Redes de Computadoras

Obligatorio 2 - Programación con Sockets

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

Levantará el argumento *port*, que será el puerto donde oirá el *receptor*, y escribirá el *emisor* y los argumentos *ipAuth* y *portAuth* que indicarán la dirección del servidor de autenticación.

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*, y el *nombre de usuario* del emisor, los mensajes serán *strings* que tendrán el *ip* del emisor, seguido de un espacio seguido del *nombre de usuario*, seguido de un espacio, seguido de "`dice: `" y el *mensaje*, estos mensajes, como ya se dijo, tendrán un largo máximo.

Ejemplo de impresión del receptor:

```
[2014.06.23 17:02] 172.23.20.23 nwirth dice: Feliz Cumple!!!!!
```

Al llegar archivos, los guardara en el directorio "*actual*" del programa en el *FileSystem* local.

Ejemplos de impresión del receptor:

```
[2014.06.23 17:05] 172.23.10.40 <Recibido ./foto.jpg de nwirth>
[2014.06.23 17:05] 172.23.10.40 <Error Recibiendo Archivo de nwirth>
```

- ***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 imprimirlo (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 emisor:

```
* &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*.

- ***Autenticación***

Los usuarios del programa deben autenticarse, para ello se dispondrá de un ejecutable *redes-auth* este será invocado de la siguiente manera:

```
./redes-auth port
```

Este programa autenticará sobre un archivo de texto con el siguiente formato, cada línea será de la forma:

```
usuario-contraseña_condificada_en_md5;Nombres_Apellidos;
```

por ejemplo:

```
aturing-35ecb6586f632e010e2347cd37b8e196;Alan_Mathison_Turing;
alovelace-bd5f3cbebf74ce4f5126e4536fdacb80;Augusta_Ada_Byron;
jkonvalina-9ea70b913234f6853e406734f61880dd;John_Konvalina;
jvneumann- b2b524bf5ad2af00eff6d53a1273cb1d;Neumann_Janos_Lajos;
nwirth-b14b9c09ef4f192978c87badbf51d4f1;Niklaus_Wirth;
cbabbage-cb42f11ca37f9d7219ad46ea672a8549;Charles_Babbage;
edijkstra-79e8782141242b7ca6877c1f949759e1;Edsger_Wybe_Dijkstra;
```

El servidor al conectarse un cliente le responderá:

```
Redes 2014 - Obligatorio 2 - Autenticacion de Usuarios
```

Los clientes enviarán lo siguiente:

```
usuario-contraseña_condificada_en_md5
```

El servidor de autenticación responderá en caso afirmativo

```
SI
Nombres_Apellidos
```

Y en caso negativo:

```
NO
```

Ejemplo de sesión del servidor de autenticación:

```
> Redes 2014 - Obligatorio 2 - Autenticacion de Usuarios
< alovelace-bd5f3cbebf74ce4f5126e4536fdacb80
> SI
> Augusta_Ada_Byron
```

(*los "<" y ">" no forman parte de la salida, indican si son mensajes entrantes o salientes*)

Ejemplo de sesión del programa de mensajería:

```
 ./mensajeria 22764
Usuario: aturing
Clave:
Bienvenido Alan_Mathison_Turing
[2014.06.23 17:02] 172.23.20.23 nwirth dice: Feliz Cumple!!!!!
 172.23.20.23 Gracias!!!!!
[2014.06.23 17:03] 172.23.20.23 nwirth dice: Merece!!!!!
172.23.1.25 No me tenés que decir nada hoy??
[2014.06.23 17:05] 172.23.1.25 edijkstra dice: Por?
172.23.1.25 Es mi cumple!!!
[2014.06.23 17:06] 172.23.1.25 edijkstra dice: ahh que boludo!!
[2014.06.23 17:06] 172.23.1.25 edijkstra dice: me pensaba que era en
mayo!!!
[2014.06.23 17:07] 172.23.1.25 edijkstra dice: Feliz Cumple!!!
* Gracias a todos los que se acordaron
[2014.06.23 17:08] 172.23.1.10 aturing dice: Gracias a todos los que
se acordaron
CTRL + C Recibido.... Cerrando Sesión

./mensajeria 2244
Usuario: aturing
Clave:
Bienvenido Alan_Mathison_Turing
172.23.20.23 &file ./foto.jpg
172.23.20.23 Te llegó el archivo???
[2014.06.23 18:06] 172.23.20.23 nwirth dice: 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:

```
<<<2014.06.23 17:07 - puerto 2244 con pids = 3354 y 3355>>>
172.23.20.23 &file ./foto.jpg
172.23.20.23 Te llegó el archivo???
[2014.06.23 18:06] 172.23.20.23 nwirth dice: 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.

Sobre la Entrega:

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 día viernes 13 de junio a las 23:59 UYT.
• Se entregará a cada grupo un usuario que podrán usar para la implementación, para realizar la entrega deberán dejar el archivo `redes-e2.tar.gz` **en la raíz del usuario.**

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