



Facultad de Ingeniería
Departamento de Ingeniería y
Ciencia de la Computación.
Universidad de Concepción



Proyecto de Redes

"Sistema de respaldos"

Integrantes : Donald Inostroza
Mauricio Cleveland

Profesor : Jorge Lopez

Introducción

Las aplicaciones dirigidas a la comunicación en redes han surgido de manera explosiva, sobre todo con el nuevo concepto de programación en la nube, desplazando de alguna manera (o quitándole protagonismo) a las aplicaciones de escritorio. En este sentido queremos presentar un trabajo que mezcla ambas partes.

Comúnmente almacenamos información crítica en nuestros computadores: documentos, archivos de configuración, imágenes y todo tipo de elementos. Con respecto a estos datos llamados críticos, no hay una "costumbre" de realizar respaldos periódicos y sostenidos en el tiempo. Por eso y a modo de prototipo proponemos un sistema que realice copias de respaldo automático, manteniendo la información original en el directorio local del usuario y realizando copias en segundo plano en un servidor que se encuentre en una red de área local o en Internet.

Análisis del problema

El problema principal radica en la detección de los eventos que realiza el usuario con los documentos o elementos, para luego tomar acciones específicas con respecto a tales eventos (por ejemplo la creación de un archivo). Luego debemos definir sobre que archivos queremos observar los eventos descritos anteriormente y entonces de alguna manera limitar el "radio de acción" del programa.

Una problemática que es de vital importancia, es la que se refiere al manejo de contingencias. Con esto nos referimos a aquellas eventualidades que debemos considerar en el momento de una falla o problemas que son ajenos al programa propiamente tal. Es el caso de no contar con una conexión permanente con el servidor, en donde los cambios detectados deben ser almacenados de alguna manera para su posterior revisión y ejecución.

También debemos realizar operaciones eficientes, la idea es no sobrecargar el tráfico de la red. El programa debe ser lo suficientemente "inteligente" como para descartar la transferencia de archivos en los casos que no sean necesarios.

El programa debe presentar integridad en los datos, pues de eso se trata un software de respaldos.

Diseño

Por las características del problema requeriremos un protocolo que nos brinde seguridad e integridad en los datos. Por ello utilizaremos el protocolo de comunicación TCP.

La respuesta a nuestro primer obstáculo lo encontramos en una característica del Kernel llamada *inotify*, implementada desde la versión 2.6.13. Inotify es un subsistema que informa de los cambios realizados en los archivos del sistema operativo, trabaja sobre inodos.

La arquitectura seleccionada será "Cliente-Servidor".

El programa cliente supervisará un directorio que puede ser definido arbitrariamente por el usuario. El programa encargado de los respaldos en cambio, tendrá un directorio establecido previamente donde se realizarán todas las operaciones requeridas.

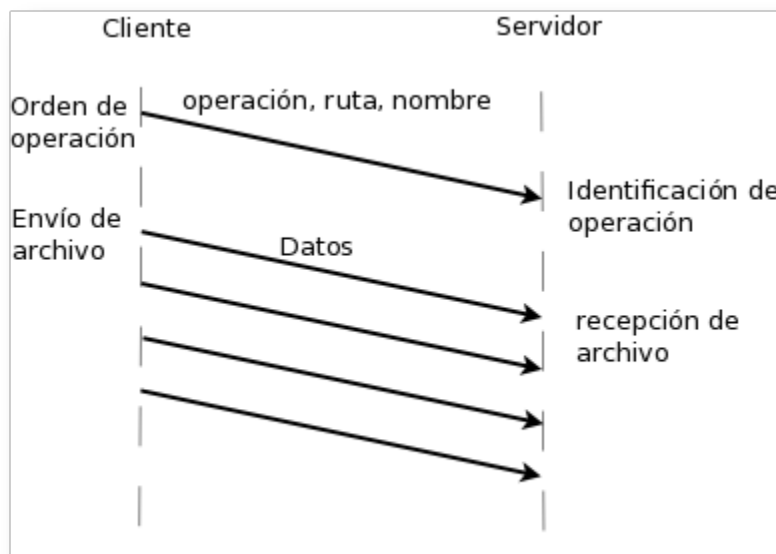
Las acciones que serán supervisadas son:

1. Creación.
2. Eliminación.
3. Modificación.
4. Movimientos

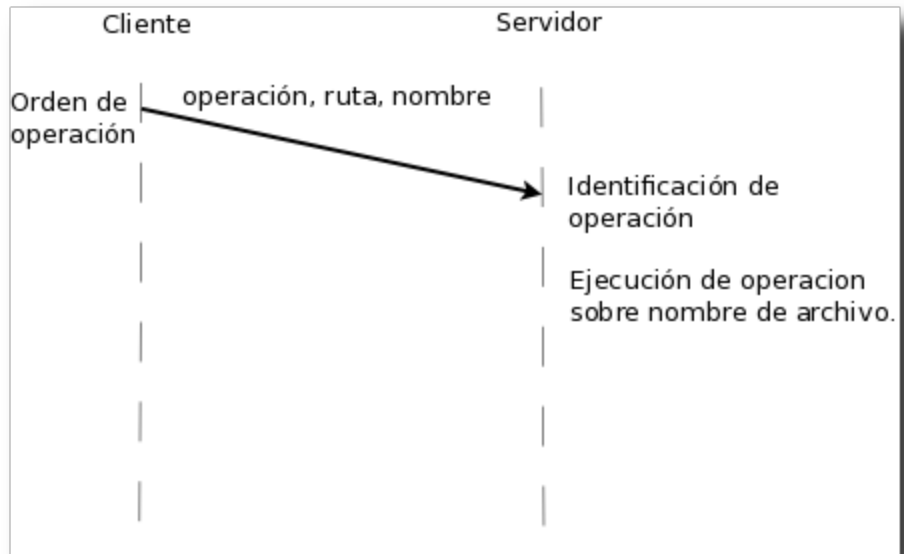
Para el caso de Creación y modificación los elementos (archivos) en el directorio local, serán enviados completamente hacia el servidor de respaldo. Si estos elementos son considerados directorios solo se enviará una orden para la creación o modificación por medio de comandos en el servidor.

Los eventos de movimiento serán utilizados para detectar cambios de nombre en los elementos y así realizar las tareas descritas anteriormente.

Entonces, veamos el caso de envío de archivo:



Caso de operación:



Existen casos en donde no necesitamos enviar los archivos o carpetas a través de la red, por ejemplo si el archivo en el directorio local solo fue visualizado, o mas recurrentemente, si al elemento se le ha cambiado el nombre, el programa debe darse cuenta de esto y no eliminar el archivo en el servidor y enviar el "aparentemente" nuevo elemento.

Implementación

El programa lo desarrollaremos en el lenguaje interpretado Python, por ser multi-plataforma y de sintaxis intuitiva, además existe la implementación de un modulo python que interactúa con el kernel e inotify.

Precisamente el modulo cuenta con una variada selección de eventos particulares, los cuales serán de bastante utilidad para filtrar los eventos que sean importantes.

Alguno de ellos son:

Nombre del evento	Descripcion
IN_CLOSE_WRITE	Se escribió el archivo al cerrarlo.
IN_CREATE	Creación de archivo o directorio.
IN_DELETE	Borrado de archivo o directorio.
IN_MODIFY	Archivo modificado.
IN_MOVED_FROM	El archivo o directorio fue movido desde origen.
IN_MOVED_TO	Destino de archivo o directorio que fue movido.

Al detectar uno de estos eventos, el modulo nos indica la ruta absoluta y el nombre del archivo en el cual ocurrió el evento mediante dos simples variables.

Además podemos seleccionar un directorio arbitrario en donde queremos realizar la supervisión de los eventos.

El desarrollo del software será realizado íntegramente en Linux y para ser usado también en Linux en versiones posteriores al kernel 2.6.13.

Para que la aplicación pueda funcionar es necesario instalar en la maquina cliente el modulo "*pyinotify*", esta es la pieza clave del programa. El resto de los módulos de Python están incluidos por defecto en el sistema.

Para la comunicación entre el cliente y el servidor ocupamos la implementación de socket que posee Python. A diferencia del lenguaje de programación C, en donde podemos enviar estructuras completas desde un host a otro, en Python (que se trabaja con objetos) debemos realizar una transformación de las instancias antes de enviarlas por la red.

Para esto ocupamos el modulo *Pickle*, que es un método para serializar este tipo de elementos dentro del lenguaje. Luego el host destino tiene que realizar la tarea inversa para poder ocupar los métodos asociados. Una curiosidad de este sistema es que no necesitamos definir las clases en el host destino, una vez que se recibe simplemente se utiliza el objeto.

Para las tareas donde no se necesite enviar los datos por la red acudiremos a un modulo que interactúa con los comandos del sistema operativo y permite realizar operaciones básicas sobre archivos: crear, eliminar, renombrar...etc. El aludido módulo se llama *os*, y listamos algunas de sus funciones:

`os.mkdir(nombre)` : Creación de directorio.

`os.remove(nombre)` : Eliminación de archivo.
`os.path.isdir(nombre)`: Identificación de directorio.

También hemos incluido el comienzo de la sección que registra las operaciones que se efectúan fuera de línea, creando un archivo de texto que podrá ser serializado en un futuro.

