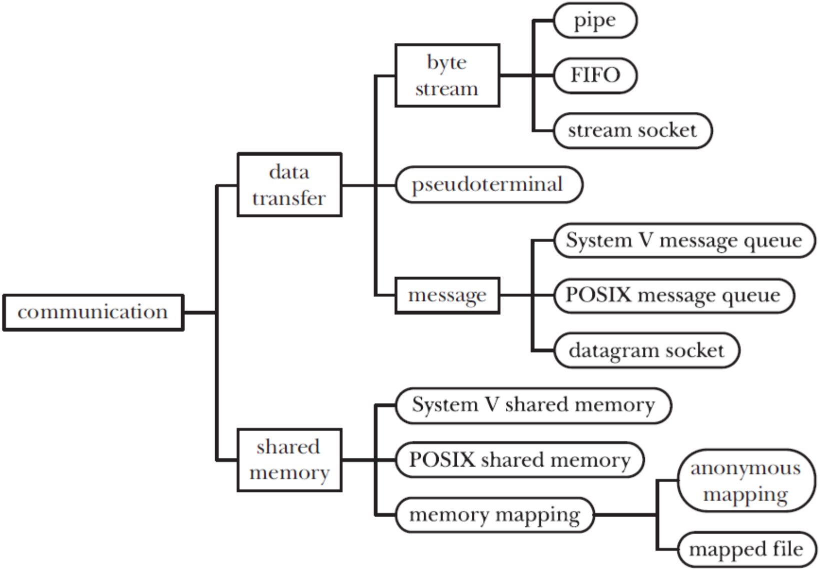
Processes (Part III)

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

Inter-Process Communication (IPC)

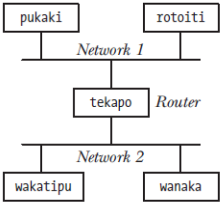

communication
data transfer
byte stream
pipe
FIFO
stream socket
pseudoterminal
message
System V message queue
POSIX message queue
datagram socket
shared memory
System V shared memory
POSIX shared memory
memory mapping
anonymous mapping
mapped file

Socket

Let's First Review The Basic Concepts of TCP/IP

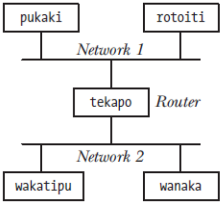
Internetworking

- An internetwork (internet (with a lowercase i)) is a network of computer networks.



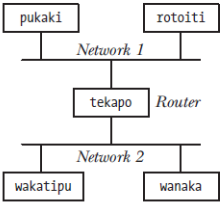
Internetworking

- An internetwork (internet (with a lowercase i)) is a network of computer networks.
- Subnetwork refers to one of the networks composing an internet.



Internetworking

- An internetwork (internet (with a lowercase i)) is a network of computer networks.
- Subnetwork refers to one of the networks composing an internet.
- An internet aims to hide the details of different physical networks, to present a unified network architecture.



The Internet

- TCP/IP has become the dominant protocol for the internetworking.

The Internet

- TCP/IP has become the dominant protocol for the internetworking.
- The Internet (with an uppercase I) refers to the TCP/IP internet that connects millions of computers globally.

The Internet

- TCP/IP has become the dominant protocol for the internetworking.
- The Internet (with an uppercase I) refers to the TCP/IP internet that connects millions of computers globally.
- The first widespread implementation of TCP/IP appeared with 4.2BSD in 1983.

Networking Protocols and Layers

- A networking protocol is a set of rules defining how information is to be transmitted across a network.

Networking Protocols and Layers

- A networking protocol is a set of rules defining how information is to be transmitted across a network.
- Networking protocols are generally organized as a series of layers.

