

# ***System Programming / C++***

## Sesión 3

IPLAN 2006 / NITS

Ing. Matías Alejo García

# ***Temas del curso***

---

- **System programming [3 clases]**
- Introducción C++ [2 clases]
- OOP en C++ [3 clases]
- C++ avanzado [2 clases]

# ***Contacto***

---

- [matias@nits.com.ar](mailto:matias@nits.com.ar)
- Moodle @ [nits.com.ar/uade](http://nits.com.ar/uade)
- LinkedIn : [www.linkedin.com/in/matias](http://www.linkedin.com/in/matias)

# Sesión anterior

---

- **Threads.** Stack.
- `pthread_create` / `pthread_join` / `pthread_exit`
- `pthread_detach`
- `pthread_mutex_lock` / `_unlock`
- **Berkeley Sockets:** Revisión de llamadas al sistema

# IPC

---

- **Pipes** ( unnamed / named ). pipe(), mkfifo(), popen().
- **Señales** C: ( signal, kill) | POSIX: (sigaction, sigprocmask, sigpending, sigsuspend )
- **Message queues (msgget, msgsend, msgctl) | POSIX (mq\_open, mq\_close, mq\_send, mq\_receive, mq\_notify, mq\_setattr)**
- **Semaphores (semget, semctl, semop)**
- **Memory** (shmget, shmat, shmt), POSIX (shm\_open, shm\_unlink)
- **Sockets**

# *Programa para hoy*

---

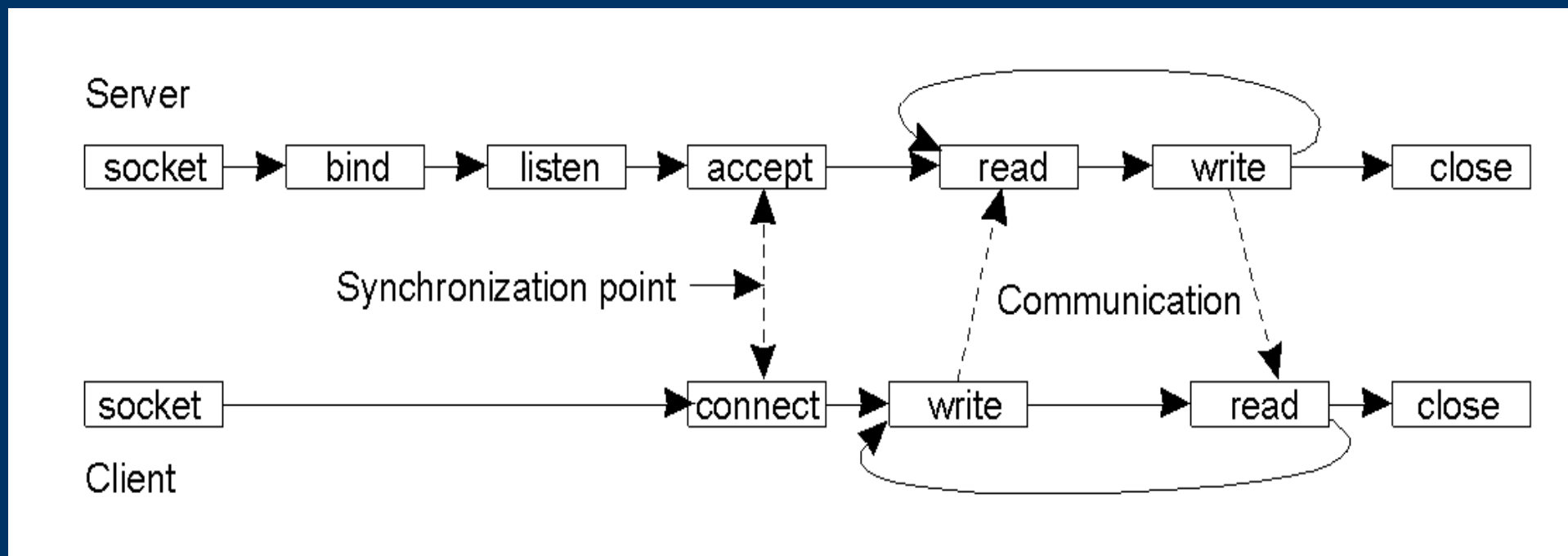
- Berkeley Sockets (socket, bind, listen, accept, send, recv y select)
- Semáforos y memoria compartida

# BSD Sockets

---

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

# BSD Sockets



# BSD Sockets | socket

---

- **int socket (int family, int type, int proto)**
- **Crea un extremo de comunicación.**
- **family:** AF\_UNIX, AF\_INET, ...  
**type:** SOCK\_STREAM, SOCK\_DGRAM, SOCK\_RAW  
**proto:** IPPROTO\_TCP, IPPROTO\_UDP ó 0, u otro si SOCK\_RAW (RFC1700).
- retorna el file descriptor del socket.
- get/setsockopt : IP options (e.j IP\_TTL / IP\_MTU\_DISCOVER)