Manual del programa

Instalación de módulos

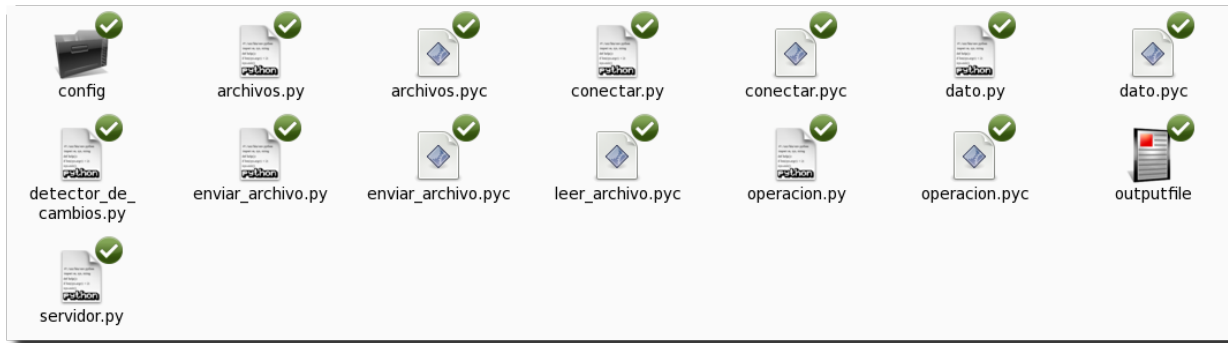
Se debe instalar el módulo adicional *pyinotify*, el cual no está incluido en las distribuciones. Para hacerlo debemos dirigirnos al gestor de paquetes que utilizemos y buscar tal módulo como:

"python-inotify"

Para instalarlo debemos contar con los permisos necesarios, de lo contrario debe contactarse con el administrador del sistema.

Ejecutando el programa

Los archivos que componen el programa son los que se muestran en la imagen:



El código fuente lo componen los archivos con extensión "py". De ellos los ejecutables son:

Servidor: **servidor.py**

Cliente: **detector_de_cambios.py**

El directorio "**config**" corresponde al lugar en el cual se realizan las copias de seguridad del cliente.

Configuración

La configuración del programa cliente está dada por la penúltima línea de código que se muestra en la imagen:

```
p = Ptmp()
notifier = Notifier(wm, p)
wdd = wm.add_watch('/home/cleve/Descargas/test', mask, rec=True, auto_add=True)
notifier.loop()
```

En donde la cadena que está entre comillas y marcada corresponde al directorio (absoluto) que queremos observar.

Ejecutando el programa

Antes de ejecutar la aplicación, debemos asignarle los permisos necesarios. Para ello ejecutamos las siguientes ordenes en un terminal:

```
chmod +x detector_de_cambios.py
chmod +x servidor.py
```

Luego para ejecutar ambos programas se deben escribir las ordenes para el servidor y cliente respectivamente:

./servidor

./detector_de_cambios

Descripción de archivos

servidor.py: Archivo único que efectúa los respaldos del programa cliente.

detector_de_cambios.py: Archivo ejecutable del cliente. Se encarga de notificar los eventos sobre el directorio indicado.

archivos.py: Están definidas las operaciones sobre archivos: lectura, escritura (Módulo).

conectar.py: Se encarga de realizar el trabajo sobre socket (Módulo).

dato.py: Archivo destinado para uso futuro (Módulo).

enviar_archivo.py: Es el encargado de enviar las operaciones y datos al servidor. Contiene el serializador capaz de transferir objetos.

operacion.py: PDU con el registro de la operación deseada.

Limitaciones

La gran limitación de nuestro programa es que a pesar de que el lenguaje es multi-plataforma, la implementación de pyinotify no lo es. Esto debido a la característica particular de inotify presente solo en el Kernel. Un desarrollo en plataformas Windows por ejemplo, traería consigo un nuevo diseño para el manejo de eventos que finalmente es el eje central de la aplicación.

Siempre que hablamos de Python, nos referimos a un lenguaje interpretado y todos los problemas de rendimiento que esto acarrea. Un lenguaje de este tipo es lento y si bien en nuestra aplicación no tiene importancia el tema de velocidad, si eventualmente quisiéramos ampliar el sistema con un soporte ilimitado de clientes seguramente deberíamos cambiar el modelo por lo menos en el lado del servidor.

La propia aplicación tiene limitantes que imposibilitan a esta ser considerada como estable, estas características no presentes son:

- Trabajo con sub-directorios.
- Soporte para mas de un cliente.
- Trabajo solamente en linea.
- Imposibilidad de deshacer cambios.
- Operaciones masivas de archivos.

Lo que sigue

Implementar el trabajo con sub-directorios e introducir el manejo de operaciones pendientes en el caso de que se produzca un error de comunicación. Además es altamente deseable incluir una interfaz amigable que permita configurar y verificar el estado del programa de manera fácil e intuitiva (por ejemplo la selección del directorio a supervisar en el caso del cliente).

Conclusiones

Para realizar aplicaciones de escritorio no es requisito (en el caso de este proyecto) una velocidad de respuesta alta si lo es para servidores que atienden a muchos clientes. Bajo esta premisa no sería lo mas adecuado implementar este proyecto a gran escala con el modelo de servidor que presentamos.

La programación sobre socket brinda un sin fin de soluciones y abstracciones que hacen mucho mas fácil el enfoque del problema. Esto nos facilita el desarrollo de soluciones que ya están implementadas, como es el caso de TCP.

Bibliografía

<http://www.python.org>
<http://docs.python.org>
<http://en.wikipedia.org/wiki/Inotify>
<http://trac.dbzteam.org/pyinotify>
<http://es.wikipedia.org/wiki/Inodo>