.UBAfiuba
FACULTAD DE INGENIERÍA

Técnicas de Programación Concurrente I

Concurrencia Distribuida - Parte I

Ing. Pablo A. Deymonnaz
pdeymon@fi.uba.ar

Facultad de Ingeniería
Universidad de Buenos Aires

Índice

Exclusión Mutua: Algoritmo Centralizado

1. Un proceso es elegido coordinador
2. Cuando un proceso quiere entrar a la SC, envía un mensaje al coordinador
3. Si no hay ningún proceso en la SC, el coordinador envía OK; si hay, el coordinador no envía respuesta hasta que se libere la SC

Exclusión Mutua: Algoritmo Distribuido

Cuando un proceso quiere entrar en una sección crítica, construye un mensaje con el nombre de la sección crítica, el número de proceso y el timestamp. Al recibir el mensaje:

1. Si no está en la CS y no quiere entrar, envía OK
2. Si está en la CS, no responde y encola el mensaje. Cuando sale de la SC, envía OK
3. Si quiere entrar en la CS, compara el timestamp y gana el menor

Exclusión Mutua: Algoritmo Token Ring

- Se conforma un anillo mediante conexiones punto a punto
- Al inicializar, el proceso 0 recibe un token que va circulando por el anillo
- Sólo el proceso que tiene el token puede entrar a la SC
- Cuando el proceso sale de la SC, continua circulando el token
- El proceso no puede entrar a otra SC con el mismo token

Índice

Introducción a Sockets (I)

- Permiten la comunicación entre dos procesos diferentes
  - En la misma máquina
  - En dos máquinas diferentes
- Se usan en aplicaciones que implementan el modelo cliente - servidor:
  - Cliente: es activo porque inicia la interacción con el servidor
  - Servidor: es pasivo porque espera recibir las peticiones de los clientes

Arquitectura cliente - servidor

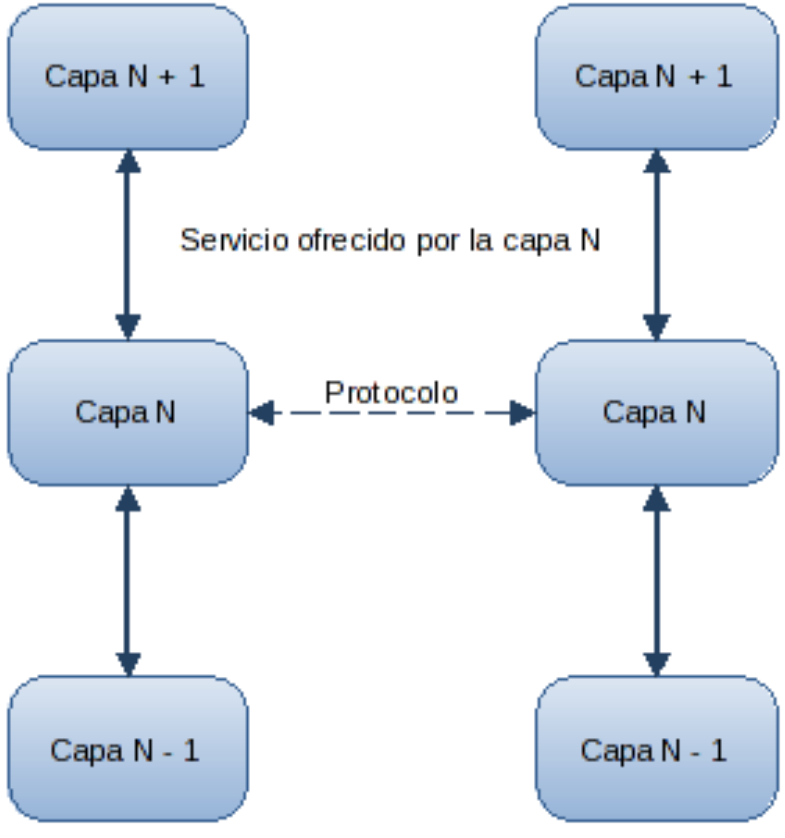
- Arquitectura cliente - servidor
  - Arquitectura de dos niveles: el cliente interactúa directamente con el servidor
  - Arquitectura de tres niveles: middleware
    - Capa de software ubicada entre el cliente y el servidor
    - Provee principalmente seguridad y balanceo de carga
- Tipos de servidor
  - Iterativo: atiende las peticiones de a una a la vez
  - Concurrente: puede atender varias peticiones a la vez

.UBAfiuba FACULTAD DE INGENIERÍA

Índice

Repaso de redes (I)

