

Manual de proyecto

Mapache servidor web.

7542. Taller de programación I

GRUPO N° 1

Integrantes

Apellido y nombre	Padrón
Castarataro, Pablo Daniel	89683
Gramajo, Gustavo	88094
Lopez, Edgardo	86140

Índice de contenido

Manual de proyecto.....	1
Integrantes.....	1
División de tareas.....	5
Evolución de proyecto.....	6
Análisis de puntos pendientes.....	7
Herramientas.....	7
Conclusiones.....	8
Documentación técnica.....	8
Requerimientos de software.....	8
Descripción general.....	8
Descripción de los módulos.....	9
Modulo Common.....	9
Descripción general.....	9
Submodulos.....	9
Diagramas.....	10
Módulo Thread.....	10
Descripción general.....	10
Clases.....	10
Thread.....	10
Métodos y funciones.....	10
Mutex.....	11
Métodos y funciones.....	11
Lock.....	11
Métodos y funciones.....	11
Diagramas.....	12
Modulo Sockets.....	12
Descripción general:.....	12
Clases.....	12
TCPSocket.....	12
Métodos y funciones.....	12
Diagramas.....	14
Modulo Configuración.....	14
Descripción general.....	14
Clases.....	14
Configuración.....	14
Métodos y funciones.....	14
Diagramas.....	17
Módulo Protocolo de control.....	18
Descripción general.....	18
Clases.....	18
ProtocoloControl.....	18
Métodos y funciones.....	18
Diagramas.....	19
Modulo Servidor.....	19
Descripción general.....	19
Clases.....	20
Servidor.....	20
Métodos y funciones.....	20

ServidorHTTP.....	20
Métodos y funciones.....	20
AdministradorClientes.....	21
Métodos y funciones.....	21
VerificadorTimeOut.....	21
Métodos y funciones.....	21
ManejadorClienteHTTP.....	22
Métodos y funciones.....	22
ProcesadorRequest.....	22
Métodos y funciones.....	22
Iprocesador.....	23
Métodos y funciones.....	23
ProcesadorTipoEstatico.....	23
Métodos y funciones.....	23
ProcesadorCGI.....	23
Métodos y funciones.....	23
Diagramas.....	24
Protocolos.....	25
Protocolo de control del servidor.....	25
Protocolo HTTP.....	25
Protocolo CGI.....	25
Modulo Protocolo HTTP.....	25
Descripción general.....	25
Clases.....	25
Protocolo HTTP.....	25
Métodos y funciones.....	25
HTTP_Response.....	26
Métodos y funciones.....	26
HTTP_Request.....	26
Métodos y funciones.....	26
Diagramas.....	26
Protocolos.....	26
Modulo Configurador.....	27
Descripción general.....	27
Submodulos.....	27
Diagramas.....	27
Modulo Gui.....	28
Diagrama.....	28
Descripción general:.....	28
Clases.....	28
VentanaPrincipal.....	28
Métodos y funciones.....	28
BarraDeEstado.....	28
Métodos y funciones.....	28
Solapas.....	29
Métodos y funciones.....	29
GrillaDinamicos, GrillaEstaticos, GrillaErrores, GrillaUsuarios.....	30
Métodos y funciones.....	30
ControladorGrilla.....	30
Métodos y funciones.....	30
VbVistaAcceso, VbVistaError.....	31
Métodos y funciones.....	31

ClienteControl.....	31
Métodos y funciones.....	31
Modulo Precompilador.....	31
Descripción general:.....	31
Clases.....	32
ContenedorDeDefines.....	32
Métodos y funciones.....	32
FlagsDePrograma.....	32
Métodos y funciones.....	33
IfDefAnidados.....	33
Métodos y funciones.....	33
Precompilador.....	33
Métodos y funciones.....	33
Procesador.....	34
Métodos y funciones.....	34
SeparadoresValidos.....	34
Métodos y funciones.....	34
Utilitarias.....	34
Métodos y funciones.....	34
Modulo CGI_Precompilador.....	35
Descripción general.....	35
Clases.....	35
ParserEntradaEstandar.....	35
Métodos y funciones.....	36
Modulo CGI_Pr-Zip.....	36
Descripción general.....	36
Clases.....	37
ParserEntradaEstandar.....	37
Modulo CGI_ParserPHP.....	37
Descripción general.....	37
Clases.....	37
ProcesadorPHP.....	37
Métodos y funciones.....	37
Programas intermedios y programas de prueba.....	38
Manual de usuario.....	39
Instalación.....	39
Requerimientos de software.....	39
Requerimientos de hardware.....	39
Proceso de instalación.....	39
Configuración.....	39
Forma de uso.....	41
Apéndice I.....	43
Pruebas Servidor.....	43
Ejemplos.....	43
Prueba 1.....	43
Prueba 2.....	44
Prueba 3.....	45
Prueba 4.....	45
Prueba 5.....	46
Apéndice II.....	48
Pruebas del Precompilador:.....	48
CGI Precompilador ZIP.....	52

Modo de uso y ejemplos.....	52
Código fuente.....	52

División de tareas

		Pablo	Gustavo	Edgardo
Martes	10/11/2011			
Semana 1		instalación php, apache.	instalación php, apache.	instalación php, apache.
		Diseno formato xml	Diseno formato xml	Diseno formato xml
		Diseno de clases del servidor	Diseno de clases del servidor	Diseno de clases del servidor
		Lectura de especificaciones html	Lectura de especificaciones html	Lectura de especificaciones html
		Lectura de especificaciones http	Lectura de especificaciones http	Lectura de especificaciones http
		Lectura de especificaciones gtkmm	Lectura de especificaciones cgi	Lectura de especificaciones cgi
		Lectura de especificaciones tiny xml		Lectura de especificaciones gtkmm
		Repositorio SVN		Grupo google
		Serializacion xml guardar configuración		implementar configurador
		Pagina php test		
		implementación configuracion xml		
Martes	10/18/2011			
Semana 2		implementación configuracion xml	Cgi	integración configurador - XML
		Clases servidor.	Pruebas CGI en APACHE	socket
		Protocolo ClienteControl		thread servidor
		implementar clienteControl		integración configurador - Cliente Control
				protocolo http 1.0
				Lectura libreria log4cpp
				implementacion modulo log
Semana 3				servidor
Martes	10/25/2011			
Semana 3		socket	Cgi	protocolo http 1.0
		thread servidor	Clase Procesador CGI	pruebas http 1.0
		Diagramas de clases	parserPHP	servidor
		servidor		
Semana 4				
Martes	11/1/2011			
Semana 4		servidor	Integración parserPHP-servidor	protocolo http
			Integración cgi-servidor	servidor
Martes	11/8/2011			
Semana 5		Pruebas de módulos	Pruebas de módulos	Pruebas de módulos
		Integración de módulos	Integración de módulos	Integración de módulos
Martes	11/15/2011	Pre entrega		
Semana 6		optimización	optimización	optimización
		Documentación	Documentación	Documentación
Martes	11/22/2011	Manuales de usuario	Ayuda usuario	
Semana 7		Corrección de bugs	Instalador	Corrección de bugs
			CMake	
			Generador de documentación	
Martes	11/29/2011	Entrega final		

Evolución de proyecto

El proyecto evoluciona según el cronograma indicado, excepto en la semana 2 y 3 en las cuales no se llegó a terminar lo estimado en el cronograma, Implementación Cgi, protocolo http, los mismos fueron solucionados en la semana siguiente cumpliendo luego con el cronograma indicado.

Inconvenientes encontrados

- ¿Que hacer con los archivos que generan los programas cgi?

Inicialmente se planteó la posibilidad de hacer una tabla de hashing con los nombres de los archivos generados y ponerlos a disposición del sistema para que cada instancia de ejecución del cgi pueda accederlos.

Finalmente se optó por un sistema de creación y eliminación de subcarpetas en arch-temp-cgi. Este consiste en:

- Se crea una subcarpeta, de nombre X, donde X es el numero de cgi ejecutado en el sistema desde su inicio.
- El cgi tomará como ámbito de trabajo el directorio de X, creará y buscará recursos en esa ruta.
- Finalmente se eliminará la carpeta, luego de que el sistema obtenga el retorno del cgi.

- Reutilización de los puertos

Encontramos problemas relativos al reinicio del servidor y a los puertos involucrados en este proceso. Se solucionó agregando un flag al socket cuando se está reiniciando un servidor.

- Cierre de sockets

- Parseo de argumentos en archivos php.

Se solucionó imponiendo una manera específica de manejar variables en los archivos php.

Veamos un ejemplo (se resalta en cursiva la manera de manejar las variables):

```
<html>
<body>
<form action="pr.cgi" method="post" enctype="multipart/form-data"><br>
<?php
$shortopts  = "";
$longopts  = array(
    "cantArchivos:",    // Valor obligatorio - invocar desde consola así: php test.php
                        --cantArchivos 2
);
$options = getopt($shortopts, $longopts);
for ($i=1; $i<=$options["cantArchivos"]; $i++)
```

```

{
    echo "<form method=\"post\" enctype=\"multipart/form-data\"><br>\n";
    echo "<label for=\"file\">Archivo </label>\n";
    echo $i;
    echo "<input type=file size=60 name=\"file\"><br>\n";
}
?>
<input type=submit value=">>"><br>
</form><br>
</body>
</html>

```

Adicionalmente, la variable de entorno `URI_PARA_PARSER_PHP_ALTERNATIVO` contiene el uri del recurso, la cual podría ser usada para parsear el archivo de otra forma.

Análisis de puntos pendientes

La gran mayoría de estos puntos pendientes quedaron relegados por poseer una menor relevancia relativa:

1. Brindar la posibilidad de elegir en que ruta se quiere instalar el sistema, para , si el usuario lo desea, tener separados los binarios, archivos de configuraciones, sitios, documentación auto-generada, ayuda, etc, del código fuente.

Solución: modificación del archivo `CMakeLists.txt` mediante la adición de los comandos adecuados (`INSTALL`).

2. Chequear las extensiones de los archivos fuente que procesa el precompilador `cgi`.

Solución: comprobación de la extensión a la hora del parseo del cuerpo (post) de un request.

3. Eliminacion de los archivos `doxy` de configuracion que generamos con `Cmake`.

Al instalar, hacemos uso de un archivo de configuracion generico. Para cada modulo, se copia el generico a la ruta donde estan los fuentes y se reemplazan ciertos tags (nombre del proyecto, directorio de salida de la documentacion, directorio donde estan los fuentes), obteniendo asi un archivo de configuracion especifico.

Solucion: modificacion del archivo `CmakeLists.txt` para lograr que estos archivos se eliminen despues de la instalacion.

4. Mas relevancia y apariciones de la mascota/cara visible de nuestro sistema (Mario Mapache) en la gui de `ConfiguradorCliente`.

Herramientas

`GCC` – Compilador utilizado para la compilación del proyecto

`Eclipse IDE` – IDE para programación en `C++`

`Cmake-gui/Cmake` – Generador de `MAKEFILES` automático.

`Valgrind` – Chequeo de memoria, recursos, etc.

Svn Google Code – Hosteo ONLINE de proyectos

RapidSVN – Manejo de SVN

Doxygen – Documentación automática.

Glade – Diseño de interfaz gráfica.

Astah – Diagramas UML.

Conclusiones

Luego de varias semanas de incesante esfuerzo, trabajo coordinado en equipo e investigación, creemos firmemente que hemos cumplido con creces todos los objetivos que nos impusimos.

El feedback semanal de nuestros ayudantes nos ayudo a no salirnos del camino, a despejar dudas y a abrir nuestras mentes a nuevas ideas.

Queremos agradecerles especialmente a nuestras familias por todo el apoyo y la comprensión que nos brindaron. Hemos estado tan ausentes y abstraídos en este, a veces intrincado, proyecto, pero aún así, ellos estuvieron ahí para nosotros. A ellos va dedicado Mapache 1.0.

Documentación técnica

Requerimientos de software

Se requiere tener instalado:

- libtinyxml 2.5.3 (para el manejo de las configuraciones del servidor)
- php5 (para el manejo de archivos php)
- cmake 2.8.3 (originalmente no se usaba, pero en las últimas etapas se volvió indispensable)
- doxygen (para generar documentación automáticamente)
- gtkmm-2.4 (librería para desarrollo de la interfaz gráfica de ConfiguradorMapache)
- Valgrind 3.6.1 (Opcional: si se desea verificar pérdidas de memoria).
- <http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml> (chequeo de normas de codificación)
- log4cpp (librería utilizada para el manejo de los archivos de LOG)

Documentación

- doxygen 1.7.3
- graphviz
- dot2tex

S.O.

