

REDES DE COMPUTADORAS

INSTITUTO DE COMPUTACIÓN

DEPARTAMENTO DE ARQUITECTURA DE SISTEMAS

API de sockets para un lenguaje de alto nivel

A continuación se presenta una API de sockets simplificada para un lenguaje de alto nivel imaginario. El lenguaje posee tipado dinámico, tiene un valor nulo llamado `nil` y soporta retornos múltiples. Está vagamente inspirado en Lua (lua.org), y la API está inspirada en `luasocket` (w3.impa.br/~diego/software/luasocket/).

Sockets UDP

Antes de empezar a invocar las funciones `skt.sendto()` y `skt.receive()` el socket deberá ser asociado a una dirección local usando `skt.bind()`. Los modos bloqueantes sólo se aplican a la operación de lectura (el envío en UDP es conceptualmente no-bloqueante).

```
skt = socket.udp()
```

Crea un socket UDP.

```
skt.close()
```

Cierra el socket y libera los recursos asociados.

```
skt.bind(address, port)
```

Establece la dirección local de un socket. El parámetro `address` puede ser una dirección IP, un hostname, o `'*'` para asociar a todas las interfaces. El parámetro

`port` es un número de puerto, o 0 para dejar que el SO elija un número disponible.

```
ip, port = skt.gethost()
```

Devuelve la dirección IP y número de puerto locales del socket.

```
skt.sendto(datagram, address, port)
```

Realiza el envío de un datagrama a la dirección indicada. Los parámetros `address`, `port` indican la dirección IP o hostname del receptor y su número de puerto respectivamente. El parámetro `datagram` debe respetar el tamaño máximo de un datagrama UDP (64Kbytes - IP overhead).

```
datagram, ip, port = skt.receive(timeout)
```

Realiza la recepción de un datagrama. El parámetro `timeout` indica el tiempo en segundos disponible para realizar una lectura. Si este parámetro está ausente o es negativo, el socket es completamente bloqueante. Si vale 0, el socket es no bloqueante y debe retornar en seguida. En caso de no obtener datos en el tiempo asignado, la llamada retorna `nil`.

Si la función devuelve un datagrama, adicionalmente devuelve la dirección IP y puerto del socket origen.

Sockets TCP

Luego de instanciados los sockets son de tipo *master*. A continuación pueden ser convertidos en sockets *server* (usando `listen`), o *client* (usando `connect`). Se pueden obtener sockets *client* adicionales invocando `accept` en un socket *server*.

Los modos bloqueante o no bloqueante se aplican a las operaciones de lectura, escritura y `accept`. Por defecto los sockets se crean como bloqueantes.

```
master = socket.tcp()
```

Crea un socket *master* TCP.

`master.bind(address, port)`

Establece la dirección local de un socket. El parámetro `address` puede ser una dirección IP, un hostname, o `'*'` para asociar a todas las interfaces. El parámetro `port` es un número de puerto, o 0 para dejar que el SO elija un número disponible.

`server = master.listen()`

Convierte un socket *master* en un socket *server*, capaz de aceptar conexiones.

`client, err = server.accept()`

Espera a que se establezca una conexión. Devuelve un socket cliente TCP conectado. Si en el tiempo asignado no se establece ninguna conexión nueva, devuelve `nil`, `'timeout'` (ver función `server.settimeout()`).

`master.close()`

`server.close()`

`client.close()`

Cierra el socket y libera los recursos asociados.

`client, err = master.connect(address, port)`

Establece una conexión a un socket *server*, y convierte al socket *master* en un socket *client*. Los parámetros indican la dirección IP o hostname del servidor, y su número de puerto respectivamente. En caso de fallar el intento de conexión devuelve `nil`, `'failure'`.

`ip, port = client.gethost()`

Devuelve la dirección IP y número de puerto locales del socket.

`ip, port = client.getpeer()`

Devuelve la dirección IP y número de puerto del socket remoto en una conexión.

```
client.settimeout(timeout)
```

```
server.settimeout(timeout)
```

Especifica el tiempo máximo disponible para las operaciones `client.send()`, `client.receive()` y `server.accept()` (según corresponda), en segundos. Si `timeout` es 0 significa que el socket es no bloqueante y las llamadas deben responder enseguida; si `timeout` es negativo significa que el socket es completamente bloqueante.

```
data, err = client.receive()
```

Realiza una lectura en un socket conectado. Devuelve la información disponible en el stream en `data`. Si expira el *timeout* sin obtener datos nuevos devuelve `''`, `'timeout'` (ver función `client.settimeout()`). Si no hay datos para devolver y la conexión esta cerrada devuelve `nil`, `'closed'`.

```
remain, err = client.send(data)
```

Realiza una escritura en un socket conectado. De `data` se transmitirá lo que se pueda en el tiempo asignado. En el parámetro `remain` se devuelve la parte de `data` que aun no se logró transmitir. Si se logró transmitir todo, `remain` es un string vacío. En caso de salir por *timeout* devuelve `remain`, `'timeout'` (ver función `client.settimeout()`). En caso que el socket esté cerrado devuelve `remain`, `'closed'`.

Threads

Se implementan *preemptive threads*.

```
thread.new(f, param1, param2, ...)
```

Lanza la función `f` en un nuevo hilo de ejecución. La función `f` recibe `param1`, `param2`, `...` como parámetros. Además de estos parámetros, la función tiene acceso a las variables globales del programa. El hilo se destruye al finalizar la función.