Modelo de capas



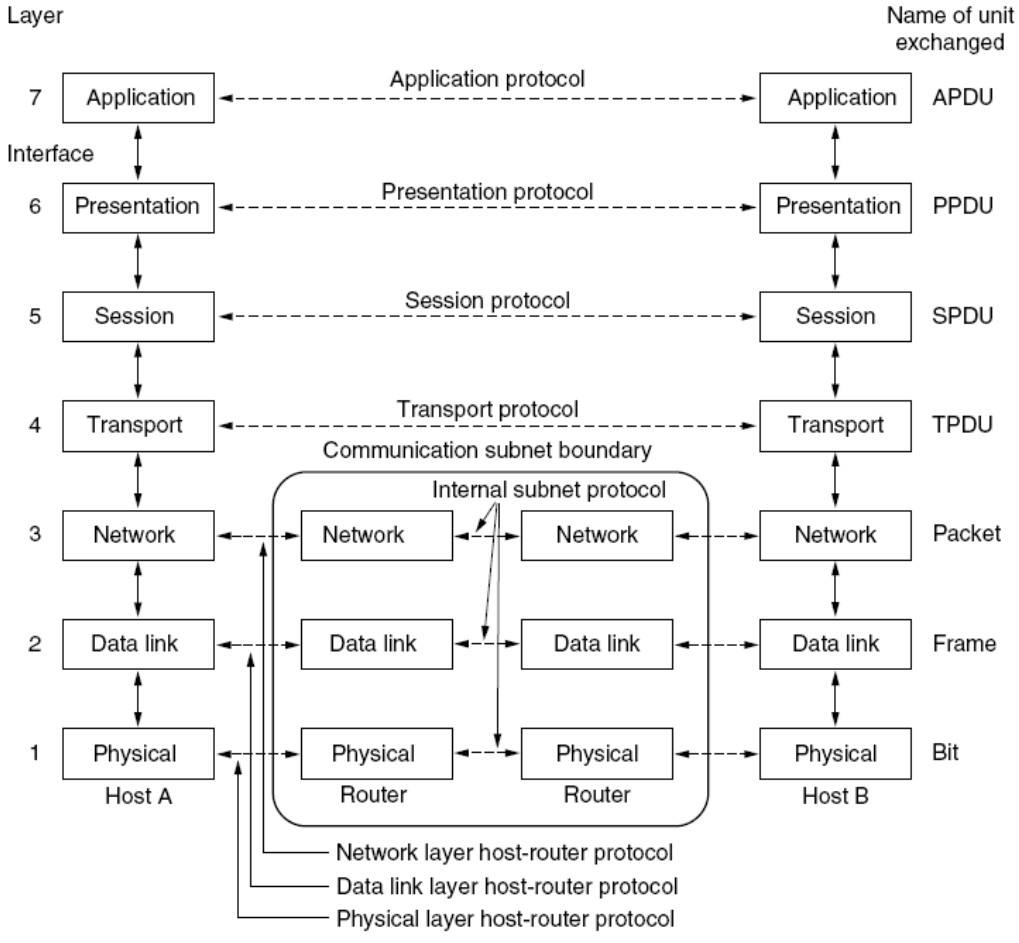
Repaso de redes (II)

Tipos de servicio:

- Sin conexión
  - Los datos se envían al receptor y no hay control de flujo ni de errores.
- Sin conexión con ACK
  - Por cada dato recibido, el receptor envía un acuse de recibo conocido como *ACK*.
- Con conexión
  - Tres fases: establecimiento de la conexión, intercambio de datos y cierre de la conexión. Hay control de flujo y control de errores.

