Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Descripción y Planificación

El presente capítulo describe en líneas generales el ámbito funcional del proyecto, para delimitar su alcance, la metodología a usar durante el desarrollo del sistema y los requisitos de alto nivel del mismo, extraídos del ámbito funcional descrito.

≪a completar≫

Contents

1.1. Descripción Funcional del Sistema

La presente sección describe el ámbito funcional del sistema software que deseamos construir, es decir, del *DrManhattan*, que es un *Sistema de Control de Accesos a Red para los Laboratorios de la Facultad de Ciencias*.

El objetivo del *DrManhattan* es controlar el acceso a la red local, existente en cada laboratorio de la Facultad de Ciencias de la Universidad de Cantabria, de los computadores conectados a ella. Se desea realizar dicho control sobre todo durante la realización de pruebas evaluables, de cara a evitar que se realicen accesos a contenidos no autorizados (e.g., páginas de internet con posibles soluciones a los problemas planteados, el directorio de trabajo de algún compañero, etc.) durante la realización de dichas pruebas. El sistema que se venía utilizando hasta ahora para evitar el mal uso de la red local consistía simplemente en desconectar la alimentación del concentrador de interconexión, deshabilitando la red. Es decir, se aplicaba el principio de muerto el perro, se acabó la rabia.

No obstante, el tener los diversos computadores interconectados mediante una red local, no sólo tiene inconvenientes, sino que también posee varias ventajas. Por ejemplo, ayuda a facilitar la distribución de material electrónico que resulte necesario para la realización del examen. También puede ser de gran utilidad para recoger los ejercicios realizados por los alumnos de una manera rápida y cómoda, ya que el alumno sólo tendría que enviar el material producido durante la prueba al computador o dirección que le indique el profesor. El método utilizado actualmente para realizar esta tarea consiste en que el profesor acude al computador del alumno cuando este desea entregar los ejercicios realizados y copia tales ejercicios en una memoria USB. Este proceso es lento, tedioso y en muchas ocasiones las memorias USB no son reconocidas por los computadores del laboratorio, lo que exige recurrir a otras técnicas. El tiempo empleado por los docentes para recopilar los ejercicios realizados por los alumnas suele oscilar entre la media hora y la hora completa, mientras que usando la red local dicho proceso podría realizarse en cuestión de minutos (cuando no menos).

Por tanto, el acceso de los computadores a la red local (e internet), se deberá controlar desde un computador distinguido, al que llamaremos *Watchman*, que será normalmente el computador asignado al docente. Dicho computador será el encargado de conceder y denegar el acceso a la red al resto de computadores conectados a la red local.

Se desea que el acceso a la red esté habilitado en las siguientes circunstancias:

1. Al comienzo de la realización de las pruebas, de forma que el docente pueda distribuir material electrónico necesario o de interés para la realización de la prueba (e.g., manuales, el enunciado de la prueba, etc.) entre los diferentes computadores de forma cómoda y eficiente.

2. Cuando el alumno haya finalizado la prueba, de forma que se puedan enviar los resultados a través de la red, evitando el penoso proceso de tener que recogerlos de forma individual mediante copia en un memoria USB o dispositivo similar. En estos casos, sería además deseable, con objeto de evitar posteriores problemas, que el sistema comprobase la integridad de los archivos recibidos, es decir, que comprobase que no se han sufrido alteración alguna durante la transmisión.

Durante el periodo de tiempo que un alumno esté realizando una prueba, se debe denegar el acceso a la red del computador que esté utilizando para la realización de la prueba.

Por tanto, la secuencia de realización de una prueba evaluable sería tal como sigue:

1. El profesor y los alumnos acceden al aula y encienden sus correspondientes computadores. El acceso a la red está habilitado para todo el mundo.

2. El profesor envía el material necesario para la realización de la prueba a los computadores de los alumnos.

3. A continuación, una vez que un alumno comienza la prueba, se le deniega el acceso a la red al computador que esté utilizando.

4. Si durante la realización de la prueba un alumno considera que ha acabado y está satisfecho con los resultados producidos, indicará al sistema que ha concluido la prueba. Los ficheros producidos como material evaluable se enviarán al *Watchman*. Se comprobará que se han recibido correctamente y no están corruptos, por lo que se pueden abrir y leer sin problema alguno. Obviamente, para el envío de los ficheros de resultados habrá que habilitar de nuevo la red, pero se ha de evitar que el alumno pueda modificar dichos ficheros.

