**Documentación del Sistema**

**Objetivo:**

Nuestra aplicación tiene como objetivo implementar un Help Desk, el cual consta de un chat cliente-servidor en el cual todos los clientes interactúan con un mismo servidor.

**Herramientas Utilizadas:**

Para el desarrollo de la aplicación se ha utilizado la versión de Linux Ubuntu 10.04, en conjunto con algunos entornos de desarrollo como ser Eclipse o Netbeans. La codificación del programa se ha hecho en lenguaje puro en C, a excepción de algunos complementos de las ventanas o eventos que se han codificado utilizando el framework gtk+ 2.0 de Gnome.

**Descripción de la Solución**

A grandes rasgos la aplicación se basa en un esquema cliente servidor, conectado a través de protocolo TCP-IP utilizando sockets. Gracias a los sockets hemos podido sincronizar el cliente y el servidor y además permitir el envió de mensajes entre ellos desde una terminal a otra siempre que estén bajo una red. Para permitir la conexión el programa pedirá a los clientes la dirección ip del servidor (ipv4) u optativamente se podrá escribir localhost en el caso que el servidor y el cliente estén ejecutando en la misma terminal.

El servidor tiene un socket siempre a la escucha y por cada cliente que llega abre un nuevo socket y crea un proceso hijo el cual se encarga de atender al cliente (cada cliente es atendido por un proceso hijo). Esto ocurre siempre y cuando el servidor no haya llegado a su máxima cantidad de clientes, la cual se configura cuando se levanta el servidor o se deja por default en 100.

Para que la persona del lado del Help-Desk pueda distinguir entre los distintos clientes, los mismos deben brindarle al servidor antes de iniciar el chat un nombre de usuario. Para que los nombres de usuarios no se repitan, son guardados en una lista dinámica (ver listas.h) en la cual cada nodo contiene como clave el nombre de usuario y otro atributo que es id, el cual es el process id del proceso hijo creado para atender a ese cliente. Si cuando se ingresa un nombre de usuario el mismo ya existe en la lista, se informa al cliente que ingrese otro nombre de usuario diferente.

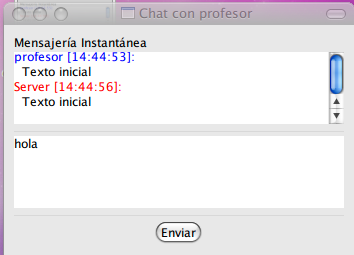
Además, cuando un cliente termina el chat el nombre de usuario del mismo se elimina de la lista. Esto lo hace el servidor-padre, ya que espera a la señal SIGCHLD y se va fijando si terminó o no algún hijo; cuando lo hace busca el nodo en el cual el id sea igual al proccess id y lo borra de la lista liberando de nuevo ese nombre de usuario. Para evitar la corrupción de los datos, esta área se ha considerado como sección critica y se la ha “protegido” poniendo semáforos tanto como para la lectura como para el borrado de la lista.

Para que el usuario del servidor pueda distinguir con facilidad entre los distintos clientes se ha diseñado una ventana de chat utilizando la librería gtk+. La cual facilita al usuario tanto cliente como servidor el manejo de la aplicación Help-Desk. Gracias a este framework una vez que se abre una ventana la aplicación comienza a funcionar orientada a eventos. Involucrados en el proceso hay 3 eventos disponibles, de los cuales 2 los ejecuta el usuario y otro se ejecuta de manera automática:

1. Evento del botón enviar: Este evento ocurre cuando se presiona el botón enviar y lo que sucede aquí es que se toma el texto de la parte de escritura, se copia en la parte superior y se escribe en el buffer del socket ese mensaje.
2. Evento del botón cerrar: Cierra la ventana y manda al otro extremo del socket un mensaje de cerrado, el cual informa a todos los involucrados que ese cliente se ha desconectado.
3. Evento de lectura: Este evento se programo utilizando threads de gtk+, para brindar “paralelismo” entre estos eventos (uno puede escribir mientras se están recibiendo mensajes sin que se bloquee el proceso). Este evento lo que hace es leer cada x cantidad de tiempo del buffer del socket.

**Imágenes de las ventanas:**

Ventana del servidor:



Ventana del Cliente:

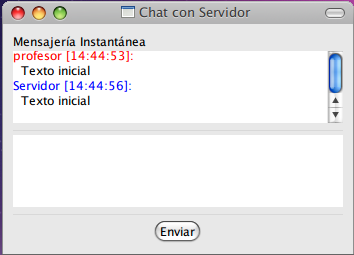
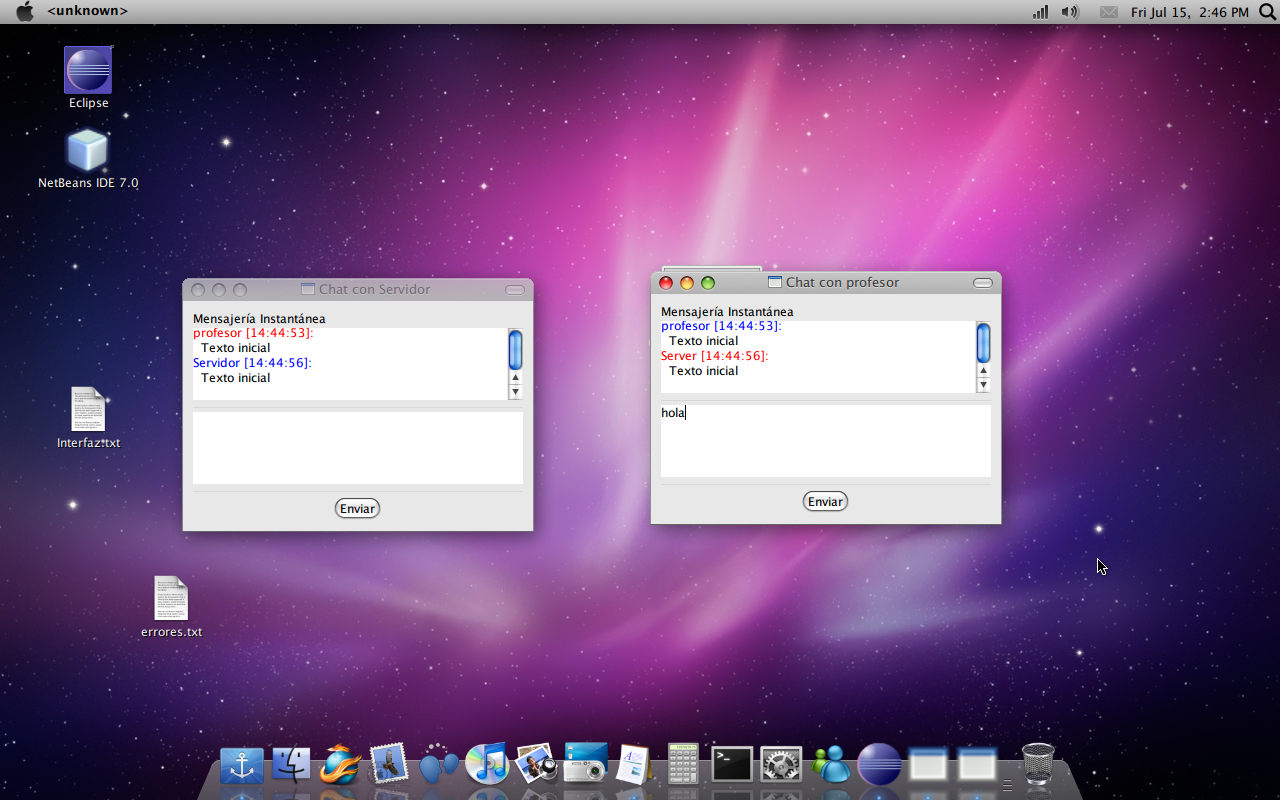


Imagen del Escritorio:



**Recursos del Sistema Operativo y conceptos aprendidos en la materia que se han utilizado:**

* Sockets: Para brindar la comunicación y sincronización entre el cliente y el servidor.
* Semáforos: Para proteger el acceso a la lista dinámica.
* Threads: Para el manejo de los eventos y las ventanas de gtk+.
* Gtk+: Para el armado de las ventanas, los botones y los eventos.

Estructuras de datos usadas:

* Lista Dinámica: Una lista doblemente enlazada circular, donde cada nodo contiene una estructura de tipo Tdato\_String
* Tdato\_String: un registro con un String de 64 caracteres llamado clave y un entero llamado id (se guarda el id del proceso de ese usuario).

**Conclusión**

Se ha intentado armar de forma eficiente y estructurada un sistema que cumpla con los objetivos propuestos al inicio del documento, utilizando conceptos aprendidos durante la cursada e incorporando otros nuevos. Nuestro objetivo primordial fue el de resolver el problema eficientemente e intentar que el sistema quede lo mas user friendly que se pudo.

**Anexo**

**Common Bugs**

Nota: todos los bugs encontrados hasta el momento en esta versión están relacionados con las funciones y estructuras de gtk+ y no con la lógica ni recursos del sistema operativo.

1. (<unknown>:2791): Gtk-WARNING \*\*: Invalid text buffer iterator: either the iterator is uninitialized, or the characters/pixbufs/widgets in the buffer have been modified since the iterator was created.You must use marks, character numbers, or line numbers to preserve a position across buffer modifications.You can apply tags and insert marks without invalidating your iterators,but any mutation that affects 'indexable' buffer contents (contents that can be referred to by character offset)will invalidate all outstanding iterators.
2. Gtk:ERROR:/build/buildd/gtk+2.0-2.20.1/gtk/gtktextlayout.c:2349:IA\_\_gtk\_text\_layout\_get\_line\_display: code should not be reached
3. Gtk:ERROR:/build/buildd/gtk+2.0-2.20.1/gtk/gtktextlayout.c:1361:set\_para\_values: code should not be reached
4. (<unknown>:2556): Pango-CRITICAL \*\*: pango\_layout\_get\_iter: assertion `PANGO\_IS\_LAYOUT (layout)' failed
5. Gtk:ERROR:/build/buildd/gtk+2.0-2.20.1/gtk/gtktextview.c:3571:gtk\_text\_view\_validate\_onscreen: assertion failed: (text\_view->onscreen\_validated)
6. (<unknown>:2903): Gtk-CRITICAL \*\*: gtk\_text\_layout\_wrap\_loop\_start: assertion `layout->one\_style\_cache == NULL' failed
7. (<unknown>:2903): Gtk-CRITICAL \*\*: gtk\_text\_layout\_wrap\_loop\_end: assertion `layout->wrap\_loop\_count > 0' failed
8. Gtk:ERROR:/build/buildd/gtk+2.0-2.20.1/gtk/gtktextlayout.c:2349:IA\_\_gtk\_text\_layout\_get\_line\_display: code should not be reached
9. cliente: malloc.c:5152: malloc\_consolidate: Assertion `nextchunk->fd\_nextsize->bk\_nextsize == nextchunk' failed.
10. \*\*\* glibc detected \*\*\* /home/santy/Documents/workspace/ChatCliente/Debug/ChatCliente: free(): invalid next size (normal): 0x08274320 \*\*\*
11. \*\*\* glibc detected \*\*\* /home/santy/Documents/NetBeans/Cliente/dist/Debug/GNU-Linux-x86/cliente: free(): invalid next size (fast): 0xb6d00500 \*\*\*