# BSD Sockets | *bind*

- **int bind** (int socket, struct sockaddr\* addr, int addr\_len)
- Enlaza un socket a una dirección local.
- En TCP/UDP, las direcciones consisten en IP/Puerto:

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    u_int16_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};
/* Internet address. */
struct in_addr {
    u_int32_t      s_addr;    /* address in network byte order */
};
```

- INADDR\_ANY

# BSD Sockets | *listen*

---

- **int listen** (int socket, int backlog)
- Configura el socket para escuchar nuevas conexiones. (socket pasivo)
- backlog define la cantidad máxima de conexiones sin aceptar. Después de este máximo, los clientes reciben ECONNREFUSED.
- backlog -> máxima de conexiones COMPLETADAS.
- Incompletas con `sysctl tcp_max_syn_backlog (1K)`
- Syncookies: `/proc/sys/net/ipv4/tcp_syncookies`

# BSD Sockets | *accept*

---

- **int accept** (int socket, struct sockaddr \* addr, int addr\_len)
- Acepta una conexión (a nivel aplicación). Sólo para sockets pasivos y binded.
- Bloquea hasta que un cliente se conecta.
- Retorna un nuevo socket, activo, para comunicarse con el nuevo cliente.
- Completa addr y addr\_len.

# BSD Sockets | connect

---

- **int connect** (int socket, struct sockaddr \* addr, int addr\_len)
- Conecta un socket con un cliente remoto. (sólo para sockets activos).
- TCP: Realiza el H-S con el destino.  
UDP: Sólo configura el addr en el OS.
- El puerto fuente se toma la ~ azar.

# BSD Sockets | *recv*

---

- `ssize_t recv (int s, const void *buf, size_t len, int flags);`
- Recibe hasta 'len' bytes de socket 's'.
- retorna la cantidad de bytes leídos (0). Si es UDP, los bytes no leídos se descartan. Si es TCP, quedan para la próxima llamada.
- Variante `recvfrom`. Agrega dos parámetros (`addr/addrlen`) que se completan con datos del emisor.
- Si no hay datos disponibles, boquea o devuelve -1) (dependiendo la configuración del socket).
- Algunos flags: `MSG_OOB/_PEEK/_WAITALL/_TRUNC`

# BSD Sockets | *send*

---

- `ssize_t send (int s, const void *buf, size_t len, int flags);`
- EMSGSIZE si demasiado grande.
- bloquea si el buffer esta lleno. (dependiendo de la configuración de s)
- Variante `sendto` para UDP (sin connect).
- flags: `_OOB/_DONTROUTE/_MORE (!)`

# BSD Sockets | Direcciones

## ➤ En General

```
struct sockaddr {  
    u_char    sa_len;  
    u_short   sa_family;  
    char      sa_data[14];  
}
```

## ➤ Para TCP/IP:

```
struct sockaddr_in {  
    u_char    sin_len;  
    u_short   sin_family;  
    u_short   sin_port;  
    struct in_addr  sin_addr;  
    char      sin_zero[8]  
}
```