5. Debe existir también la posibilidad de que el docente decida que el tiempo de realización de la prueba ha concluido, por lo que deberá realizarse todo el proceso descrito en el punto anterior, pero para todos los computadores que se encuentren activos en ese momento y siendo el proceso iniciado desde el *Watchman*, en lugar de desde el propio computador del alumno.

Además, se guardarán registros de los eventos ocurridos en cada prueba, con el objetivo de tener información que permita auditar el transcurso de la misma. Dicho registro debe mantener constancia de eventos tales como hora de inicio de la prueba, hora de entrega de cada ejercicio, número de ficheros que se entregan, etc.

Para comodidad del alumno, en las pruebas con hora de finalización predeterminada se mostrará en la interfaz de la aplicación los minutos restantes hasta el final.

La siguiente sección detalla la metodología de desarrollo que usaremos para la construcción de este sistema.

1.2. Metodología de Desarrollo

≪Describe brevemente la metodología en plan abstracto≫

En esta sección se describe la metodología utilizada para desarrollar el sistema.

Se ha escogido el proceso de desarrollo *iterativo e incremental* que suele ser parte esencial en las metodologías de desarrollo ágiles.

La idea principal de este proceso se basa, como su nombre, en iteraciones e incrementos, que no son lo mismo. El software se construye mediante tareas repetitivas, *iteraciones*, en las que se añaden nuevas funcionalidades progresivamente, *incrementos*, para crear una versión del sistema que cumple más requisitos que la anterior. Se realizan tantas iteraciones sean necesarias hasta que todos los requisitos se hayan implementado y, por tanto, el sistema este finalizado.

Es decir, una iteración es un mini-proyecto en el que se obtiene una versión de cada una de las piezas del sistema, sean código o no, y un incremento se puede medir como la diferencia entre una iteración y la anterior.

El proceso se divide en cuatro partes fundamentales:

1. **Iniciación:** en esta fase se describe el ámbito funcional del sistema, los requisitos de alto nivel y se identifican posibles riesgos. Hay que comprender el sistema y sus límites.

2. **Elaboración:** el objetivo principal de esta fase es el de crear la arquitectura básica, para tener una visión de cómo será el sistema completo y además, proponer soluciones a los riesgos identificados.

3. **Construcción:** cómo su nombre indica, esta es la fase dónde se construye y prueba la aplicación. Se realiza un pequeño proceso en cascada en cada iteración de esta fase.

4. **Transición:** se despliega el sistema y finaliza el proceso.

Cada una de estas fases, como es lógico, puede tener más de una iteración, en función del tamaño del proyecto, con tareas propias de la etapa en que se esté, así por ejemplo, en la etapa de construcción, en cada iteración se escoge un grupo de requisitos de alto nivel, se refinan, y partiendo de la versión del sistema obtenida en la iteración anterior, se diseñan, implementan y prueban esos requisitos, creando la versión que se utilizará en la siguiente iteración.

Al finalizar el proceso obtenemos tanto el código de la aplicación, como modelos y documentación de la misma que se van creando a lo largo de las iteraciones.
≪añadir imágenes del ciclo de vida≫

Las principales razones por las que se ha escogido este tipo de proceso y no otro son:

- ✦ Poca incertidumbre a la hora de diseñar y planificar, ya que al realizarse en cada iteración y no del sistema completo, se tiene mayor comprensión de los riesgos y de las tareas a realizar.
- ✦ Se adapta bien a cambios de requisitos.
- ✦ Si se priorizan los requisitos para implementarlos en las primeras iteraciones de la construcción, se obtiene una aplicación con funcionalidades clave en la que el cliente puede decidir si está o no correcto.
- ✦ Al trabajar con un subconjunto de requisitos en cada iteración la complejidad se reduce, lo que facilita que se reduzca también el número de errores producidos.

De la primera fase, *iniciación*, la descripción funcional detallando el alcance del sistema está descrita en la sección 1.1, en la siguiente se describen los requisitos de alto nivel extraídos.

1.3. Requisitos de Alto Nivel del Sistema

Esta sección describe el segundo paso en nuestro proceso de desarrollo, de acuerdo a la metodología descrita en la sección anterior, que es la identificación de los requisitos de alto nivel que ha de satisfacer nuestro sistema software, de acuerdo a la descripción del ámbito funcional proporcionada en la Sección 1.1.