Networking Protocols and Layers

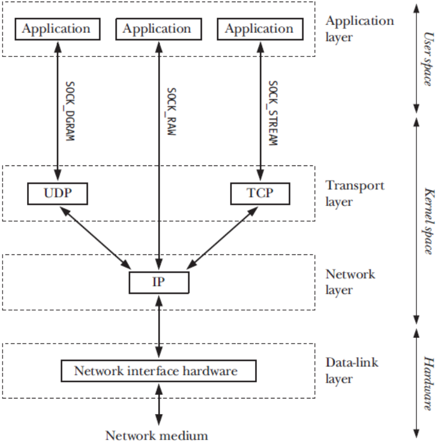
- A networking protocol is a set of rules defining how information is to be transmitted across a network.
- Networking protocols are generally organized as a series of layers.
- Each layer building on the layer below it to add features that are made available to higher layers.

Networking Protocols and Layers

- A networking protocol is a set of rules defining how information is to be transmitted across a network.
- Networking protocols are generally organized as a series of layers.
- Each layer building on the layer below it to add features that are made available to higher layers.
- Transparency: each protocol layer shields higher layers from the operation and complexity of lower layers.

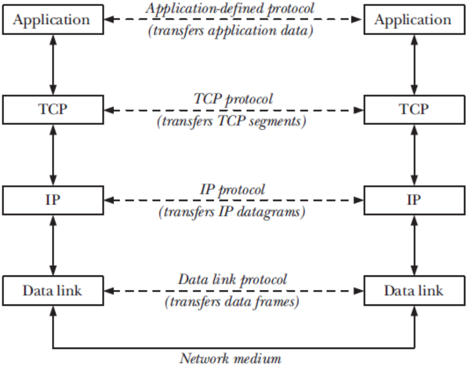
TCP/IP Protocol Suite

- The TCP/IP protocol suite is a layered networking protocol.



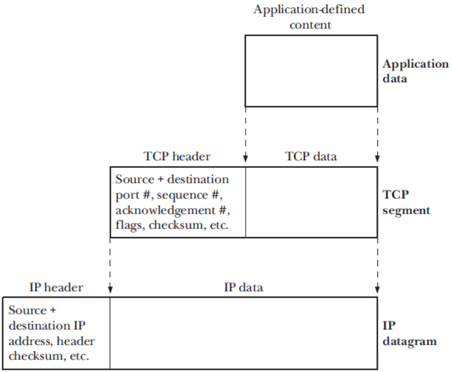
TCP/IP Protocol Layers

- Data-Link layer
- Network layer (IP)
- Transport layer (TCP, UDP)
- Application



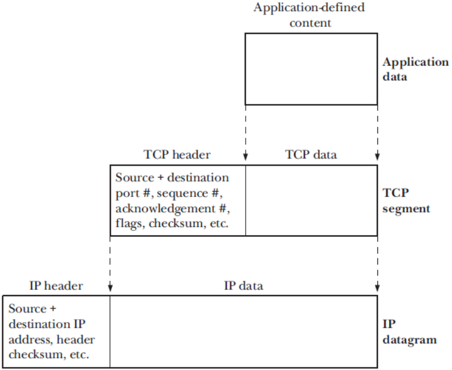
Encapsulation

- Encapsulation: the information passed from a higher layer to a lower layer is treated as opaque data by the lower layer.
  - The lower layer does not interpret information from the upper layer.



Encapsulation

- Encapsulation: the information passed from a higher layer to a lower layer is treated as opaque data by the lower layer.
  - The lower layer does not interpret information from the upper layer.
- When data is passed up from a lower layer to a higher layer, a converse unpacking process takes place.



Data-Link Layer (1/3)

- It is concerned with transferring data across a physical link in a network.

Data-Link Layer (1/3)

- It is concerned with transferring data across a physical link in a network.
- It consists of the device driver and the hardware interface (network card) to the underlying physical medium, e.g., fiber-optic cable.

Data-Link Layer (2/3)

- The data-link layer encapsulates datagrams from the network layer into units, called frames.

Data-Link Layer (2/3)

- The data-link layer encapsulates datagrams from the network layer into units, called frames.
- It also adds each frame a header containing the destination address and frame size.

