

# Programación Distribuida y Tiempo Real

---

Explicación de Práctica 2

---

# “Evolución” TP1 a TP2

- Sockets y Tipos de Datos
  - Sockets TCP/UDP... ¿qué capa de red?

---

# “Evolución” TP1 a TP2

- Sockets y Tipos de Datos
    - Sockets TCP/UDP... ¿qué capa de red? ==> ¡4!
    - Veremos comunicaciones en SD... que no son sockets
    - Datos binarios, seguro NO char ni string
      - Dependerá del lenguaje cómo se implementa “byte”
  - Medición de tiempos
    - “Despegarse” del ejemplo de sockets dado...
  - “Warning”: caso de write con menor cantidad de la requerida
-

---

# Medición de Tiempos en TP2 (SD)

- $t_0$  inmediatamente anterior al snd
  - $t_1$  inmediatamente rcv en el mismo sitio del snd
  - “Elapsed” =  $t_1 - t_0$
  - Pseudocódigo:
    - $t_0 = \text{“timestamp”}$
    - snd(...)
    - rcv(...)
    - $t_1 = \text{“timestamp”}$
    - elapsed =  $t_1 - t_0$
  - ¿Datos?
    - Binarios
    - ¿Cuántos?
-

# Medición de Tiempos en TP2 (SD)

- $t_0$  inmediatamente anterior al snd
- $t_1$  inmediatamente rcv en el mismo sitio del snd
- “Elapsed” =  $t_1 - t_0$
- Pseudocódigo:

$t_0 = \text{“timestamp”}$

snd(...)

rcv(...)

$t_1 = \text{“timestamp”}$

elapsed =  $t_1 - t_0$

- ¿Datos?
  - Binarios
  - ¿Cuántos?

snd ==> todo lo necesario para enviar
rcv ==> todo lo necesario para recibir

---

# Medición de Tiempos en TP2 (SD)

- Cantidad de Datos snd - rcv
    - Problema del “ack” en el rcv
    - Misma cantidad ==> mismo mensaje ida y vuelta (p-p)
  - Al tener la misma cantidad en snd-rcv
    - En general no necesariamente son las mismas rutas
    - “elapsed” son 2 comunicaciones idénticas
    - “One way”: elapsed/2
  - No hay forma de evitar todos los problemas/las opciones
-

---

# Protocolos y Bibliotecas

- Pendiente de TR en SD
    - Casi no existe entorno de TCP/IP con QoS
  - Socket NO es TCP ni UDP
    - Interfaz de comunicaciones entre procesos (UNIX)
  - Misma cantidad ==> mismo mensaje ida y vuelta (p-p)
  - Al tener la misma cantidad en snd-rcv
    - En general no necesariamente son las mismas rutas
    - “elapsed” son 2 comunicaciones idénticas
    - “One way”: elapsed/2
  - No hay forma de evitar todos los problemas/las opciones
    - Ver los puntos de sincronización
    - Sincronización y Comunicaciones
-

---

# Sincronizaciones

- Dependerá casi exclusivamente del lenguaje de progr.
  - En C
    - “Server” - “Client”  
srvr: socket() - bind() - listen() - accept() - read() - write()  
clnt: socket() - connect() - write() - read()
-



---

# Sincronizaciones

- Dependerá casi exclusivamente del lenguaje de progr.

- En C

- “Server” - “Client”

srvr: socket() - bind() - listen() - accept() - read() - write()

clnt: socket() - connect() - write() - read()

srvr: socket() - bind() - listen() - accept() - read() - write()

clnt: socket() - connect() - write() - read()

---

# Sincronizaciones

- Dependerá casi exclusivamente del lenguaje de progr.
- En C

- “Server” - “Client”

srvr: socket() - bind() - listen() - accept() - read() - write()

clnt: socket() - connect() - write() - read()

srvr: socket() - bind() - listen() - accept() - read() - write()

clnt: socket() - connect() - write() - read()



Asincrónico



Sincrónico



# Sincronizaciones

- Dependerá casi exclusivamente del lenguaje de progr.
- En C

- “Server” - “Client”

srvr: socket() - bind() - listen() - accept() - read() - write()

clnt: socket() - connect() - write() - read()

srvr: socket() - bind() - listen() - accept() - read() - write()

clnt: socket() - connect() - write() - read()



Asincrónico



Sincrónico



¿Sincrónico o Asincrónico?