Probado con resultados positivos en Ubuntu 10.10, 11.04, 11.10.

Descripción general

El trabajo práctico fue implementado a partir de la creación de distintos módulos. El primero de ellos fue el modulo Common. Este modulo será el que utilizarán el configurador y el servidor para poder manejar conexiones, hilos, xml. Paralelamente, se crearon los módulos necesarios para poder

precompilar y generar las salidas pertinentes respetando el protocolo CGI.

El trabajo tiene esta estructura:

Mapache

- common
 - threads
 - sockets
 - protocoloControl
 - configuraciones
- servidor
 - utils
 - protocoloHTTP
- cliente
 - gui
- precompilador
- pr
- pr-zip
- parserPHP

En la próxima sección se explicará cada uno de los módulos anteriores.

Descripción de los módulos

Modulo Common

Descripción general

Este módulo es una recopilación de otros módulos de menor tamaño que proveen utilidad tanto al servidor como al cliente.

Submodulos

Common esta compuesto por 4 submódulos:

1. Thread.
2. Sockets.
3. ProtocoloControl.
4. Configuración.

Mas adelante se describe a cada uno de ellos.

Diagramas

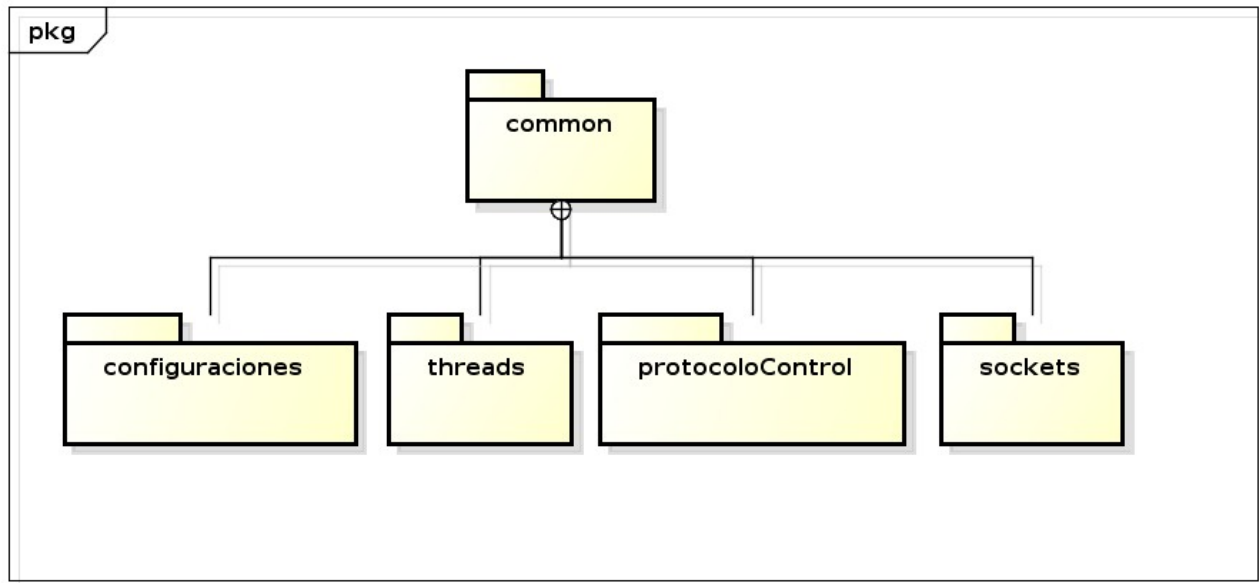


Diagrama de paquetes con los submódulos para Common

Módulo Thread

Descripción general

Este módulo es el utilizado para el manejo de los hilos de ejecución dentro de un programa. Se crea una abstracción utilizando polimorfismo que permite implementar nuestros propios hilos. Para ello solo se debe implementar un método para nuestra clase.

Mutex y Lock son la parte vital de nuestro sistema de protección de recursos ante la posible modificación de los mismos por parte de diversos hilos.

Clases

Thread

Se encarga de representar a un hilo dentro de la aplicación. Utiliza una abstracción realizada sobre la librería pthread

Métodos y funciones

void start()	Comienza con la ejecución del hilo
void join()	Joina el hilo. Queda bloqueado hasta que el hilo finaliza su ejecución.
void morir()	Cambia el estado del hilo a muerto.
bool vivo()	Devuelve el estado del hilo. True si esta vivo. False si ya murió.

protected: void run()	Este método es el que se debe implementar para
-----------------------	--

	que el hilo ejecute lo deseado.
--	---------------------------------

Para crear nuestros propios hilos sólo debemos implementar el método **run()**. Para controlar la vida de los mismo es posible utilizar los métodos **vivo()** y **morir()**.

Mutex

Clase utilizada para la protección de los recursos compartidos entre distintos hilos.

Métodos y funciones

Mutex()	Constructor. Crea un nuevo mutex que se encargará de la protección de algún recurso compartido de nuestro sistema.
~Mutex()	Destructor. Libera los recursos asociados al Mutex.
void lock()	Realiza el bloqueo del mutex. Una vez realizada esta operación el recurso estará protegido hasta que se realice el unlock()
void unlock()	Realiza el desbloqueo del Mutex. Una vez realizada esta operación el recurso deja de estar protegido.

Lock

Encargada de bloquear y desbloquear instancias de Mutex.

Métodos y funciones

Lock(Mutex&)	Constructor. Una vez creado el Lock se realiza el bloqueo del mutex asociado. Este bloqueo permanecerá por toda la vida del objeto Lock
~Lock()	Destructor. Realiza el desbloqueo del mutex asociado.

Diagramas

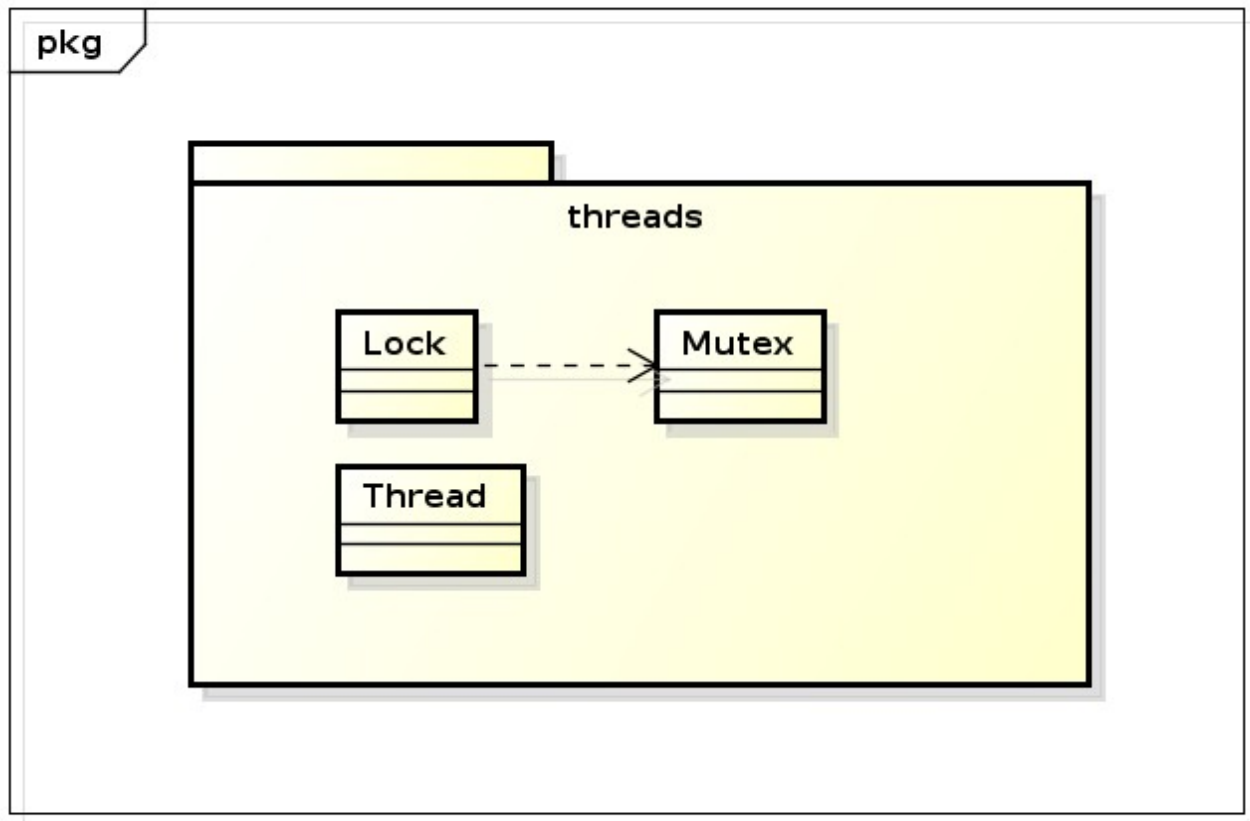


Diagrama de clases del Paquete Thread

Modulo Sockets

Descripción general:

Este paquete ofrece la funcionalidad necesaria para el manejo de conexiones desde un cliente o un servidor.

Clases

TCPSocket

Este módulo es el utilizado para el manejo de las conexiones entre distintos programas. Es una abstracción para C++ de la interfaz de Berkeley Sockets de POSIX.

Métodos y funciones

TCPSocket()	Constructor. Crea un nuevo socket de tipo TCP que podrá ser utilizado para escuchar conexiones o conectarse a algún servidor. En caso de no poder crear el socket, arroja InitException .
void setearComoReusable()	Agrega un flag al socket para que el sistema operativo ignore el TIME_WAIT. Este método es utilizado cuando se desea reiniciar un socket que se bindeará en un puerto recién liberado.

void bindear(int puerto)	<p>Le indica al socket el número de puerto donde deberá escuchar conexiones.</p> <p>puerto debe ser un número de puerto válido para el sistema.</p> <p>En caso de error, arroja BindException.</p>
void escuchar(int maxColaClientes)	<p>Le indica al socket la cantidad máxima de clientes que tendrá en cola de espera para aceptar.</p> <p>maxColaClientes es un número positivo</p> <p>En caso de error, arroja ListenException</p>
void conectar(string ip, int puerto)	<p>Le indica al socket que debe conectarse al socket que se encuentra escuchando conexiones en el socket con la ip y el puerto indicado.</p> <p>ip es la dirección Ip del servidor.</p> <p>puerto es el numero de puerto donde está escuchando conexiones el servidor.</p> <p>En caso de error, arroja ConnectException.</p>
TCPSocket* aceptar()	<p>Le indica al socket que debe aceptar al próximo cliente que se intente conectar. Para que este método funcione, el socket debe haber sido bindeado con éxito anteriormente. Queda bloqueado en este método hasta que se conecta un cliente. Para desbloquear utilizar apagar().</p> <p>En caso de éxito devuelve un socket para conectarse con ese cliente.</p> <p>En caso de error, arroja AcceptException</p>
void recibir(char* data, int cantidad)	<p>Recibe la cantidad de bytes indicada y los almacena en data.</p> <p>Queda bloqueado hasta recibir los bytes indicados. Para desbloquear utilizar apagar().</p> <p>data es un buffer creado para recibir los datos que se esperan.</p> <p>cantidad es una cantidad de bytes que se están esperando.</p> <p>En caso de error, arroja SocketException.</p>
void enviar(char* data, int cantidad)	<p>Envía la cantidad de bytes indicada a partir de la posición indicada por data.</p> <p>data es la posición desde donde se leen los datos a enviar</p> <p>cantidad es la cantidad que se desea enviar</p> <p>En caso de error, arroja SocketException.</p>
void apagar()	<p>Cierra la entrada y salida de datos del socket.</p> <p>Una vez llamado a este método no se podrán enviar y recibir datos.</p>
void cerrar()	Finaliza la ejecución del socket.

Además de esta clase, también existen las excepciones que fueron mencionadas.