Data-Link Layer (2/3)

- The data-link layer encapsulates datagrams from the network layer into units, called frames.
- It also adds each frame a header containing the destination address and frame size.
- The data-link layer transmits the frames across the physical link and handles acknowledgements from the receiver.

Data-Link Layer (3/3)

- From an application-programming point of view, we can generally ignore the data-link layer, since all communication details are handled in the driver and hardware.

Data-Link Layer (3/3)

- From an application-programming point of view, we can generally ignore the data-link layer, since all communication details are handled in the driver and hardware.

- Maximum Transmission Unit (MTU): the upper limit that the layer places on the size of a frame.
  - data-link layers have different MTUs.

```
netstat -i
```

Network Layer (1/4)

- It is concerned with delivering data from the source host to the destination host.

Network Layer (1/4)

- It is concerned with delivering data from the source host to the destination host.

- It tasks include:
  - Breaking data into fragments small enough for transmission via the data-link layer.
  - Routing data across the internet.
  - Providing services to the transport layer.

Network Layer (1/4)

- It is concerned with delivering data from the source host to the destination host.

- It tasks include:
  - Breaking data into fragments small enough for transmission via the data-link layer.
  - Routing data across the internet.
  - Providing services to the transport layer.

- In the TCP/IP protocol suite, the principal protocol in the network layer is IP.

Network Layer (2/4)

- IP transmits data in the form of **packets**.

Network Layer (2/4)

- IP transmits data in the form of packets.

- Each packet sent between two hosts travels independently across the network.

Network Layer (2/4)

- IP transmits data in the form of packets.

- Each packet sent between two hosts travels independently across the network.

- An IP packet includes a header that contains the address of the source and target hosts.

Network Layer (3/4)

- IP is a connectionless protocol: it does not provide a virtual circuit connecting two hosts.

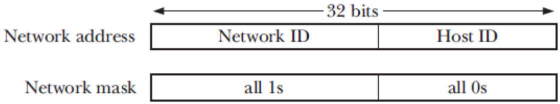
Network Layer (3/4)

- IP is a connectionless protocol: it does not provide a virtual circuit connecting two hosts.

- IP is an unreliable protocol: it makes a best effort to transmit datagrams from the sender to the receiver, but it does not guarantee:
  - that packets will arrive in the order they were transmitted,
  - that they will not be duplicated,
  - that they will arrive at all.

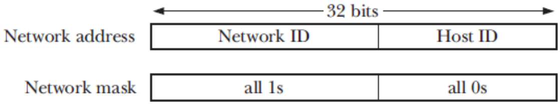
Network Layer (4/4)

- An IP address consists of two parts:
  - Network ID: specifies the network on which a host resides.
  - Host ID: identifies the host within that network.



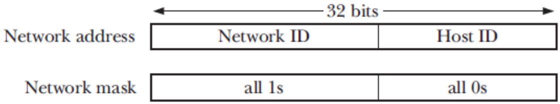
Network Layer (4/4)

- An IP address consists of two parts:
  - Network ID: specifies the network on which a host resides.
  - Host ID: identifies the host within that network.
- An IPv4 address consists of 32 bits: `204.152.189.0/24`
  - loopback `127.0.0.1` refers to system on which process is running.



Network Layer (4/4)

- An IP address consists of two parts:
  - Network ID: specifies the network on which a host resides.
  - Host ID: identifies the host within that network.
- An IPv4 address consists of 32 bits: `204.152.189.0/24`
  - loopback `127.0.0.1` refers to system on which process is running.
- Network mask: a sequence of 1s in the leftmost bits, followed by a sequence of 0s
  - The 1s indicate which part of the address contains the assigned network ID.
  - The 0s indicate which part of the address is available to assign as host IDs.



Transport Layer (1/5)

- Transport protocol provides an end-to-end communication service to applications residing on different hosts.

Transport Layer (1/5)

- Transport protocol provides an end-to-end communication service to applications residing on different hosts.