# Sincronizaciones

- Dependerá casi exclusivamente del lenguaje de progr.
- En C

- “Server” - “Client”

srvr: socket() - bind() - listen() - accept() - read() - write()

clnt: socket() - connect() - write() - read()

srvr: socket() - bind() - listen() - accept() - read() - write()

clnt: socket() - connect() - write() - read()



Asincrónico



Sincrónico



Unica vez para todas las comunicaciones

¿Sincrónico o Asincrónico?

---

# Sincronizaciones

- Lo “administrativo” puede cambiar según el lenguaje
    - Configurar y conectar sockets
    - Usualmente se simplifica respecto de C
      - Tal como lo vimos con Java
  - Envío y recepción en general mantienen la semántica
    - Finalmente se “reduce” a envío-recepción
    - En varios casos, aún con comunicaciones en SD de mayor nivel de abstracción se mantiene
-

---

# Read-Write ↔ Write-Read

- Funciones read-write ¿Sincrónicas o Asincrónicas?
    - Siempre usan “la misma” implementación de socket
      - Dependerá de cuánto “sobre” ellas hay
    - Involucran el subsistema de entrada/salida
    - Función write: lo necesario continuar sin interferencia
      - Normalmente buffers de socket (¿de I/O?)
    - Función read: datos recibidos... ¿desde dónde?
      - <https://www.linuxjournal.com/article/6345>
  - Función write: copia y se devuelve el control
    - Lo más probable es que ni llegue a las colas de salida
  - Función read: vuelve con “algo leído y disponible”
    - “Blocking” vs. “Non-Blocking”
-

---

# Read-Write $\leftrightarrow$ Write-Read

- Para los experimentos en general:

read() - write()

write() - read()



- ¿Por qué varias iteraciones?
    - vs. “one-shot”
    - En general: promedios
    - En TR: máximos, variaciones
-


---

# Read-Write $\leftrightarrow$ Write-Read

- Para los experimentos en general:

read() - write()

write() - read()

- ¿Por qué varias iteraciones?
    - vs. “one-shot”
    - En general: promedios
    - En TR: máximos, variaciones
  - Suele ser representativo del “mejor caso” de comunicaciones
- 
-



---

# Read-Write $\leftrightarrow$ Write-Read

- Para los experimentos en general:  
    read() - write()  
    write() - read()
  - Pseudocódigo:  
    for i ... cant\_exp  
         $t_0 = \text{"timestamp"}$   
        snd(...)  
        rcv(...)  
         $t_1 = \text{"timestamp"}$   
         $\text{elapsed}_i = t_1 - t_0$   
    end for
-

# Read-Write $\leftrightarrow$ Write-Read

- Para los experimentos en general:

read() - write()

write() - read()

- Pseudocódigo:

```
for i ... cant_exp
```

```
    t0 = “timestamp”
```

```
    snd(...)
```

```
    rcv(...)
```

```
    t1 = “timestamp”
```

```
    elapsedi = t1 - t0
```

```
end for
```

snd ==> todo lo necesario para enviar
rcv ==> todo lo necesario para recibir

# Read-Write $\leftrightarrow$ Write-Read

- Para los experimentos en general:

read() - write()

write() - read()

- Pseudocódigo:

```
for i ... cant_exp
```

```
  t0 = “timestamp”
```

```
  snd(...)
```

```
  rcv(...)
```

```
  t1 = “timestamp”
```

```
  elapsedi = t1 - t0
```

```
end for
```

snd ==> todo lo necesario para enviar  
rcv ==> todo lo necesario para recibir

... o t<sub>0</sub> y t<sub>1</sub> “fuera”  
de la iteración...

---

# Read-Write ↔ Write-Read

- Para los experimentos en general:

read() - write()

write() - read()



- Con elapsed<sub>i</sub>: todo lo que se necesite de análisis
  - Suele ser representativo del “mejor caso” de comunicaciones
  - Objetivos de la Práctica 2:
    - Ambiente distribuido en LAN (vs. local o simulado)
    - Ambiente de experimentación “automatizado”
  - Vagrant
    - Ej. vagrantfile
    - `vagrant ssh <mv> -c “<comando>”`
-