Diagramas

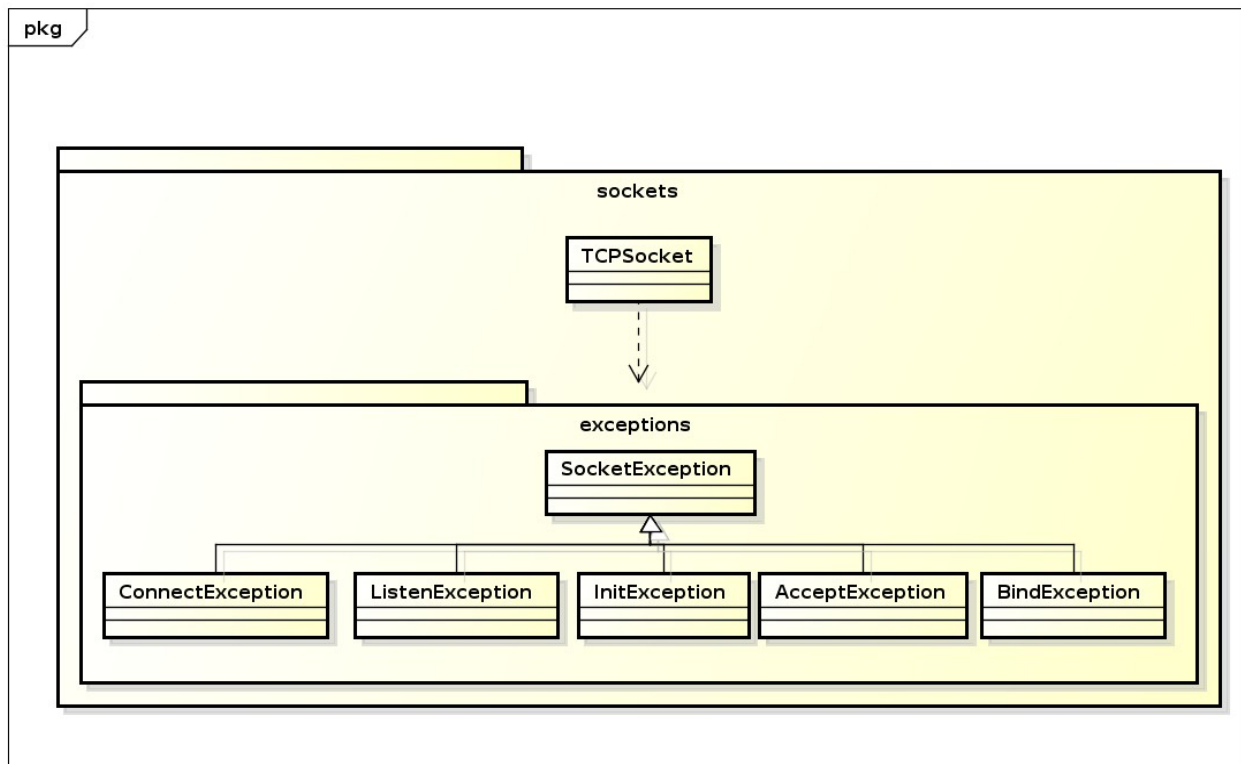


Diagrama de clases del paquete sockets

Modulo Configuración

Descripción general

Este modulo tiene como función principal abstraer totalmente al usuario del manejo de xml. Utilizando todos los métodos de estas clases es posible modificar una configuración para que tenga los valores que se deseen.

Clases

Configuración

Encargada del manejo de todas las configuraciones posibles. A partir de ella es posible realizar cualquier modificación, leer cualquier valor, guardar o recuperar desde un archivo xml.

Métodos y funciones

void setConfiguracionBasica(ConfiguracionBasica c)	Setea la configuración básica pasada por parámetro como la configuración básica del servidor.
--	---

<code>void setConfiguracionLogs(ConfiguracionLogs c)</code>	Setea la configuración de logs pasada por parámetro como la configuración de log del servidor.
<code>bool agregarTipoEstatico(TipoEstatico* t)</code>	<p>Agrega un nuevo tipo estático a la colección de tipos estáticos del servidor.</p> <p>En caso de encontrar una extensión repetida retorna falso y no agrega el elemento.</p> <p>En otro caso devuelve verdadero.</p>
<code>bool agregarTipoDinamico(TipoDinamico* t)</code>	<p>Agrega un nuevo tipo dinámico a la colección de tipos dinámicos del servidor.</p> <p>En caso de encontrar una extensión repetida retorna falso y no agrega el elemento.</p> <p>En otro caso devuelve verdadero.</p>
<code>bool agregarError(TipoError* t)</code>	Agrega un nuevo error a la colección de errores del servidor. En caso de encontrar otro elemento con el mismo código retorna falso. En otro caso verdadero.
<code>bool agregarUsuario(Usuario* u)</code>	Agrega un nuevo usuario a la colección de usuarios. Un usuario tiene un nombre único. En caso de encontrar uno con el mismo nombre devuelve falso. Verdadero en otro caso.
<code>bool removerTipoEstatico(const std::string& ext)</code>	Remueve al tipo con la extensión pasada por parámetro. En caso de no existir devuelve falso. Verdadero en otro caso.
<code>bool removerTipoDinamico(const std::string& ext)</code>	Remueve al tipo con la extensión pasada por parámetro. En caso de no existir devuelve falso. Verdadero en otro caso.
<code>bool removerError(int cod)</code>	Remueve el error con el código pasado por parámetro. Devuelve falso si no lo encuentra.
<code>bool removerUsuario(const std::string& n)</code>	Remueve al usuario con el nombre pasado. Si no lo encuentra devuelve falso. Verdadero en otro caso.
<code>std::string getTipoEstatico(const std::string& e)</code>	Devuelve el tipo del tipo estático con la extensión pasada por parámetro. Si no la encuentra devuelve un string vacío.
<code>std::string getTipoDinamico(const std::string& e)</code>	Devuelve el comando del tipo dinámico con extensión pasada por parámetro. En caso de no encontrarla devuelve un string vacío.
<code>std::string getError(int cod)</code>	Devuelve la página de error del error con el código pasado por parámetro.
<code>std::string getPassUsuario(const std::string& n)</code>	Devuelve la contraseña del usuario con nombre pasado por parámetro. En caso de no encontrar el usuario devuelve un string vacío.
<code>ConfiguracionBasica getConfiguracionBasica()</code>	Devuelve la configuración básica del servidor.

ConfiguracionLogs getConfiguracionLogs()	Devuelve la configuración de logs del servidor.
std::list<TipoEstatico> getTiposEstaticos()	Devuelve una lista de los tipos estáticos. Útil cuando se necesita recorrer todos los elementos para mostrarlos.
std::list<TipoDinamico> getTiposDinamicos()	Devuelve una lista de los tipos dinámicos. Útil cuando se necesita recorrer todos los elementos para mostrarlos.
std::list<Usuario> getUsuarios()	Devuelve la lista de los usuarios. Útil para mostrar todos los elementos.
std::list<TipoError> getTiposErrores()	Devuelve una lista con todos los errores. Útil para mostrar todos los elementos.
bool cargarDesde(const std::string& path)	Permite cargar la configuración desde un archivo xml con path pasado como argumento. Devuelve verdadero si tiene éxito y falso en otro caso.
void guardarComo(const std::string& ruta)	Permite guardar la configuración actual en un archivo xml con nombre pasado por parámetro. Si ya existía lo sobre escribe.

Para las clases ConfiguracionBasica, ConfiguracionLogs, TipoEstatico, TipoDinamica, TipoError, Usuario no se agregaron los métodos en este documento pues son sólo getters y setters de los atributos. Si han sido agregados a la documentación generada por Doxygen.

Diagramas

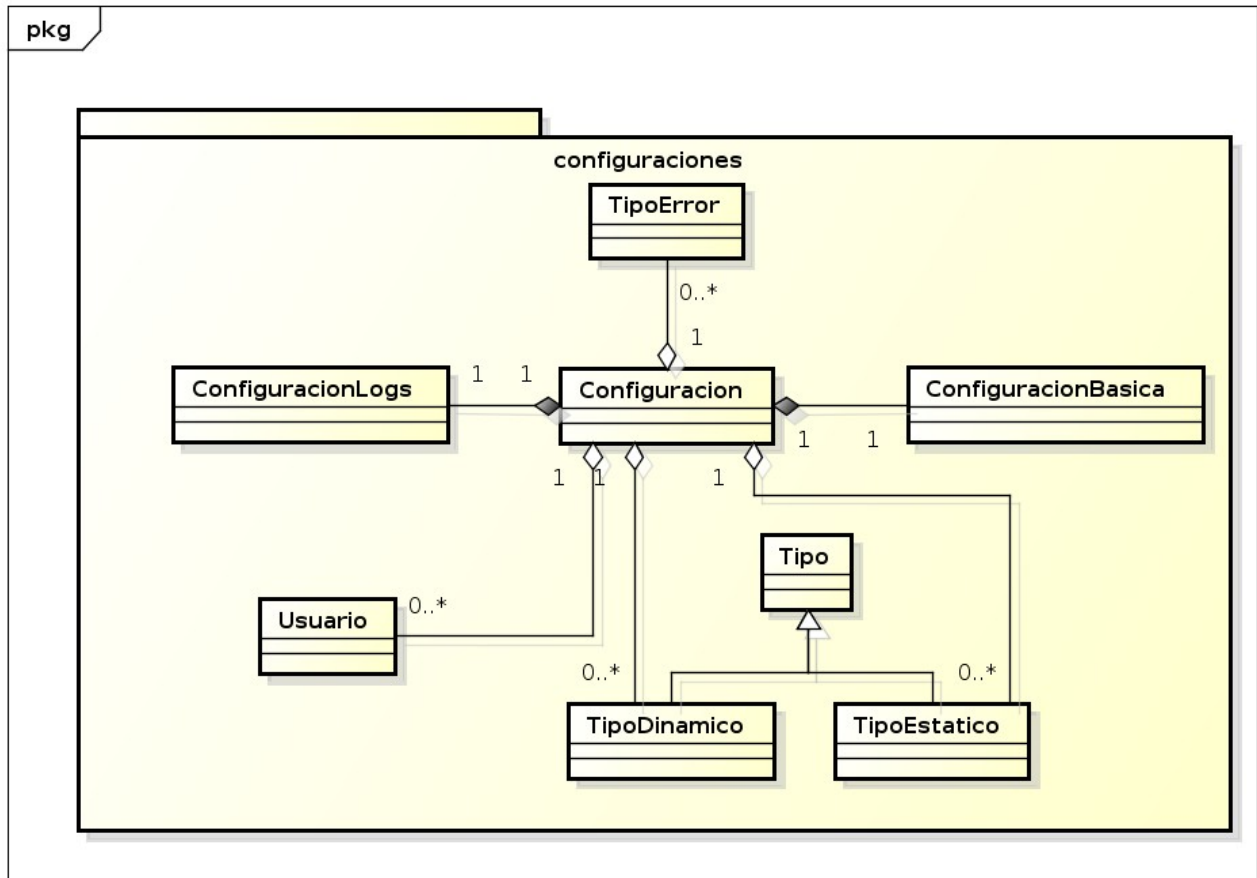


Diagrama de clases del modulo de configuración

Ejemplo de formato del xml:

```
<configuracion>
  <configuracionBasica puerto="2020" puertoControl="2021" maximoConexiones="3000"
maximoConexionesCliente="300" raiz="/home/algo/tp" protegido="no" timeOut="20"/>
  <tiposEstaticos>
    <tipoEstatico extension="html" contenido="text/html"/>
  </tiposEstaticos>
  <tiposDinamicos>
    <tipoDinamico extension="cgi" comando="cgi.exe"/>
  </tiposDinamicos>
  <errores>
    <error codigo="404" path="./errores/error404.html"/>
    <error codigo="504" path="./errores/504.html"/>
  </errores>
  <usuarios>
    <usuario nombre="root" pass="root"/>
  </usuarios>
</configuracion>
```

```
<configuracionLogs logAccesos="../Accesos.log" logErrores="../Errores.log"/>
</configuracion>
```

Módulo Protocolo de control

Descripción general

Este módulo es el encargado del manejo de los mensajes entre el configurador y un servidor.

El protocolo definido propone que solo un configurador puede enviarle mensajes al servidor al mismo tiempo.

Para el manejo de los mensajes propuestos por este protocolo se creo una clase que encapsula el protocolo.

Clases

ProtocoloControl

Clase que abstrae el uso del protocolo de control definido.

Métodos y funciones

ProtocoloControl(TCPSocket& s)	Constructor. Recibe el socket donde trabajará el protocolo.
void enviarOperacionDetenerServidor()	Envía una operación a través del socket pidiendo que se detenga el servidor.
void enviarRespuestaDetenido(bool detenido)	Envía una respuesta a través del socket según lo que se haya enviado en el parámetro. True:éxito , false: error
bool recibirRespuestaDetenido()	Recibe a traves del socket una respuesta que indica si se detuvo con éxito o no. En caso de que la respuesta sea positiva devuelve true y false en otro caso
bool recibirOperacionDetener()	Recibe a través del socket una operación. En el caso de que sea de detener devuelve true. False en otro caso.