- Two widely used transport-layer protocols in the TCP/IP suite:
  - User Datagram Protocol (UDP): the protocol used for datagram sockets.
  - Transmission Control Protocol (TCP): the protocol used for stream sockets.

Transport Layer (2/5)

- Port: a method of differentiating the applications on a host.
  - 16-bit number

Transport Layer (2/5)

- Port: a method of differentiating the applications on a host.
  - 16-bit number
  - All ports below 1024 are well known, used for standard services, e.g., http: 80, ssh: 22.

Transport Layer (2/5)

- Port: a method of differentiating the applications on a host.
  - 16-bit number
  - All ports below 1024 are well known, used for standard services, e.g., http: 80, ssh: 22.
  - Shown as `192.168.1.1:8080`.

Transport Layer (3/5)

- UDP, like IP, is connectionless and unreliable.

Transport Layer (3/5)

- UDP, like IP, is connectionless and unreliable.
- If an application layered on top of UDP requires reliability, then this must be implemented within the application.

Transport Layer (3/5)

- UDP, like IP, is connectionless and unreliable.
- If an application layered on top of UDP requires reliability, then this must be implemented within the application.
- UDP adds just two features to IP:
  - Port number
  - Data checksum to allow the detection of errors in the transmitted data.

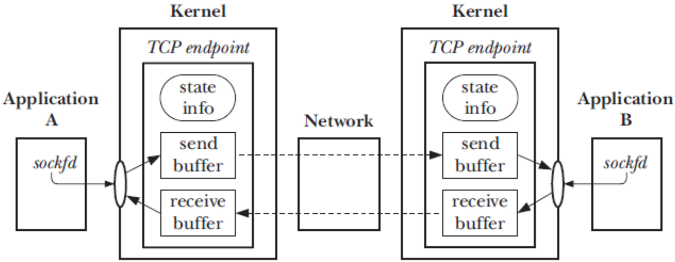
[http://www.tamos.net/∼rhay/overhead/ip-packet-overhead.htm]

Transport Layer (4/5)

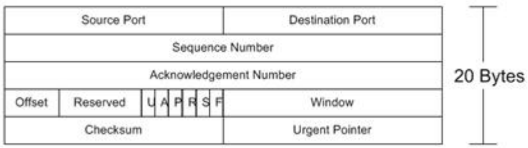
- TCP provides a reliable, connection-oriented, bidirectional, byte-stream communication channel between two endpoints.

Transport Layer (4/5)

- TCP provides a reliable, connection-oriented, bidirectional, byte-stream communication channel between two endpoints.
- Before communication can commence, TCP establishes a communication channel between the two endpoints.



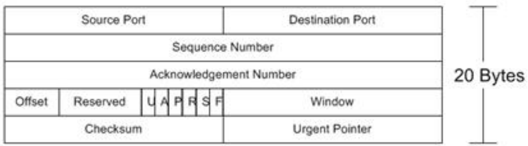
- In TCP, data is broken into segments: each is transmitted in a single IP packet.



[http://www.tamos.net/∼rhay/overhead/ip-packet-overhead.htm]

Transport Layer (5/5)

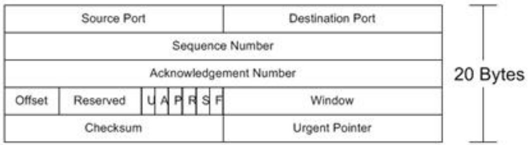
- In TCP, data is broken into segments: each is transmitted in a single IP packet.
- When a destination receives a TCP segment, it sends an ack. to the sender, informing weather it received the segment correctly or not.



[http://www.tamos.net/~rhay/overhead/ip-packet-overhead.htm]

Transport Layer (5/5)

- In TCP, data is broken into segments: each is transmitted in a single IP packet.
- When a destination receives a TCP segment, it sends an ack. to the sender, informing weather it received the segment correctly or not.
- Other features of TCP:
  - Sequencing
  - Flow control
  - Congestion control



[http://www.tamos.net/~rhay/overhead/ip-packet-overhead.htm]

OK, Let's Back to Socket

Socket

- A socket is defined as an endpoint for communication.