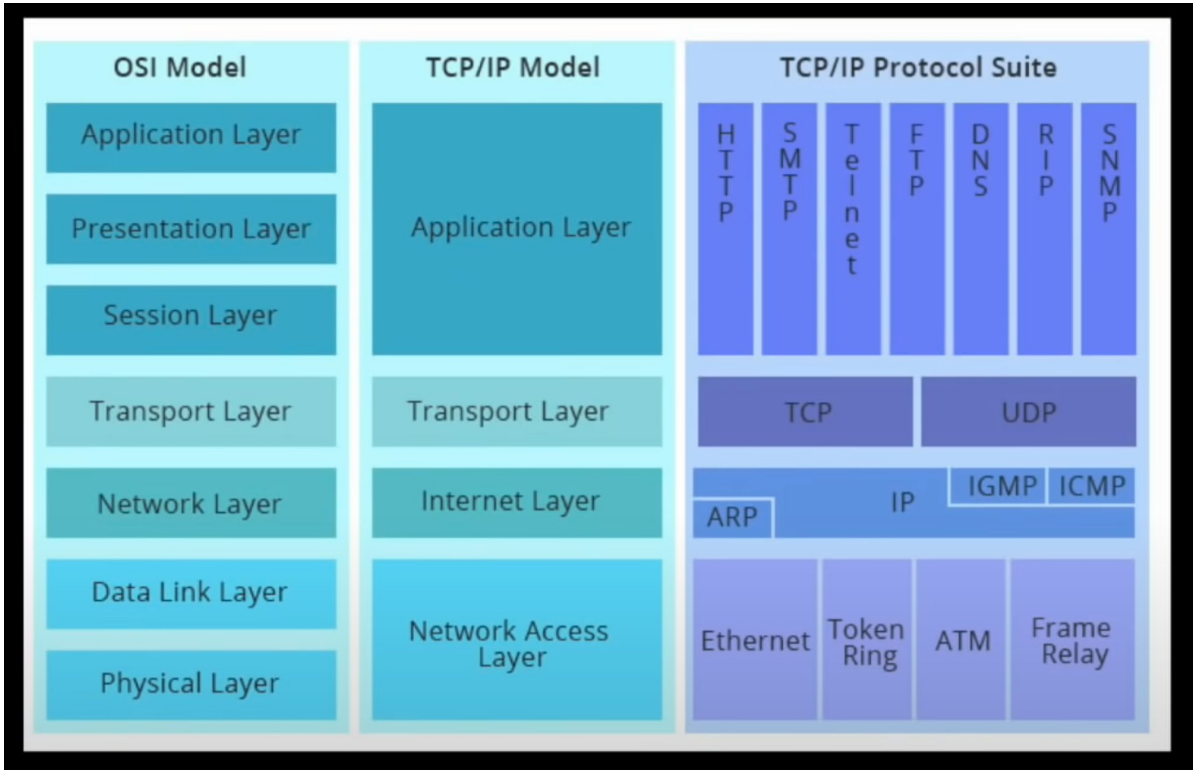
Repaso de redes (III)

Modelo OSI



Protocolos

Protocolos



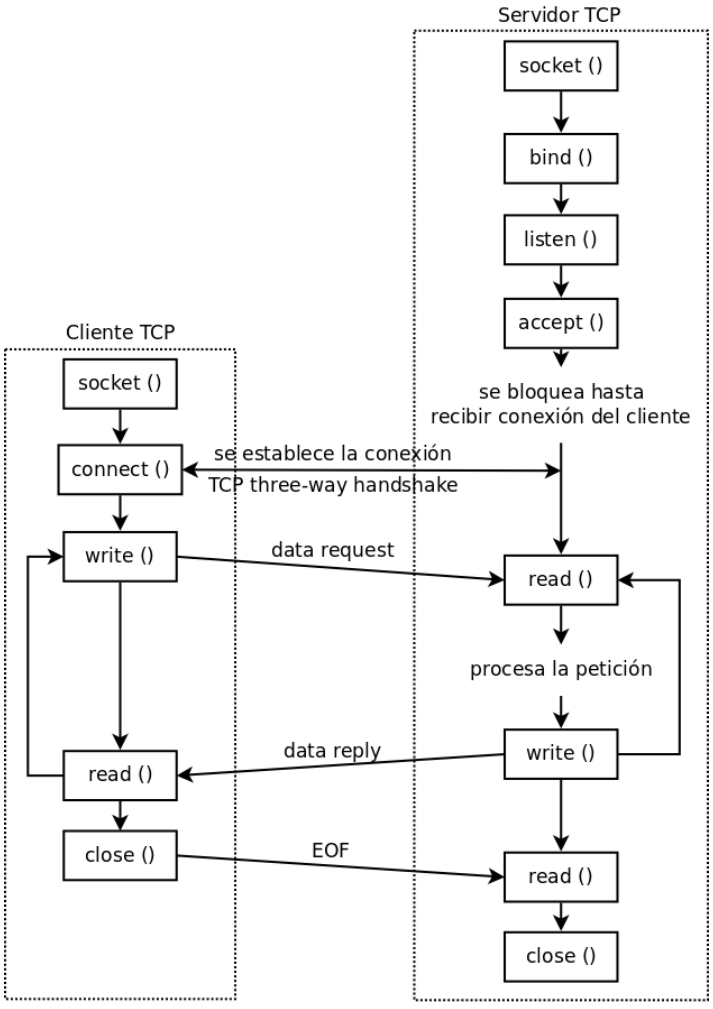
Índice

Tipos de Sockets

Tipos de Sockets

- Stream sockets: usan el protocolo TCP: entrega garantizada del flujo de bytes.
- Datagram sockets: usan el protocolo UDP: la entrega no está garantizada; servicio sin conexión.
- Raw sockets: permiten a las aplicaciones enviar paquetes IP.
- Sequenced packet sockets: similares a *stream sockets*, pero preservan los delimitadores de registro. Utilizan el protocolo SPP (Sequenced Packet Protocol).