Diagramas

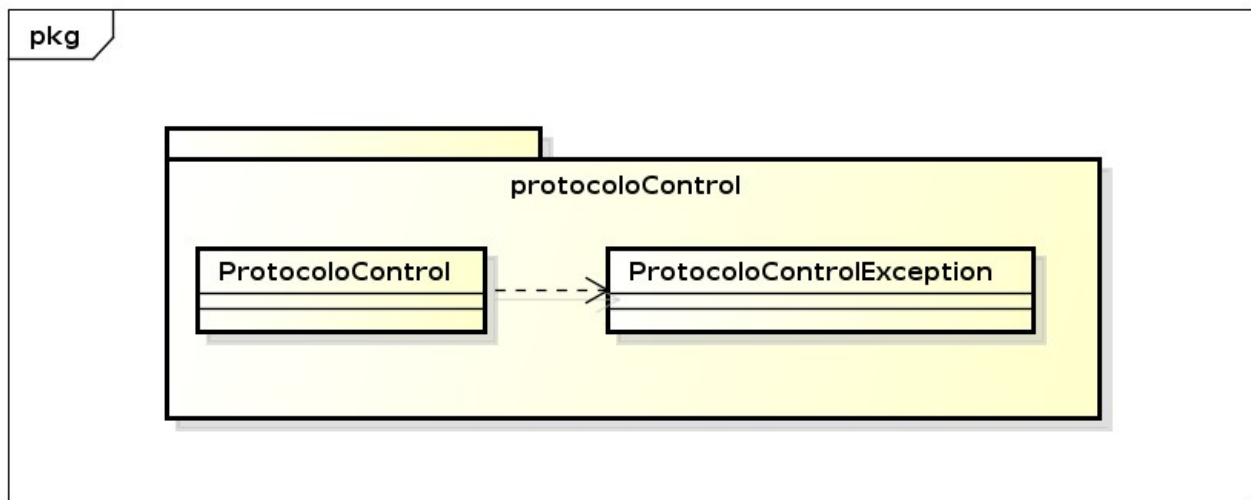


Diagrama de clases del módulo

Protocolos

Se respeta el protocolo de control explicado en el Modulo Servidor.

Modulo Servidor

Descripción general

Este modulo es el encargado de la aplicación MapacheServidor. La función principal es la del manejo del protocolo HTTP para poder procesar pedidos de múltiples clientes en simultáneo.

El funcionamiento del servidor consiste en primero crear una conexión donde estará escuchando pedidos de cierre del mismo. Una vez abierto el servidor, se crea otra conexión para el servidor que estará manejando el protocolo HTTP. Al ser necesario permitir que muchos clientes se conecten concurrentemente, surgió la necesidad de que aquel que este aceptando conexiones entrantes en el puerto del HTTP se ejecute en un hilo separado del que verifica el cierre.

Hasta aquí se pueden distinguir dos entidades importantes:

1. El Servidor que es la encargada de escuchar el pedido de cierre
2. El ServidorHTTP que es el encargado del manejo de múltiples clientes que respetan el protocolo HTTP.

El servidor HTTP, por su parte, debe cumplir alguna funcionalidad extra. Por ejemplo, según lo que pide el enunciado, debe verificar el tiempo de espera de los clientes y dar de baja a todos aquellos que superen el valor establecido en la configuración. Esta verificación debe hacerse permanentemente. Aquí surge nuevamente la necesidad de utilizar un nuevo hilo.

También es necesario restringir accesos cuando se cumplen algunas restricciones de cantidad de conexiones. Para ello se crea un administrador de clientes que tiene como funcionalidad principal tomar la decisión acerca de si un cliente está en condiciones o no de acceder en un momento dado. Además, por lo mencionado al principio de este modulo, es necesario que se procesen en simultaneo los distintos pedidos. El encargado de brindar dicha funcionalidad es ManejadorClienteHTTP, quien se comunica con el cliente conectado respetando el protocolo HTTP.

A continuación, se explica brevemente como se procesa a un cliente:

- Recepción de un pedido.
- Procesamiento según la extensión del pedido en cuestión.

- Devolución de una respuesta acorde al pedido o código de error correspondiente.

NOTA: Este modulo utiliza al modulo common.

Clases

Servidor

Esta clase hace las veces de servidor para nuestro sistema. El servidor posee un puerto de control donde escucha las peticiones de apagado del mismo. Además tiene, según el estado, un servidor HTTP corriendo.

Métodos y funciones

Servidor(const Configuracion& c, bool reiniciando)	Constructor. Crea un nuevo servidor con la configuración pasada. Escuchará el pedido de cierre del servidor HTTP en el puerto indicado en la configuración como puerto de control. Si reiniciando es true entonces el modo de apertura es de reinicio. Por lo tanto, el SO, tratará de reutilizar el puerto que se encontraba en estado de TIME_WAIT.
Int start()	Ejecuta el servidor y devuelve el código de retorno.

ServidorHTTP

Esta clase hace las veces de un servidor HTTP dentro de la aplicación. Un servidor HTTP puede manejar múltiples clientes en simultáneo respetando el protocolo HTTP. Además puede ser apagado en cualquier momento.

Esta clase implementa Thread(common/threads/Thread.h).

Métodos y funciones

ServidorHTTP(const Configuracion& c, bool reiniciando)	Constructor. Crea un nuevo servidor HTTP con la configuración pasada. Si reiniciando es true entonces el modo de apertura es de reinicio. Por lo tanto, el SO, tratará de reutilizar el puerto que se encontraba en estado de TIME_WAIT.
~ServidorHTTP()	Destructor. Libera los recursos que estaba manejando el servidor
void start()	Método de la clase base Thread. Ejecuta el método Run del servidor
void apagar()	Detiene el hilo del servidor.

AdministradorClientes

Esta clase esta encargada de administrar a los clientes dentro del servidor.

Dentro de la configuración existe un máximo de conexiones y un máximo de conexiones por cliente, los cuales deben ser respetados. Esta clase tiene como objetivo realizar los chequeos pertinentes para que sólo se atiendan peticiones que respeten estos parámetros.

En caso de errores, AdministradorClientes es el encargado de informar a los clientes (con un error de fuera de servicio).

Métodos y funciones

AdministradorClientes(const Configuracion&)	Constructor. Crea un administrador de clientes con la configuración pasada por parámetro.
~AdministradorClientes()	Destructor. Libera los recursos que estaba manejando el administrador.
void agregarCliente(ManejadorClienteHTTP*)	Agrega el manejador asociado a un cliente en el servidor. En el caso de que los parámetros de la configuración restrinjan el acceso de un nuevo cliente, entonces es el encargado de enviarle la respuesta con el error pertinente al cliente. Si pudo ser agregado entonces comienza a atenderlo.
void limpiarFinalizados()	Limpia todos los manejadores de los clientes que ya finalizaron su ejecución o que tuvieron algún error durante su ejecución.
void limpiarActivosConTimeOutMayor()	Limpia todos los clientes activos que hayan verificado la condición de timeOut indicada por la configuración.

VerificadorTimeOut

Esta clase es la encargada de verificar el tiempo de espera para todos los clientes conectados al servidor. En caso de encontrar clientes que superen el máximo especificado en la configuración del servidor, se encarga de liberar/deshabilitar a esos clientes.

El verificador corre sobre un hilo separado, pues necesita permanentemente verificar el tiempo de espera de todos los clientes.

Métodos y funciones

VerificadorTimeOut(int tiempoRefre,int tiempoConfig , AdministradorClientes* ad)	Constructor. Crea un nuevo verificador de timeout que cada tiempoRefre segundos verificará la colección de clientes conectados a ad y eliminará todos aquellos que tengan un tiempo de espera mayor a tiempoConfig
~VerificadorTimeOut()	Destructor. Libera los recursos que estaba manejando el

	verificador.
Void start();	Método de la clase base Thread. Ejecuta el método Run del verificador de timeOut.

ManejadorClienteHTTP

ManejadorClienteHTTP se encarga de realizar toda la interacción entre un cliente y el servidor. Dicha interacción se resume en:

- Recepción del pedido.
- Interpretación y procesamiento del mismo.
- Devolución de la respuesta generada al cliente.

Para permitir la atención en simultaneo de múltiples clientes, cada manejador corre sobre un hilo separado y el método run asociado al mismo es el que está implementando toda aquella interacción antes mencionada.

Métodos y funciones

ManejadorClienteHTTP(TCPSocket* s, const Configuraciopn& c)	Constructor Crea un nuevo manejador para el cliente conectado en el socket pasado por argumento y con la configuración indicada.
std::string getIpCliente()	Devuelve la ip asociada al socket del manejador
void enviarMsg(HTTP_Response* m)	Envía la respuesta indicada al cliente que está conectado al socket asociado al manejador. El Thread ya debe haber finalizado su ejecución o no debe haberla comenzado.
void apagar()	Cambia el estado del hilo a detenido.
int getOffsetSegundos() const	Devuelve el tiempo de espera asociado al manejador. Este tiempo se calcula como la diferencia en segundos entre que el cliente se conecto y el tiempo actual.

ProcesadorRequest

Esta entidad es la encargada de, según la extensión del pedido recibido y de la configuración asociada, procesar un pedido entrante al servidor.

Métodos y funciones

ProcesadorRequest(const Configuracion&)	Constructor. Crea un procesador con la configuración pasada como parámetro.
HTTP_Response* procesar(HTTP_Request*)	Procesa el pedido que recibe y devuelve la respuesta apropiada al pedido.

IProcesador

Provee una interfaz que deben implementar todos los procesadores específicos que se deseen implementar.

Métodos y funciones

HTTP_Response* procesar(HTTP_Request*)	Debe implementarse para cada procesador, con el algoritmo necesario para el proceso de un pedido.
--	---

ProcesadorTipoEstatico

Se encarga de procesar un pedido de tipo estático.

Métodos y funciones

ProcesadorTipoEstatico(const Configuracion&)	Constructor Crea un nuevo procesador para pedidos de recursos de tipo estático.
HTTP_Response* procesar(HTTP_Request*)	Procesa el pedido de tipo estático y devuelve la respuesta apropiada.

ProcesadorCGI

Esta clase es la encargada de procesar un pedido de un tipo dinámico. Para generar contenido dinámicamente se utiliza el protocolo CGI.

Métodos y funciones

ProcesadorCGI(const Configuracion& c, const std::string& ruta , const std::string rutaCGI);	Constructor Crea un nuevo procesador para pedidos que debe generar la respuesta en tiempo de ejecución utilizando el protocolo CGI.
HTTP_Response* procesar(HTTP_Request*)	Procesa el pedido CGI y devuelve la respuesta apropiada.

Algunas de las clases mencionadas anteriormente utilizan para su implementación clases utilitarias que controlan algunos aspectos como carga de archivos, manejo de fechas, protocolo HTTP. Se recomienda consultar la documentación automática de Doxygen para obtener mas información al respecto.