Socket

- A socket is defined as an endpoint for communication.
- A typical client-server scenario:
  - Each process creates a socket: both processes require one.
  - The server binds its socket to a well-known address (name) so that clients can locate it.

Creating a Socket

- `socket()` creates a new socket.

```
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

Socket Domains

- The UNIX domain (AF_UNIX)
  - Communication between processes on the same host (within the kernel).
  - Address format: path name.

```
int socket(int domain, int type, int protocol);
```

Socket Domains

- The UNIX domain (`AF_UNIX`)
  - Communication between processes on the same host (within the kernel).
  - Address format: path name.

- The IPV4 domain (`AF_INET`)
  - Communication between processes running on hosts connected via an IPv4 network.
  - Address format: 32-bit IPv4 address + 16-bit port number.

```c
int socket(int domain, int type, int protocol);
```

Socket Types

- Stream sockets (SOCK_STREAM)
  - It provides a reliable, bidirectional, byte-stream communication channel.
  - Called connection-oriented.

```
int socket(int domain, int type, int protocol);
```

Socket Types

- Stream sockets (SOCK_STREAM)
  - It provides a reliable, bidirectional, byte-stream communication channel.
  - Called connection-oriented.
- Datagram sockets (SOCK_DGRAM)
  - Allow data to be exchanged in the form of messages called datagrams.
  - Called connectionless.

```
int socket(int domain, int type, int protocol);
```

Binding a Socket to an Address

- `bind()` binds a socket to an address.

```
#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Listening for Incoming Connections

- `listen()` marks the stream socket passive.
- The socket will subsequently be used to accept connections from other (active) sockets.

```
#include <sys/socket.h>

int listen(int sockfd, int backlog);
```

Accepting a Connection

- accept() accepts an incoming connection on the listening stream socket.
- If there are no pending connections when accept() is called, the call blocks until a connection request arrives.

```
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

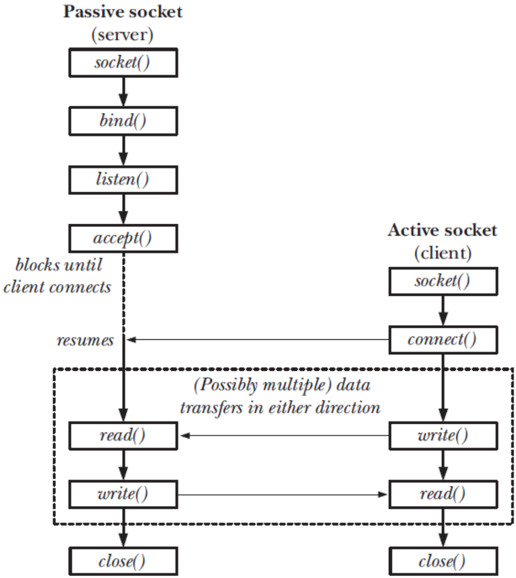
Connecting to a Peer Socket

- `connect()` connects the active socket to the listening socket whose address is specified by `addr` and `addrlen`.

```
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Stream Sockets



Producer-Consumer vi Stream Socket (1/2)

- Producer (Server)

```
int sockfd, connfd;
struct sockaddr_in serv_addr, cli_addr;
socklen_t cli_len;
char buffer[256];

bzero(&serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(32000);

sockfd = socket(AF_INET, SOCK_STREAM, 0);
bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr));
listen(sockfd, 5);

cli_len = sizeof(cli_addr);
connfd = accept(sockfd, (struct sockaddr *)&cli_addr, &cli_len);
read(connfd, buffer, 255);
write(connfd, "I got your message", 18);
```

Producer-Consumer vi Stream Socket (2/2)

- Consumer (Client)

```
int sockfd, connfd;
struct sockaddr_in serv_addr, cli_addr;
socklen_t clilen;
char *buf = "hello";
char rec_buf[256];

bzero(&serv_addr,sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr("x.x.x.x");
serv_addr.sin_port = htons(32000);


sockfd = socket(AF_INET, SOCK_STREAM, 0);
connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr));
write(sockfd, buf, strlen(buf));
read(sockfd, recv_buf, 255);
```

Internet Socket Addresses

- An IPv4 socket address is stored in a `sockaddr_in` structure, defined in `<netinet/in.h>`.