Comunicación



Creación del socket

Función *socket ()*

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

- Crea el file descriptor del socket
- Parámetros:
  - Family: AF_INET (IPv4), AF_INET6 (IPv6), AF_UNIX, AF_LOCAL (local)
  - Type: SOCK_STREAM (stream sockets), SOCK_DGRAM (datagram sockets)
  - Protocol: protocolo a utilizar (0, porque normalmente hay un único protocolo por cada tipo de socket)
- Retorna:
  - El file descriptor del socket en caso de éxito (entero positivo)
  - -1 en caso de error, seteando la variable externa *errno*

Conexión

Función *connect ()*

```
int connect ( int sockfd,struct sockaddr *serv_addr,int addrlen );
```

- Inicia una conexión con el servidor
- Parámetros:
  - sockfd: file descriptor del socket
  - serv_addr: puntero a estructura que contiene dirección IP y puerto destino; se arma con *gethostbyname()* y *bcopy()*
  - addrlen: tamaño de struct sockaddr
- Retorna:
  - 0 en caso de éxito
  - -1 en caso de error, seteando la variable externa *errno*

Lectura, escritura y cierre del socket

Lectura y escritura

- Función *read()*: lee bytes del socket
- Función *write()*: escribe bytes en el socket
- Funciones *send()* y *recv()*: para comunicación usando stream sockets
- Funciones *sendto()* y *recvfrom()*: para comunicación usando datagram sockets

Cierre

- Función *close()*

Conexión pasiva (I)

Función *socket ()*: ídem cliente

Función *bind ()*

```
int bind ( int sockfd,struct sockaddr *my_addr,int addrlen );
```

- Asigna una dirección local al socket
- Parámetros:
  - sockfd: file descriptor del socket
  - my_addr: puntero a estructura que contiene dirección IP y puerto local
  - addrlen: tamaño de struct sockaddr
- Retorna:
  - 0 en caso de éxito
  - -1 en caso de error, seteando la variable externa *errno*

Conexión pasiva (II)

Función *listen ()*

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

- Convierte un socket sin conectar en socket pasivo
- Parámetros:
  - sockfd: file descriptor del socket
  - backlog: longitud máxima de la cola de conexiones pendientes que puede tener el servidor
- Retorna:
  - 0 en caso de éxito
  - -1 en caso de error, seteando la variable externa *errno*

Conexión pasiva (III)

Función *accept ()*

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

- Retorna la siguiente conexión completa de la cola de conexiones
- Parámetros:
  - sockfd: file descriptor del socket
  - cliaddr: puntero a estructura con la dirección del cliente
  - addrlen: tamaño de struct sockaddr
- Retorna:
  - El file descriptor del cliente en caso de éxito; se utiliza para comunicarse con el cliente.
  - -1 en caso de error, seteando la variable externa *errno*.

Lectura, escritura y cierre del socket

Lectura y escritura

- Función *read()*: ídem cliente
- Función *write()*: ídem cliente
- Funciones *send()* y *recv()*: ídem cliente
- Funciones *sendto()* y *recvfrom()*: ídem cliente

Cierre

- Función *close()*

Estructuras (I)

```
struct sockaddr {
        unsigned short sa_family;
        char sa_data[14];
};
```

Permite guardar información sobre la dirección de un socket.

- sa_family: familia a la cual pertenece la dirección; usaremos únicamente AF_INET
- sa_data: representa la dirección del socket (dirección y puerto); depende del protocolo que se esté utilizando

Estructuras (II)

```
struct sockaddr_in {
        short int sin_family;
        unsigned short int sin_port;
        struct in_addr sin_addr;
        unsigned char sin_zero[8];
};
```

Solamente para IPv4.

- sin_family: familia de la dirección (AF_INET)
- sin_port: puerto del servicio
- sin_addr: dirección IP (INADDR_ANY)
- sin_zero: se utiliza solamente para que el tamaño coincida con *struct sockaddr* y poder castear

Estructuras (III)

```
typedef uint32_t in_addr_t;
struct in_addr {
        in_addr_t s_addr;
};
```

Miembros:

- s_addr: dirección donde escuchará el servidor (INADDR_ANY)

Bibliografía

- **Distributed Operating Systems**, Andrew S. Tanenbaum, capítulo 3
- Manuales del sistema operativo
- **Computer Networks**, Andrew S. Tanenbaum y David J. Wetherall, quinta edición
- **Unix Network Programming - Volume 1 - The Sockets Networking API**, Richard Stevens, tercera edición