Diagramas

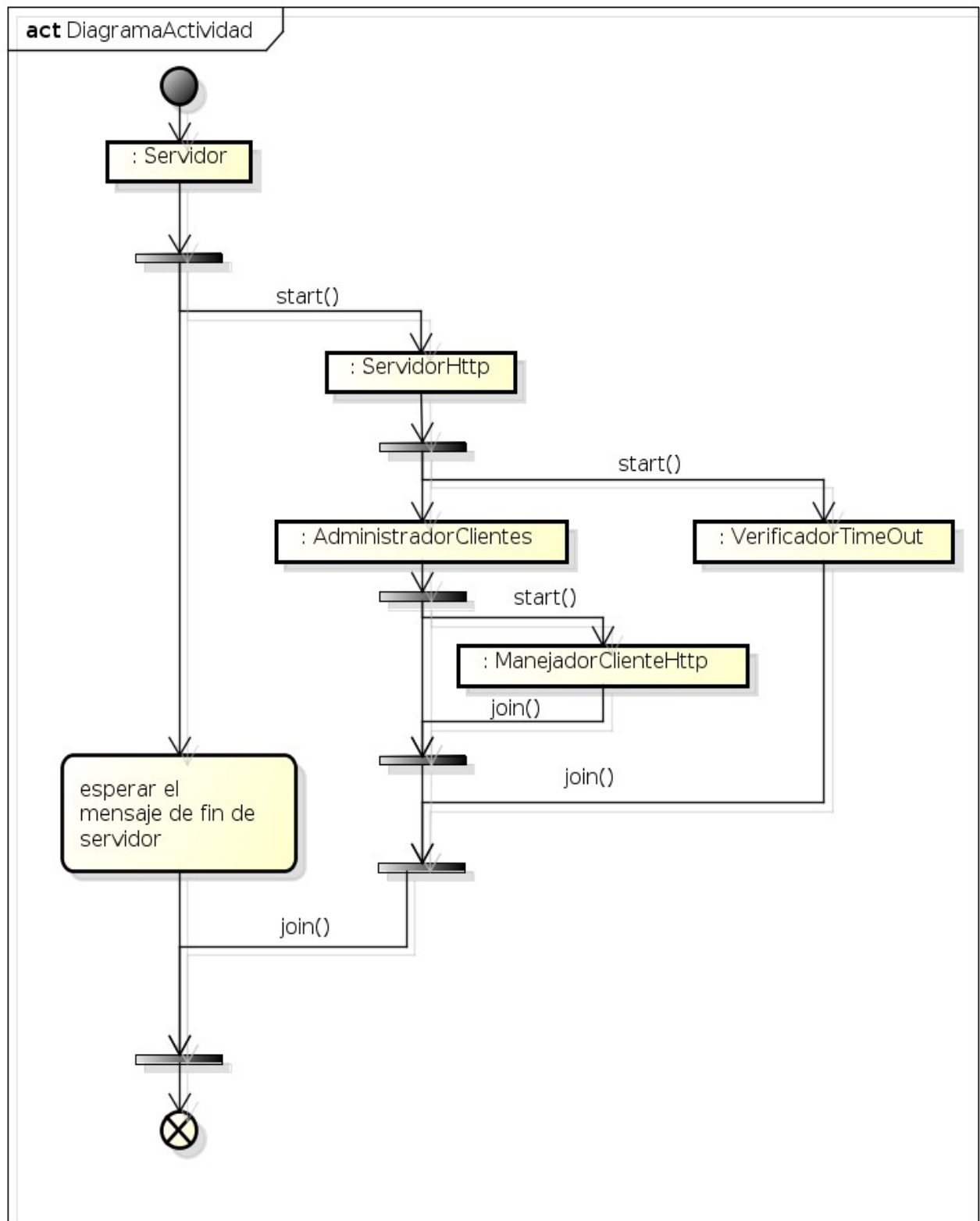


Diagrama de actividades que representa la situación del inicio de un servidor, la conexión de un cliente, el proceso del mismo (recepción del pedido, procesamiento según su tipo y devolución de una respuesta acorde). Luego el servidor recibe la operación de apagado y se cierra el mismo.

Protocolos

Protocolo de control del servidor

Para la implementación del servidor se propone un protocolo de control muy básico, pero que cumple con lo pedido. El mismo define que se puede controlar a un único cliente por vez.

El intercambio de mensajes entre el cliente y el servidor (todos los mensajes son de un byte de longitud) se da básicamente con los siguientes códigos:

CODIGO_DETENER_SERVIDOR: 'D'

CODIGO_DETENIDO_OK: 'S'

CODIGO_DETENIDO_ERROR: 'E'

Protocolo HTTP

Para implementar este servidor se utilizo el protocolo HTTP definido en <http://www.w3.org/Protocols/HTTP/1.0/spec.html>

Protocolo CGI

Para implementar este servidor se utilizó el protocolo CGI definido en <http://www.w3.org/CGI/>

Modulo Protocolo HTTP

Descripción general

Este módulo surge con el objetivo de abstraer el manejo del protocolo HTTP dentro del servidor.

Provee una serie de clases que permiten abstraer los distintos mensajes y además permite enviarlos y recibirlos.

Clases

Protocolo HTTP

Permite enviar y recibir mensajes del protocolo HTTP

Métodos y funciones

ProtocoloHTTP(TCPSocket* s)	Crea un nuevo objeto que abstraerá el uso del protocolo HTTP.
HTTP_Request* recibirOperacion()	Recibe una operación a través del socket asociado. En caso de error de conexión, arroja socket exception
Void enviarRespuesta(HTTP_Response*)	Envía la respuesta a través del socket asociado. En caso de error de conexión, arroja socketException.

HTTP_Response

Abstrae una respuesta generada por el servidor

Métodos y funciones

Aquellos necesarios para poder setear cualquier campo de un response.

Para mas información acerca de esta clase ver la documentación generada por Doxygen.

HTTP_Request

Abstrae un pedido recibido por el servidor.

Métodos y funciones

Posee métodos para el proceso un pedido.

Para más información ver la documentación generada por Doxygen.

Diagramas

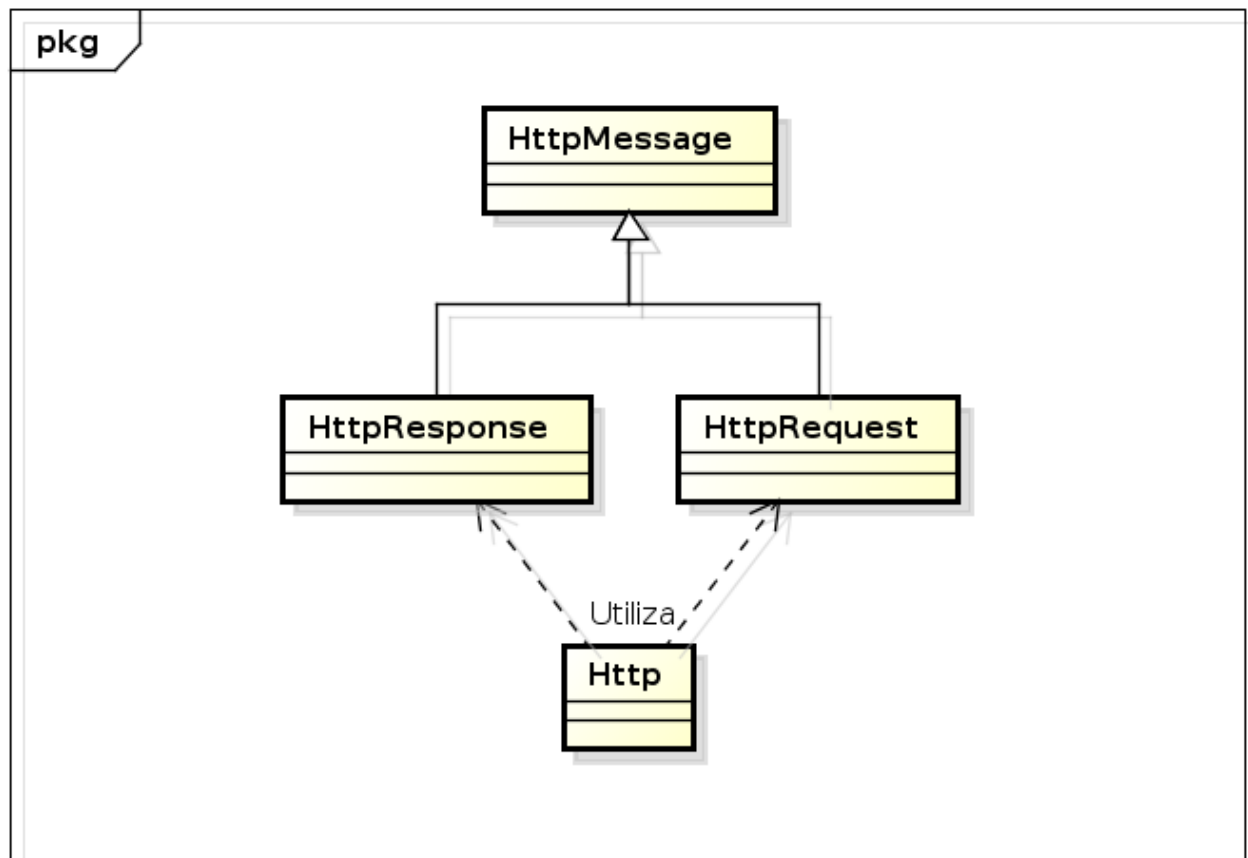


Diagrama de clases del protocolo HTTP

Protocolos

Estas clases respetan el código HTTP antes mencionado.

Modulo Configurator

Descripción general

Este modulo se encarga de establecer la interacción de la aplicación con el usuario, permitiendo al usuario configurar los parámetros de la aplicación de manera gráfica.

Para la interfaz gráfica se utiliza Gtkmm-2.4

Submodulos

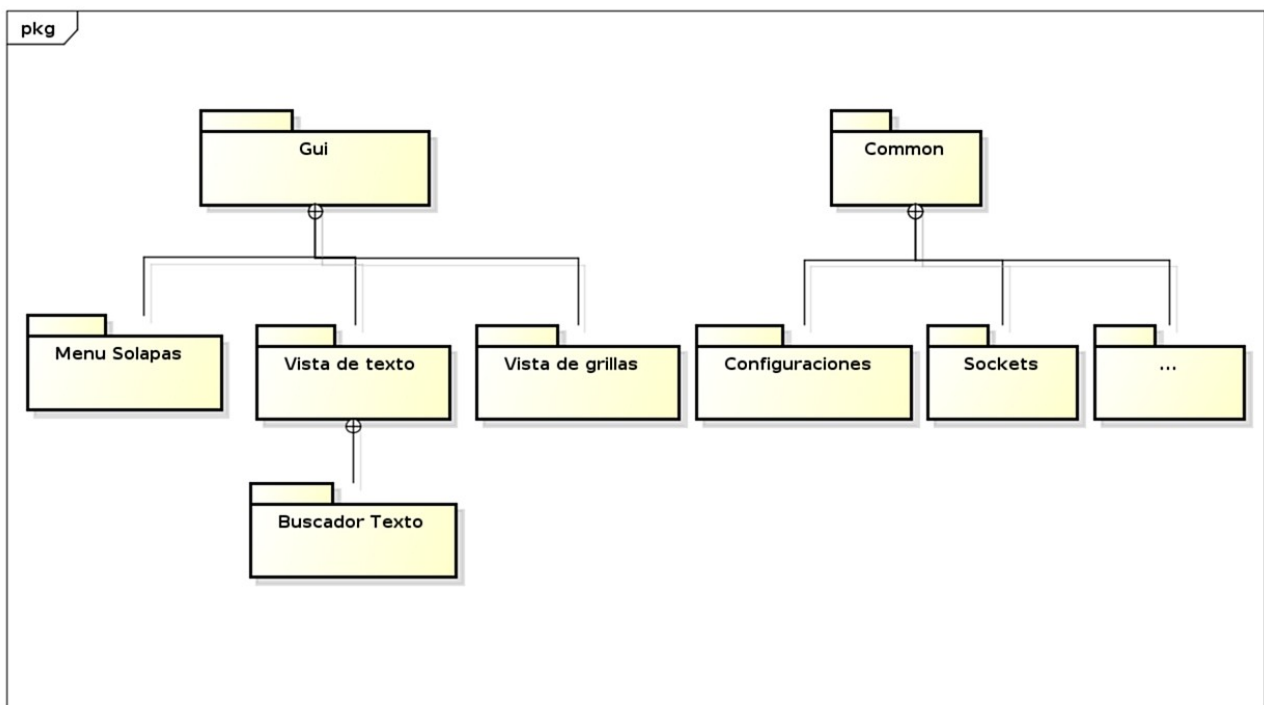
Menú Solapas: Menú en el cual se accede a las opciones mediante solapas

Vista Texto: visualizador de reportes que incluye buscador de texto.