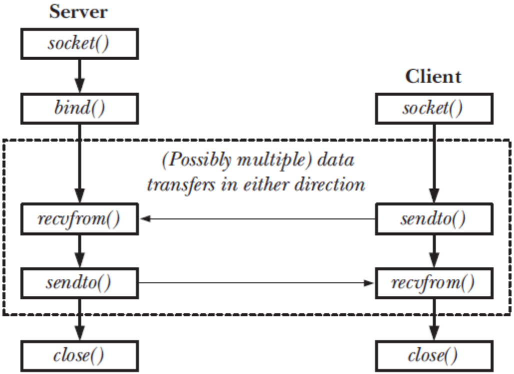
```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

struct sockaddr {
  sa_family_t sa_family; // Address family (AF_* constant)
  char sa_data[14]; // Socket address
};

struct sockaddr_in { // IPv4 socket address
  sa_family_t sin_family; // Address family (AF_INET)
  in_port_t sin_port; // Port number
  struct in_addr sin_addr; // IPv4 address
  unsigned char _pad[X]; // Pad to size of 'sockaddr' structure (16 bytes)
};

struct in_addr { // IPv4 4-byte address
  in_addr_t s_addr; // Unsigned 32-bit integer
};
```

Datagram Sockets



Exchanging Datagrams

- recvfrom() and sendto() receive and send datagrams on a datagram socket.

```
#include <sys/socket.h>

ssize_t sendto(int sockfd, const void *buffer, size_t length, int flags,
  const struct sockaddr *dest_addr, socklen_t addrlen);

ssize_t recvfrom(int sockfd, void *buffer, size_t length, int flags,
  struct sockaddr *src_addr, socklen_t *addrlen);
```

Producer-Consumer vi Datagram Socket (1/2)

- Producer (Server)

```
int sockfd, n;
struct sockaddr_in serv_addr, cli_addr;
socklen_t cli_len;
char buf[256];

bzero(&serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(32000);

sockfd = socket(AF_INET, SOCK_DGRAM, 0);
bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr));

cli_len = sizeof(cli_addr);
n = recvfrom(sockfd, buf, 255, 0, (struct sockaddr *)&cli_addr, &cli_len);
sendto(sockfd, buf, n, 0, (struct sockaddr *)&cli_addr,sizeof(cli_addr));
```

Producer-Consumer vi Datagram Socket (2/2)

- Consumer (Client)

```
int sockfd, connfd;
struct sockaddr_in serv_addr, cli_addr;
socklen_t clilen;
char *buf = "hello";
char recv_buf[256];

bzero(&serv_addr,sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr("x.x.x.x");
serv_addr.sin_port = htons(32000);

sockfd = socket(AF_INET, SOCK_DGRAM, 0);
sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr *)&serv_addr,
  sizeof(serv_addr));
recvfrom(sockfd, recv_buf, 255, 0, NULL, NULL);
```

Signals

Signals (1/2)

- Signals are software interrupts to notify a process that a particular event has occurred.

Signals (1/2)

- Signals are software interrupts to notify a process that a particular event has occurred.
- These events can originate from outside the system, e.g., by pressing Ctrl-C, or when a process executes code that divides by zero.