```
➤ struct in_addr { u_int32_t s_addr };
```

# BSD Sockets | Enteros

---

- Network byte order vs. Host byte order
- 32 / 16 bits
- htonl / htons
- ntohl / ntohs
- Ej:  
struct sockaddr\_in sin;  
sin.sin\_port = htons ( atoi (user\_input\_port) ) ;

# ***BSD Sockets | Utilidades***

---

- `in_addr_t inet_addr (char* in); (net. order)`  
`char* inet_ntoa(struct in_addr in);`
- `int inet_aton(char* in, struct in_addr *out);`
- Ej:  
`struct sockaddr_in sin;`  
`sin.sin_addr.s_addr = inet_addr ("12.23.34.45");`

# BSD Sockets | Utilidades

- Otras llamadas útiles
- (struct hostent\*) = **gethostbyname** ("confronte.com")
- struct hostent {  
    char \*h\_name; /\* official name \*/  
    char \*\*h\_aliases; /\* alias list \*/  
    int h\_addrtype; /\* host address ty\*/  
    int h\_length; /\* length of address \*/  
    char \*\*h\_addr\_list; /\* list of addresses \*/  
}  
    #define h\_addr h\_addr\_list[0] /\* for compat \*/
- struct servent\* = getservbyname ("smtp, "tcp");
- struct protoent\* = getprotobyname ("udp");

# BSD Sockets

---

- shutdown / getpeername
- Las señales despiertan las llamadas bloqueantes. SIGALRM.
- baby steps.
- man is your friend!

# ***BSD Sockets | Práctica***

---

➤ Práctica:

UDPEchoServer / Cliente

TCPtelnet server (no concurrente)

# BSD Sockets | Concurrency

---

- **Concurrency** : Manejar dos clientes a la vez
- No tiene que ver con la velocidad.
- Se puede implementar de dos maneras:
  - Single Thread
  - Multithread (ó Multiproceso)

# BSD Sockets | TCP

---

- Multithread
  - accept (socket pasivo)
  - fork
    - // Hijo
    - send / recv sobre el socket activo.
    - close
  - // Padre repite

# *BSD Sockets | Práctica*

---

➤ Práctica:

## TCPTelnet server

(concurrente: fork o pthreads)

# BSD Sockets | TCP

---

- Single Thread: Tenemos que evitar bloquear la ejecución por sólo 1 evento.
- Opciones:
  - sockets Non-blocking + pulling
  - alarm
  - poll
  - **select**

# ***BSD Sockets | select***

---

- ***select*** recibe:
  - un conjunto de file descriptors para monitorear
  - un timeout
- ***select*** bloquea hasta que algunos de los file descriptors tengan actividad ó hasta que expire el timeout.
- ***select*** no reemplaza a las llamadas de sockets!

# BSD Sockets | select

- *int select (int n, fd\_set \*readfs, fd\_set \*writefs, fd\_set \*exceptfs, struct timeval \*timeout)*
- **retorna** el número de fds con actividad ó cero en caso de salida por timeout.
- struct **timeval** {  
    unsigned int tv\_sec;  
    unsigned int tv\_usec;  
};
- En algunos OS, en caso de actividad en el socket, timeout se devuelve con el tiempo transcurrido desde la llamada a select...pero no siempre!
- Esto significa que debemos reinicializar el valor de timeval cada llamada.
- Cuando usleep() no esta disponible, select se usa para sleep menos de 1 segundo. (con los tres sets en NULL y N en 0).

# BSD Sockets | select

---

- *int select (int n, fd\_set \*readfs, fd\_set \*writefs, fd\_set \*exceptfs, struct timeval \*timeout)*
- *readfs* : se monitorea que un read sobre estos fds no bloquee. Incluye close/resets. accept() si el socket es pasivo.
- *writefs* : se monitorea que un write sobre estos fds no bloquee. (ó NON\_BLK connect ok)
- *exceptfs* : se monitorea que estos fds posean excepciones (OOB, ó NON\_BLK connect failed ).

# BSD Sockets | select

- *int select (int n, fd\_set \*readfs, fd\_set \*writefs, fd\_set \*exceptfs, struct timeval \*timeout)*
- fd\_set es un contenedor de file descriptor.
- Su valor se modifica luego de select, reflejando los fds que **tienen actividad**. Hay que reinicializarlos cada llamada.
- Para utilizar fd\_set existen los siguientes macros:
  - FD\_CLR (int fd, fd\_set\* set);
  - FD\_ISSET (int fd, fd\_set\* set);
  - FD\_SET (int fd, fd\_set\* set);
  - FD\_ZERO (fd\_set \*);
- **n** debe contener el número de fd más grande.

# BSD Sockets | select

```
while (running) {
    FD_ZERO(&sockSet);
    FD_SET(STDIN_FILENO, &sockSet);
    for (port = 0; port < noPorts; port++)
        FD_SET(servSock[port], &sockSet);
    selTimeout.tv_sec = timeout;      /* timeout (secs.) */
    selTimeout.tv_usec = 0;          /* 0 microseconds */

    if (select(maxDescriptor + 1, &sockSet, NULL, NULL, &selTimeout) == 0)
        printf("No echo requests for %ld secs...Server still alive\n", timeout);
    else {
        if (FD_ISSET(0, &sockSet)) {
            printf("Shutting down server\n");
            getchar();
            running = 0;
        }
        for (port = 0; port < noPorts; port++)
            if (FD_ISSET(servSock[port], &sockSet)) {
                printf("Request on port %d: ", port);
                HandleTCPClient(AcceptTCPConnection(servSock[port]));
            }
    }
}
```

# ***BSD Sockets | Práctica***

---

➤ Práctica:

TCP Chat server

(concurrente / single thread)

# BSD Sockets | Sendall

---

```
int sendall(int s, char *buf, int *len)
{
    int total = 0;           // how many bytes we've sent
    int bytesleft = *len;   // how many we have left to send
    int n;
    while(total < *len) {
        n = send(s, buf+total, bytesleft, 0);
        if (n == -1) { break; }
        total += n;
        bytesleft -= n;
    }
    *len = total; // return number actually sent here
    return n==-1?-1:0; // return -1 on failure, 0 on success
}
```