Vista Grilla: visualizador de grillas las cuales permiten alta, baja y modificación de datos

Common: se utiliza para persistir configuraciones (en xml) y realizar conexiones a la aplicación mediante sockets, este modulo se detalla en el archivo “Modulo_Common”

Diagramas



powered by astah

Descripción modulos-submodulos del configurador

Modulo Gui

Diagrama

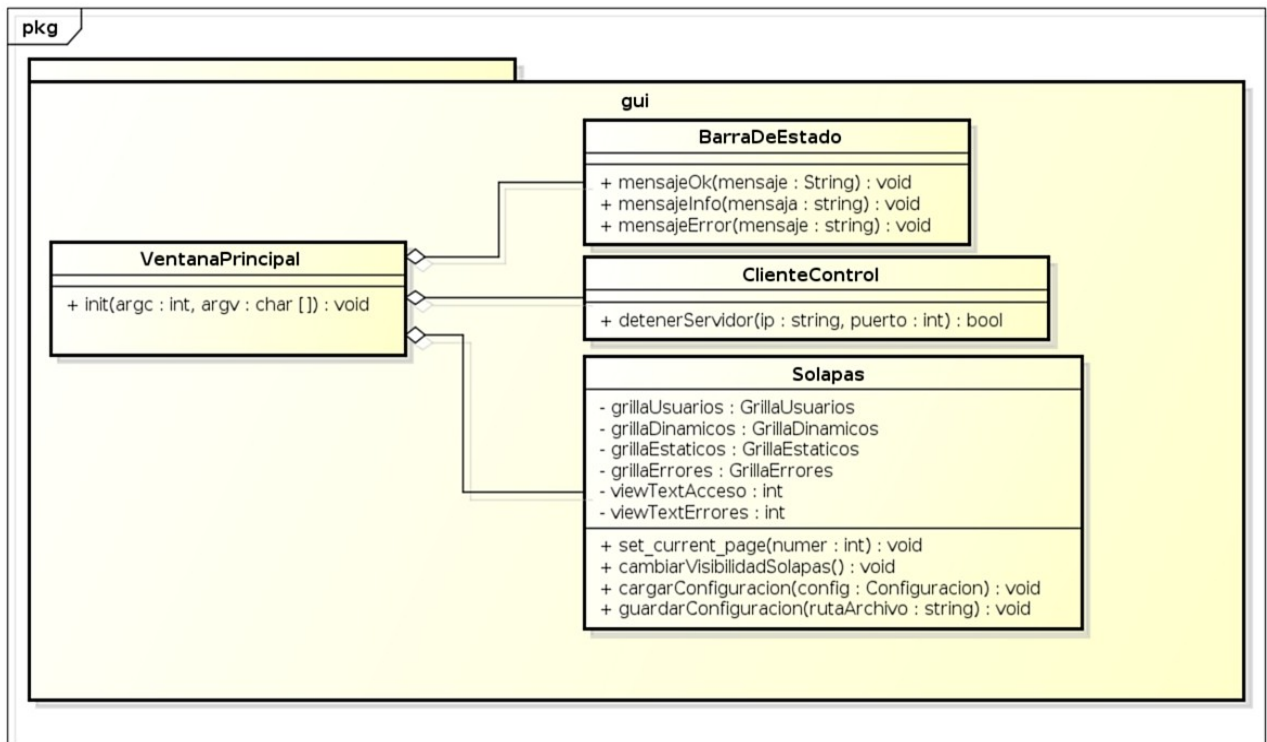


Diagrama de clases, que define las relaciones entre los componentes principales de la interfaz gráfica

Descripción general:

Este módulo es el utilizado para la creación de ventanas gráficas, las cuales interactúan con el usuario para poder configurar la aplicación.

Para la creación de ventanas gráficas, se utiliza Gtkmm 2.4.

Clases

VentanaPrincipal

Clase que se encarga de crear la ventana principal de la aplicación.

Métodos y funciones

void init (int argc, char *argv[])	Comienza la aplicación gráfica.
------------------------------------	---------------------------------

BarraDeEstado

Clase utilizada para la creación y administración de mensajes en la barra de estado de la aplicación

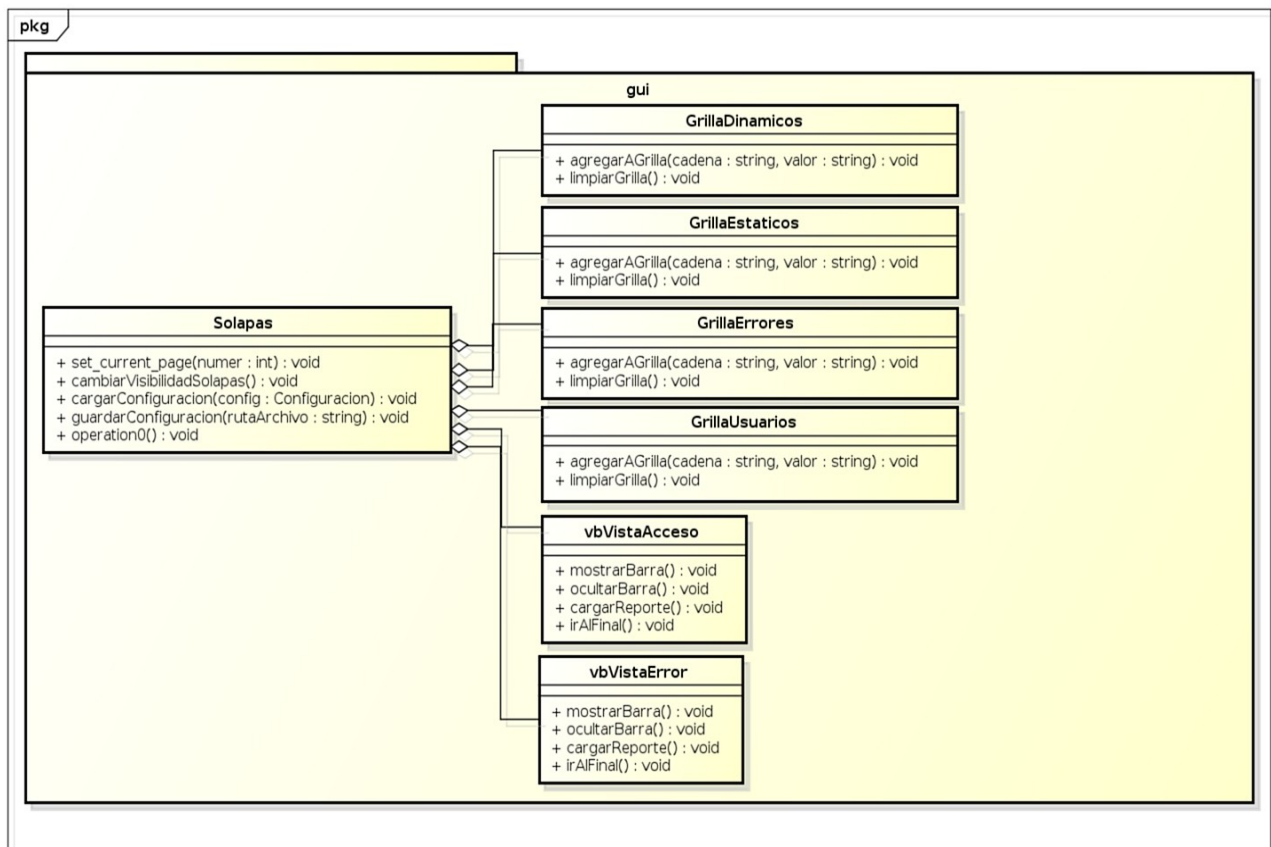
Métodos y funciones

void mensajeOk(const std::string& mensaje)	Muestra el mensaje ingresado como parámetro, incluye icono que define como estado ok
void mensajeError(const std::string& mensaje);	Muestra el mensaje ingresado como parámetro,

	con un icono que define como estado error
void mensajeInfo(const std::string& mensaje)	Muestra el mensaje ingresado como parámetro, con un icono que define como estado información

Solapas

Diagrama de clases



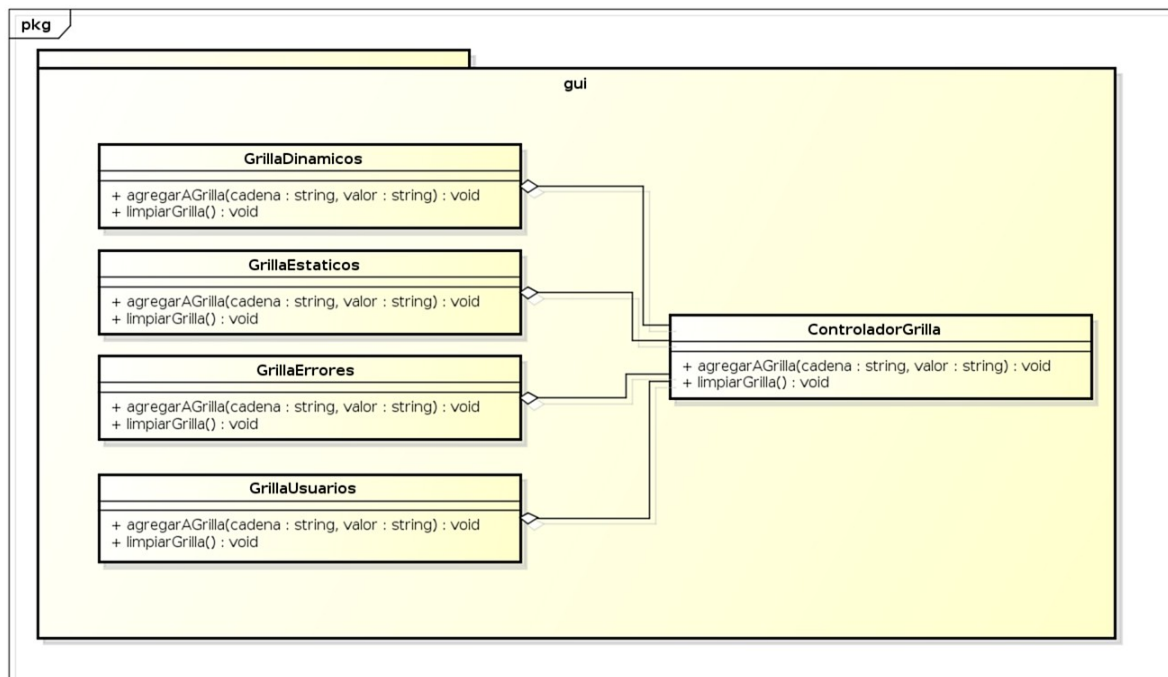
Esta clase ofrece un menú para la aplicación en el cual se tienen opciones mediante solapas.

Métodos y funciones

void set_current_page(int numero);	Muestra la solapa que esta en la posición ingresada por el parámetro
void cambiarVisibilidadSolapas();	Muestra u oculta las solapas, permitiendo ocultar las solapas para tener mas espacio en la aplicación
void cargarConfiguracion(Configuracion& config);	Carga la configuracion a los distintos controles de la aplicación.
void guardarConfiguracion(const std::string &rutaArchivo);	Guarda la configuracion de la aplicación en formato xml.

GrillaDinamicos, GrillaEstaticos, GrillaErrores, GrillaUsuarios

Diagrama de clases



Estas clases son controles de usuario personalizados.

Su funcionalidad es tener una grilla con opciones para realizar operaciones de alta, baja y modificación en dicha grilla, para eso utilizan un controlador de grilla.

El mismo control es utilizado en las 4 clases, la diferencia que hay entre estas clases es los controles gráficos que utiliza cada una, tomando información desde esos controles.

Métodos y funciones

void agregarAGrilla(const std::string &cadena, const std::string &valor);	Agrega a la grilla los datos ingresados como parametros.
void limpiarGrilla();	Borra todos los datos que tiene cargados la grilla

ControladorGrilla

Se encarga de controlar los cambios en la grilla, administra los eventos que se producen en la grilla, Alta baja y modificación, además de borrar todos los datos de la grilla.

Métodos y funciones

void agregarAGrilla(const std::string &cadena, const std::string &valor);	Agrega a la grilla los datos ingresados como parametros.
void limpiarGrilla();	Borra todos los datos que tiene cargados la grilla

VbVistaAcceso, VbVistaError

Estas clases son controles de usuario personalizados.

Su funcionalidad mostrar por pantalla un reporte, permitiendo al usuario realizar búsquedas personalizadas sobre dicho reporte.

El mismo control es utilizado en las 2 clases, la diferencia que hay entre estas clases es los controles gráficos que utiliza cada una, tomando información desde esos controles.

Métodos y funciones

void mostrarBarra()	Muestra la barra de búsqueda, dando la posibilidad de realizar búsquedas personalizadas sobre el reporte que se visualiza
void ocultarBarra()	Oculto la barra de búsqueda
void cargarReporte()	Carga el reporte
void irAlfinal()	Posiciona el cursor en el final del reporte

ClienteControl

Esta clase se encarga de comunicarse mediante sockets con la aplicación (ServidorMapache) para detenerla.

Métodos y funciones

bool detenerServidor(const std::string& ip, unsigned int puerto);	Detiene el servidor, se conecta mediante socket y le envía una señal definida en el protocolo para que el servidor se detenga.
---	--

Modulo Precompilador

Descripción general:

Este módulo se encarga íntegramente de precompilar (según los lineamientos dados en el TP 1) uno o varios archivos fuente de c/c++. El resultado de la precompilación es mostrado por STDOUT.

