Redes de Computadoras

Obligatorio 2 - Programación con Sockets

Se debe implementar en *C* una aplicación que mediante *sockets* permita enviar y recibir mensajes, a otros pares, es decir, la aplicación en todo momento puede recibir mensajes, y también puede enviarlos a otra aplicación, para esto el programa debe *bifurcarse*, y una parte de este debe encargarse de *recibir* mensajes, y otra de *transmitir*. Estos mensajes tendrán un largo máximo definido.

```
#define MAX_LARGO_MENSAJE 255
#define MAX_LARGO_ARCHIVO 65535
```

Para la implementación se deben usar las siguientes bibliotecas de *C* entre otras.

```
<unistd.h>
<sys/types.h>
<sys/wait.h>
<sys/stat.h>
<sys/ipc.h>
<sys/shm.h>
<sys/socket.h>
```

El programa será invocado de la siguiente manera:

```
./mensajeria port
```

Levantará el argumento *port*, que será el puerto donde oirá el *receptor*, y escribirá el *emisor*.

El programa constará de dos grandes partes, una *emisora* y otra *receptora*.

- **Receptora:**

Esta oirá continuamente por un *puerto* que se le pasará por argumento al programa
Al llegar los mensajes a ese *puerto*, los imprimirá en pantalla (salida estándar), junto con la fecha y hora, los mensajes serán *strings* que tendrán el *ip* del emisor, seguido de un espacio y el mensaje, estos mensajes, como ya se dijo, tendrán un largo máximo.
Ejemplo de impresión del receptor:

```
[2013.05.12 17:02] 172.23.20.23 Feliz Cumple!!!!!
```

Al llegar archivos, los guardara en el directorio "*actual*" del programa en el *FileSystem* local.
Ejemplos de impresión del receptor:

```
[2013.05.12 17:05] 172.23.10.40 <Recibido ./foto.jpg>
[2013.05.12 17:05] 172.23.10.40 <Error Recibiendo Archivo>
```

- **Emisora:**

Esta enviará mensajes a un determinado *receptor* que se le pasara como parámetro al puerto pasado, levantará de la entrada estándar el destino del mensaje (*ip*) y el contenido, por ejemplo:

```
172.23.20.23 Feliz Cumple!!!!!
```

Si en lugar de poner un *ip* se pone un "*" se debe enviar el mensaje a todos los receptores presentes en la red del emisor.

```
* Gracias a todos los que se acordaron
```

Cuando un mensaje es enviado a todos todos (*Broadcast*) el receptor también debe imprimir quien en el emisor (*como con el resto de los mensajes*).

Si en lugar de un mensaje se pone un "*&file"* y un "*path*" se debe enviar el archivo que esta en *path* en el FS local.
Ejemplo de impresión del receptor:

```
* &file ./foto.jpg
172.23.20.25 &file ./foto.jpg
```

Ambas partes del programa escribirán en la salida estándar, por lo que en pantalla aparecerán mezcladas las salidas del *emisor* y del *receptor*.

Ejemplo de Sesión:

```
 ./mensajeria 22764
[2013.05.12 17:02] 172.23.20.23 Feliz Cumple!!!!!
 172.23.20.23 Gracias!!!!!
[2013.05.12 17:03] 172.23.20.23 Merece!!!!!
172.23.1.25 No me tenes que decir nada hoy??
[2013.05.12 17:05] 172.23.1.25 Por?
172.23.1.25 Es mi cumple!!!
[2013.05.12 17:06] 172.23.1.25 ahh que boludo!!
[2013.05.12 17:06] 172.23.1.25 me pensaba que era en mayo!!!
[2013.05.12 17:07] 172.23.1.25 Feliz Cumple!!!
* Gracias a todos los que se acordaron
[2013.05.12 17:08] 172.23.1.10 Gracias a todos los que se
acordaron
CTRL + C Recibido.... Cerrando Sesión

./mensajeria 2244
172.23.20.23 &file ./foto.jpg
172.23.20.23 Te llegó el archivo???
[2013.05.12 18:06] 172.23.20.23 Si, impecable!!!!!
CTRL + C Recibido.... Cerrando Sesión
```

El programa debe ir generando un *log*, volcando todo lo que sale en la pantalla en un archivo `mensajeria.log` aclarando la hora de inicio de cada *sesión*, el *puerto* a la escucha y el *pid* de los procesos que participan de la solución.

Contenido generado para `mensajeria.log` para el ejemplo anterior:

```
<<<2013.05.12 17:07 - puerto 2244 con pids = 3354 y 3355>>>
172.23.20.23 &file ./foto.jpg
172.23.20.23 Te llegó el archivo???
[2013.05.12 18:06] 172.23.20.23 Si, impecable!!!!!
CTRL + C Recibido.... Cerrando Sesión
.
```

El programa deberá atender las señales y alarmas del sistema, y se terminará al recibir un CTRL + C o recibir un KILL o TERM. Al recibir estas señales deberá cerrar de manera correcta los recursos que haya solicitado, sockets, memoria, etc.

El comportamiento de entradas y salidas debe ser el que figura en los ejemplos.

- **El trabajo se realizará en grupos de hasta 4 personas.**
- **Se realizará en un entorno tipo UNIX (se recomienda la máquina virtual con backtrack2 disponible para el curso) y se compilará con gcc o g++.**
- **Se deberá entregar:**
    - `mensajeria.c`

Este contendrá la solución al problema y un comentario con los nombres y números de documento de todos los integrantes del grupo.

    - `makefile`

Para compilar usando `make`**.**

    - `comentarios.txt`

Contendrá comentarios sobre la solución, errores, etc.

- **Estos archivos deberán venir comprimidos en uno que se llamará:**
    - `redes-e2.tar.gz`
- **Hay tiempo hasta el hasta el día viernes 14 de junio a las 23:59.**
- **Los detalles de la entrega se aclararán más adelante.**

Los trabajos que no respeten el formato de entrega no serán considerados.

- ***Lecturas Recomendadas:***
    - ***Bifurcación:***

http://www.opengroup.org/onlinepubs/000095399/functions/fork.html

    - ***Programación Básica con Sockets:***

http://es.tldp.org/Tutoriales/PROG-SOCKETS/prog-sockets.html

    - ***Señales:***

http://www2.dis.ulpgc.es/~itis-ps/signal/index.html
http://www.chuidiang.com/clinux/senhales/senhales.php