Signals (1/2)

- Signals are software interrupts to notify a process that a particular event has occurred.
- These events can originate from outside the system, e.g., by pressing Ctrl-C, or when a process executes code that divides by zero.
- As a primitive form of IPC, one process can also send a signal to another process.

```
ps -aux | grep acrobat
amir      7302  0.0  0.0   ...

kill -9 7302
```

Signals (2/2)

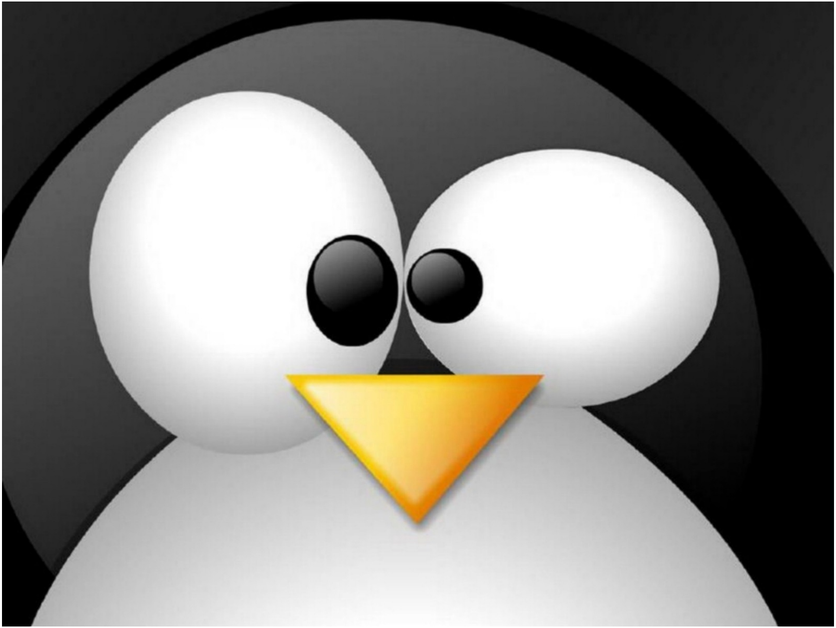
- A signal handler is used to process signals.
  1. Signal is generated by particular event.
  2. Signal is delivered to a process.
  3. Signal is handled by one of two signal handlers: default or user-defined.

Signals (2/2)

- A signal handler is used to process signals.
  1. Signal is generated by particular event.
  2. Signal is delivered to a process.
  3. Signal is handled by one of two signal handlers: default or user-defined.

- Every signal has default handler that kernel runs when handling signal

- User-defined signal handler can override default.



Signal Management

- signal() removes the current action taken on receipt of the signal signo and instead handles the signal with the signal handler specified by handler.

```
#include <signal.h>

typedef void (*sighandler_t)(int);

sighandler_t signal(int signo, sighandler_t handler);
```

Waiting for a Signal

- pause() puts a process to sleep until it receives a signal.

```
#include <unistd.h>

int pause(void);
```

Signal Example

```
// handler for SIGINT
static void sigint_handler(int signo) {
  printf("Caught SIGINT!\n");
  exit(0);
}

int main(void) {
  // Register sigint_handler as our signal handler for SIGINT.
  if (signal(SIGINT, sigint_handler) == SIG_ERR)
    exit(1);

  pause();

  return 0;
}
```

Summary

Summary

- TCP-IP protocol layers: data-link, network, transport, application

Summary

- TCP-IP protocol layers: data-link, network, transport, application
- Data-link: network card

Summary

- TCP-IP protocol layers: data-link, network, transport, application
- Data-link: network card
- Network layer: routing, IP, 32-bit address, 16-bit port

Summary

- TCP-IP protocol layers: data-link, network, transport, application
- Data-link: network card
- Network layer: routing, IP, 32-bit address, 16-bit port
- Transport layer: TCP (stream, connection-oriented), UDP (datagram, connectionless)

Summary

- TCP-IP protocol layers: data-link, network, transport, application
- Data-link: network card
- Network layer: routing, IP, 32-bit address, 16-bit port
- Transport layer: TCP (stream, connection-oriented), UDP (datagram, connectionless)
- Sockets
- Signal: software interrupts

Questions?