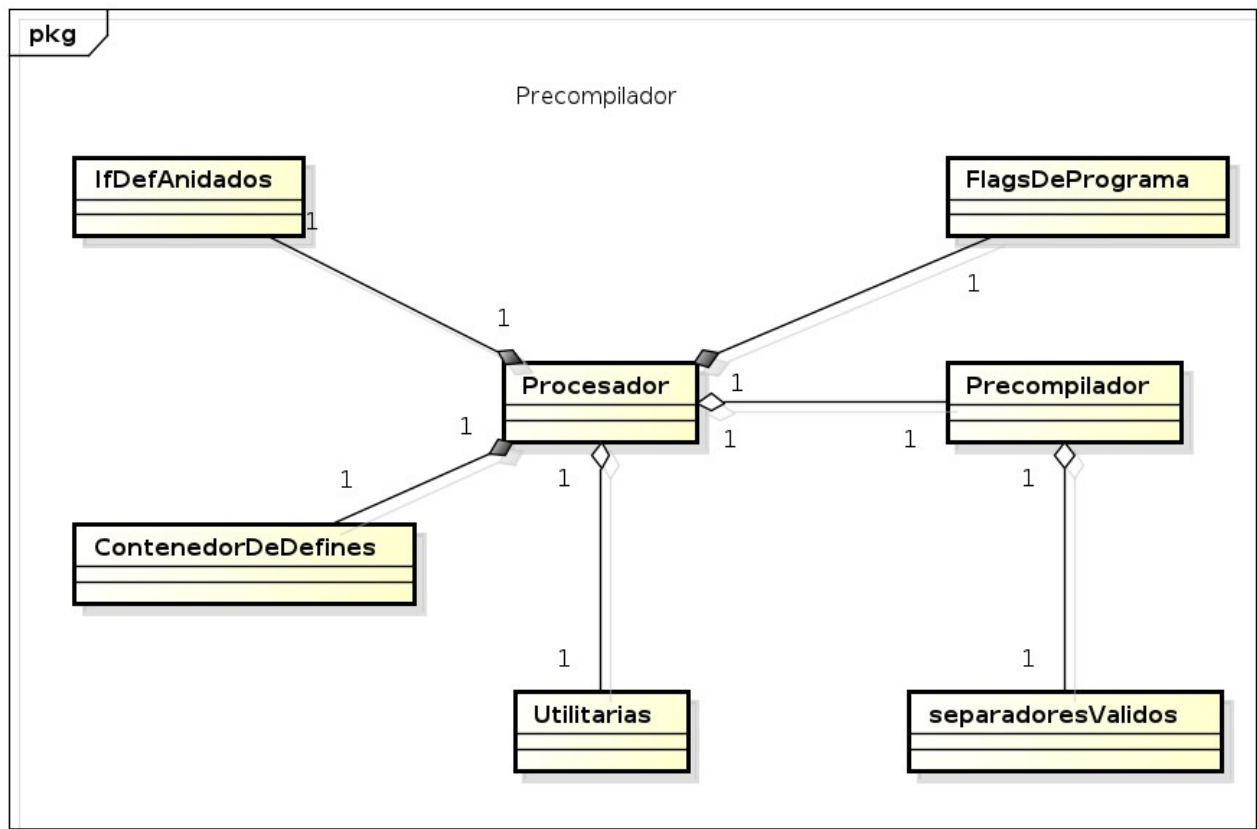


Diagrama de Clases

Clases

ContenedorDeDefines

Entidad que almacena, establece, y devuelve define`s. Es manejada por Procesador y usada por Procesador y Precompilador.

Métodos y funciones

ContenedorDeDefines()	Constructor. Inicializa la estructura con valores por defecto.
bool existeEsteDefine(char* buscado, int size)	Busca en toda la estructura la cadena apuntada por buscado. True si existe, false en caso contrario.
void incrementarIndex()	Incrementa en uno el índice que lleva la cuenta de la cantidad de defines que tiene almacenada la estructura.

Getters y setters fueron omitidos. Consultar la documentación automática de Doxygen para mas detalles.

FlagsDePrograma

Clase que representa el estado de las variadas banderas que tiene el programa.

Métodos y funciones

FlagsDePrograma()	Constructor. Establece un valor inicial valido para todos los flags de la estructura.
bool lineaNoEstaTotalmenteComentada()	Determina si la linea actualmente en proceso esta totalmente o no comentada. Quien procesa dicha linea es quien establece el valor de los flags que intervienen en este método.

IfDefAnidados

Encargada de almacenar los estados de compilación o no compilación. Cambien brinda servicios de consulta de los mismos. Es manejada por Procesador.

Métodos y funciones

void chequearProfundidadDeAnidamiento(FlagsDePrograma& flags)	Verifica que la profundidad de anidamiento no supere el máximo permitido. Si lo excede, el flag correspondiente es seteado en flags.
bool compilar()	Determina si el ultimo valor anidado corresponde a “compilar”.
void anidar(char codigo_A_Anidar, FlagsDePrograma& flags)	Anida en la estructura el codigo dado en codigo_A_Anidar.
void desanidar()	Ignora el valor de la ultima anidación y disminuye en uno el valor de indiceIfdefActual, el cual lleva la cuenta de la cantidad de anidaciones.
void invertirUltimoValorAnidado()	Invierte el valor de la ultima anidación.

Getters y setters fueron omitidos. Consultar la documentación automática de Doxygen para mas detalles.

Precompilador

Descripción general:

Se encarga de:

- Buscar y realizar reemplazos de defines por sus valores dentro del código fuente.
- Buscar y eliminar comentarios.

Métodos y funciones

char* buscarReemplazos(char* linea, ContenedorDeDefines& contenedor)	Busca define`s en linea, y efectúa reemplazos si halla coincidencias.
char* quitarComentarios(char* linea, FlagsDePrograma& flags)	Remueve los comentarios que pueda tener linea.
Void	Busca apertura de comentarios en linea y

buscarAperturaDeComentarioOBarraDoble(char* lineaAux, char* linea, FlagsDePrograma& flags)	establece los flags pertinentes en flags.
void buscarCierreDeComentario(char* lineaAux, char* linea, FlagsDePrograma& flags)	Busca cierre de comentarios en linea y establece los flags pertinentes en flags.

Procesador

Se encarga del parseo del o los archivos que el programa necesite leer.

Extrae nombres y valores de los defines.

Parsea las instrucciones y valida su sintaxis.

Métodos y funciones

void procesoArchivoEntrada(const char* nombreArchivo)	Lee linea a linea el archivo nombreArchivo, y las procesa.
bool esDirectivaDeCompilador(char* linea)	Decide si el contenido de linea es una directiva de compilador.
void procesarDirectivaOBuscarReemplazos(char* linea)	Decide si hay que procesar una directiva de compilador o buscar reemplazos.
void procesoLinea(char* linea)	Quita comentarios, reemplaza define's, o procesa directivas de compilador que se encuentren en la linea.
char* procesarDirectivaCompilador(char* linea)	Sabiendo que se trata de una directiva de compilador, se intenta saber cual de ellas es.
void procesarXX(char* linea, ...)	Serie de métodos para procesamiento de instrucciones. XX puede ser include, else, define, ifdef, ifndef, endif.

SeparadoresValidos

Contiene todos los caracteres separadores para un define que el programa considera validos.

Métodos y funciones

esSeparadorValido(char car)	Define si car es un separador valido
-----------------------------	--------------------------------------

Utilitarias

Posee un conjunto de funciones auxiliares para el manejo de cadenas y caracteres.

Métodos y funciones

void copiarCadenaYAgregarNullAlFinal(char* char1, char* char2, int sizeCh2)	Copia el contenido de la segunda cadena en la
---	---

	primera y agrega un carácter nulo al final.
getLargoCadena(char* cadena, char* vectorDeTopes, int sizeVec)	Esta función se diferencia de strlen en que determina el fin de la cadena cuando encuentra alguno de los caracteres dentro de vectorDeTopes.
esNumero(char* c)	Determina si la cadena de caracteres puede ser interpretada como un número.

Modulo CGI_Precompilador

Descripción general

Este módulo se encarga de parsear STDIN (por el cual recibe un post) con el fin de generar temporalmente aquellos archivos que allí se encuentren.

Posteriormente, el modulo precompilador se hará cargo de precompilar todos los archivos fuentes que hayan sido generados.

Al finalizar, dichos archivos temporales son eliminados y envía a STDOUT el resultado de la recompilación.

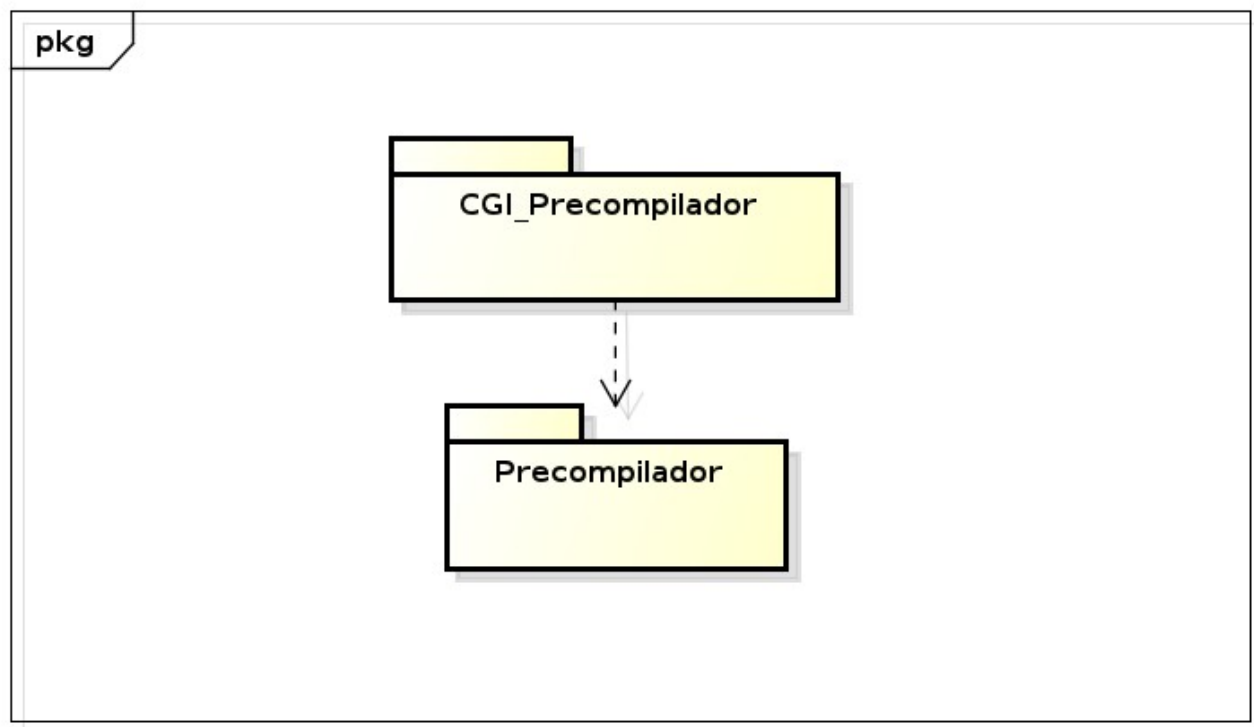


Diagrama de Paquetes

Clases

ParserEntradaEstandar

Parsea y crea los archivos temporales que encuentre en el post recibido por STDIN.

Métodos y funciones

ParserEntradaEstandar(int content_length, char* ruta)	Constructor. content_length determina la cantidad máxima de bytes a ser leídos de la STDIN. ruta especifica el path en donde generar los archivos temporales.
int crearArchivosAPartirDeEntradaEstandar()	Parsea el post, que viene por STDIN, y extrae la información de los archivos que contiene, colocando dicha info en archivos temporales.
void levantarDatosDeCabecera()	Se parsea la marca de fin de archivo actual y la marca de fin de post.
void analizarLinea(char* cadena)	Chequea si en cadena se encuentra la marca de fin de archivo actual, de fin de post o si se trata de datos del archivo actual, y realiza las acciones pertinentes.
void leerLinea(char* linea)	Lee una linea de STDIN.
void escribirLineaAArchivo(string* lineaStr)	Escribe lineaStr en el ultimo archivo creado.
bool marcaDeFinDeArchivo_Post(string* linea)	Determina si linea contiene la marca de fin de post.
bool quedanBytesPorLeer()	Verifica que no se haya superado la cant. máxima de bytes leídos, que viene dada por content_length.

Observaciones:

- Getters y setters fueron omitidos. Consultar la documentación automática de Doxygen para mas detalles.
- Este modulo hace uso del modulo precompilador.

Modulo CGI_Pr-Zip

Descripción general

Este módulo se encarga de parsear STDIN (por el cual recibe un post, por el cual se ha enviado un archivo zip) con el fin de generar temporalmente un único archivo zip, el cual sera descomprimido. Los archivos resultantes de la descompresión serán puestos a disposición del modulo precompilador.

Al finalizar, tras eliminar al archivo zip y sus archivos descomprimidos, se envía a STDOUT el resultado de la recompilación.