En concreto, se han identificado los siguientes requisitos de alto nivel para nuestro sistema

≪arreglar la tabla≫

Identificador	**Descripción**
R01	Un computador de la red debe poder ser designado como *Watchman*.
R02	Todos los computadores que no sean *Watchman* serán computadores normales, y po
R03	El profesor debe ser capaz desde el *Watchman* de indicar el inicio de la prueba.
R04	El profesor debe ser capaz desde el *Watchman* de indicar el fin de la prueba.
R05	El profesor ha de ser capaz desde el *Watchman* de establecer una hora límite para l
R06	El profesor debe ser capaz desde el *Watchman* de enviar el fichero de enunciado al r
R07	El alumno desde su computador debe ser capaz de conectarse al *Watchman*.
R08	El alumno desde su computador debe ser capaz de enviar sus resultados al profesor.
R09	El alumno desde su computador debe ser capaz de indicar que da por finalizada la p
R10	El alumno desde su computador debe tener la posibilidad de ver el tiempo restante
R11	La aplicación del alumno ha de ser capaz de denegar el acceso a la red al empezar l
R12	La aplicación del alumno ha de ser capaz de permitir el acceso a la red al finalizar l
R13	La aplicación ha de ser capaz de comprobar que los archivos se han enviado correct
R14	La aplicación del profesor ha de ser capaz de guardar registros de actividad.

Capítulo 2

Antecedentes

El siguiente capítulo describe brevemente las tecnologías sobre las que se fundamenta el presente proyecto. Más concretamente, se explica el funcionamiento de las redes locales, del framework Netfilter para el filtrado de paquetes, y de los demonios en los sistemas Linux, tres aspectos fuertemente relacionados con el proyecto.

La aplicación a desarrollar está pensado que se utilice en los laboratorios de la Facultad de Ciencias de la Universidad de Cantabria, en estos laboratorios los computadores están conectado mediante una *red local* y utilizan sistemas Linux. Para gestionar los permisos de acceso a la red se va a utilizar un *demonio* que interactúe con *Netfilter* por medio de iptables.

Contents

2.1. Red local

Una *Red local* o LAN, siglas en inglés de Local Area Network, es un conjunto de computadoras conectadas entre sí en un área relativamente pequeña, como los laboratorios de la Facultad.

Cada uno de estos equipos interconectados en la red se conoce como nodo. Estos nodos son capaces de enviar, recibir y procesar comandos con el fin de transportar datos, así como compartir información y recursos a través de la red.

El funcionamiento de la red está estandarizado siendo el protocolo TCP/IP el más extendido.

2.2. Demonio

Un *demonio* (del inglés, *daemon*) es un tipo de proceso que posee la siguientes características:

1. Se ejecuta en segundo plano;
2. Generalmente se inicia en tiempo de arranque;
3. No usa los sistemas de entrada/salida estándar;
4. Mantienen la información que necesitan en ficheros especiales bien identificados.

Normalmente están cargados en memoria esperando una señal para ser ejecutados, por lo que su gasto de recursos no suele ser significativo.

Un ejemplo claro de demonio es *httpd*, que se ejecuta en los servidores web. El nombre viene de *HTTP D*aemon y es utilizado para aceptar peticiones, una vez que las acepta, crea otros procesos se encargan de atenderlas.

En el proyecto se implementa un demonio para interactuar con Netfilter, descrito en la siguiente sección.

2.3. Cortafuegos : Netfilter

Netfilter es un framework que permite filtrar de paquetes, traducción de direcciones y puertos de red y varias funcionalidades más para el manejo de paquetes. Es parte del núcleo de linux desde la versión 2.4 del mismo, sustituyendo a ipchains, bastante limitado en comparación con Netfilter.

Para interactuar con Netfilter una de las aplicaciones más usadas es *iptables*, siendo necesarios permisos de administrador para ello.

Ejemplo de uso de iptables:

```
# iptables -A INPUT -s 195.65.34.234 -j ACCEPT
```

El parámetro -A indica que se va a añadir una regla, el objetivo de la mismo es aceptar todos los paquetes entrantes provenientes del host indicado[1]. Del mismo modo, si lo que queremos es no aceptar las peticiones se cambiaría ACCEPT por DROP y si nos queremos referir a los paquetes salientes OUTPUT por INPUT. Si tenemos iptables con la configuración por defecto, se aceptan todos los paquetes entrantes y salientes, no hay ninguna restricción.

[1] en vez de la IP podemos poner su FQDN (*Fully Qualified Domain Name)* si lo deseáramos