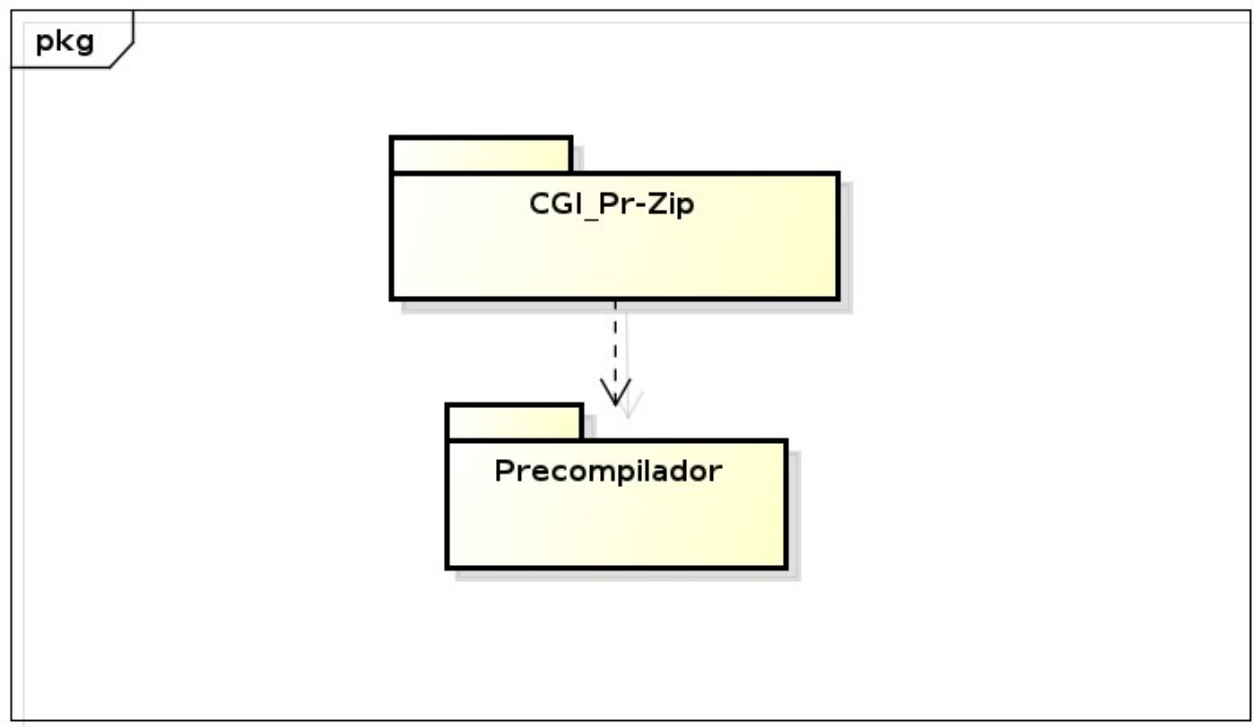


Diagrama de Paquetes

Clases

ParserEntradaEstandar

Parsea el archivo zip encuentre en el post recibido por STDIN y lo descomprime.

Observaciones:

- Getters y setters fueron omitidos. Consultar la documentación automática de Doxygen para mas detalles.
- Este modulo hace uso del modulo precompilador.

Modulo CGI_ParsePHP

Descripción general

Este módulo se encarga de invocar a la linea de comandos de php, no sin antes generar/armar el comando adecuado según los parametros recibidos. La salida se envía a STDOUT.

Clases

ProcesadorPHP

Arma el comando adecuado para ser ingresado por linea de comandos.

Métodos y funciones

bool extraerParametrosParaElPHP(string& rutaPHP , const string parametros , list<string>&	Parsea la cadena parametros y agrega cada ocurrencia a listaParams. Cambien parsea
---	--

listaParams)	rutaPHP.
string armarComandoPHP(const string rutaPHP , std::list<string> listaParams)	Genera el comando para que sea ejecutado en una terminal.

Programas intermedios y programas de prueba

Mozilla Firebug

WireShark

Apache Web Server

Netcat

Manual de usuario

Instalación

Requerimientos de software

Se requiere tener instalado:

- libtinyxml 2.5.3
- php5
- cmake 2.8.3
- gtkmm-2.4
- log4cpp

Opcional para generar documentación automática

- doxygen 1.7.3
- graphviz
- dot2tex

S.O.

Probado en Ubuntu 10.10, 11.04, 11.10.

Requerimientos de hardware

Memoria Ram: 256 MB (Minimo) / 512 MB (Recomendado)

Espacio en disco rígido: 50Mb

Procesador: Intel Pentium 4 o superior.

Proceso de instalación

El proceso de instalación es muy simple. Una vez instaladas las librerías indicadas en requerimientos de software solo se debe ejecutar el comando `./install.sh` que se encuentra dentro de la carpeta base del proyecto.

Para más información: Ver [Mapache/Ayuda/instalacion.html](#)

Configuración

Para poder hostear su propio sitio web deberá crear una configuración válida. Para ello se ha creado una interfaz de usuario que le permite realizarlo de una manera fácil, rápida e intuitiva.

El proceso de configuración consta de una serie de etapas. A continuación se explica cada una de ellas:

1. **Crear una configuración básica:**

La configuración básica será la que configurará los parámetros más indispensables de un servidor.

- Puerto: este campo sirve para indicar a que puerto responderá nuestro servidor HTTP una vez iniciado.
- Puerto Control: este campo es utilizado para indicarle al servidor en que puerto estará escuchando la petición de fin de ejecución.
- Máximo conexiones: este campo indica la cantidad máxima de conexiones que podrá manejar nuestro servidor en simultaneo. Se debe tener en cuenta que este número no debe ser demasiado alto ya que no alcanzarían los recursos de la máquina.
- Máximo conexiones por cliente: este campo indica la cantidad máxima de conexiones que podrá tener un mismo cliente simultáneamente. No tendría sentido que este número sea mayor que el máximo de conexiones pues nunca se podrían aprovechar.
- Raíz: este campo le indica al servidor donde se encuentra la raíz del sitio. Se recomienda que todos los sitios hospedados en el servidor tengan dentro de la carpeta raíz un archivo llamado "index.html". Ese será el que el servidor tome por defecto. En caso de no encontrarlo devuelve error 404.
Para seleccionar la raíz del sitio se debe utilizar el buscador de carpetas que se encuentra a la derecha del campo de texto.
- Protegido: este campo le indica al servidor si estará protegido por autenticación BASIC o no. Para un servidor autenticado marque la casilla.
- Time Out : este campo le indica al servidor cuanto deberá esperar para determinar que una conexión esta caída. Una vez pasado ese tiempo indicado en segundos, el servidor detendrá esas conexiones e indicará esa acción dentro del LOG de errores del mismo.

2. **Agregando tipos estáticos:**

Para la utilización de nuestro servidor se deberán agregar los tipos de datos que manejará.

Para realizarlo, primero debemos movernos a la vista de tipos estáticos. Una vez allí, utilizaremos los botones de la parte inferior para agregar, modificar y eliminar tipos de datos.

3. **Agregando tipos dinámicos:**

Al igual que con los estáticos, se debe llegar a la vista de dinámicos y agregar, modificar o eliminar los datos que se deseen.

4. **Agregando páginas de error:**

La idea es similar al agregado de tipos. Primero se debe llegar a la vista de errores y luego se agregan las páginas de error utilizando los botones inferiores.

5. **Agregar usuarios permitidos:**

Cuando nuestro servidor este protegido es necesario agregar usuarios permitidos que puedan navegarlo. Para ello se accede a la vista de usuarios y con los botones ubicados en la parte inferior de la ventana se podrán realizar las modificaciones pertinentes.

6. **Configuración de LOGs:**

Por ultimo debemos configurar la locación de los archivos de log del servidor

Para más información y capturas de pantalla: Ver [Mapache/Ayuda/configuracion.html](#)

Forma de uso

Una vez que tenemos configurado nuestro servidor es hora de iniciarlo. Para ello, tenemos un botón en la parte superior de la ventana que permite hacerlo.

Es importante que a la hora de ejecutar el servidor tengamos una configuración consistente. En caso de no hacerlo es posible que muchos errores se presenten en el LOG de errores.

Además de poder iniciarlo, se ofrece la posibilidad de reiniciarlo y detenerlo.(También utilizando los botones en la parte superior, o utilizando el menú Control).

Además como administrador de un servidor le interesará ver que es lo que estuvo pasando. Para ello tenemos la posibilidad de revisar los LOGs desde la misma aplicación. Además se provee de métodos para la búsqueda en los LOGs.

Para más información: Ver [Mapache/Ayuda/control.html](#) y [Mapache/Ayuda/administracion.html](#)

Apéndice I

Pruebas Servidor

Ejemplos

Se provee de una carpeta con distintos sitios de ejemplo. Además se proponen una serie de pruebas, que utilizando estos sitios, permiten verificar la funcionalidad del servidor.

Prueba 1

La siguiente prueba consiste en comprobar que el servidor funciona para una página HTML básica. Cuando decimos que es básica nos referimos a que solo tendrá texto.

Configuración necesaria para esta prueba:

Configuración básica:

Puerto: 2025

Puerto Control: 10000

Máximo conexiones: 100

Máximo conexiones cliente: 10

Raíz: Seleccionar “ejemplos/0/html” utilizando el buscador.

Protegido: Sin marcar

Time out: 20 segundos

Tipos estáticos:

Extensión	Tipo
html	text/html

Tipos dinámicos:

No son necesarios

Usuarios:

No son necesarios

Errores:

Agregar los errores que se encuentran en “ejemplos/0/errores/”

Código	Error
404	404.html
503	503.html

Configuración de logs:

Se seleccionan nombres para los logs de errores y de accesos, Prueba1Errores.log y

Prueba1Accesos.log respectivamente.

Una vez finalizada la configuración, iniciamos el servidor y veremos un mensaje en la parte inferior del configurador que nos indicará si tuvimos éxito o no en el inicio del mismo.

Ahora verificamos que el mismo se haya iniciado correctamente. Para ello abrimos un navegador y escribimos en la barra de direcciones localhost:2025 y vemos que se abre una página indicando Ejemplo 0.

Luego de hacer la prueba se debe detener el servidor.

Prueba 2

La siguiente prueba consiste en probar una página básica pero, a diferencia de la prueba anterior, este servidor estará protegido con autenticación BASIC.

Se debe utilizar una configuración semejante a la anterior pero con las siguientes modificaciones.

Primero se debe marcar a la casilla Protegido de la configuración básica, luego se deben agregar usuarios que tengan permitidos el accesos. Además cambiamos los puertos por los siguientes:

Puerto: 2026

Puerto Control: 10001

Para este ejemplo agregamos un único usuario permitido:

Nombre	Contraseña
admin	admin

Se debe considerar que estos campos son case sensitive a la hora de ingresar al sitio.

Una vez finalizada la configuración se inicia el servidor.

Luego, utilizando un navegador, ingresamos a localhost:2026 y veremos que el navegador nos exigirá un nombre de usuario y una contraseña.

Distintos casos:

1. Campos vacíos:

Nombre	Contraseña

Veremos a continuación que nuevamente nos exige que nos autentiquemos. Esto se debe a que no es valida esa acción.

2. Usuario inválido:

Nombre	Contraseña
Usuario	Pass

Veremos nuevamente que nos exige autenticarnos nuevamente ya que el usuario no existe.

3. Usuario válido, contraseña invalida:

Nombre	Contraseña
admin	invalido

Nuevamente nos exige autenticarnos.

4. Usuario válido y contraseña válida:

Nombre	Contraseña
admin	admin

Luego de ingresar estos datos, los válidos, se nos mostrará nuevamente una página indicando que es el ejemplo 0.

Se debe detener el servidor.

Consideración:

Una vez que el usuario se autenticó con éxito, el navegador no volverá a pedir que se ingresen estos datos.

Prueba 3

El objetivo de esta prueba es mostrar que el servidor deja de recibir clientes cuando supera el máximo de conexiones. Para simular esa situación utilizamos la configuración de la Prueba 1 pero modificando los máximos de conexiones totales y por cliente a cero.

Si un cliente quiere entrar entonces se supera el máximo de conexiones y por ende se superan esos límites.

Máximo conexiones: 0

Máximo conexiones cliente: 0

Además cambiamos los puertos por los siguientes:

Puerto: 2027

Puerto Control: 10002

Iniciamos el servidor con esa configuración y utilizando el navegador accedemos a localhost:2027 y observamos que se informará que el servidor se encuentra fuera de servicio por el momento.

Por último, se debe detener el servidor.

Prueba 4

El objetivo de la siguiente prueba es mostrar que es posible el manejo de dos sitios distintos al mismo tiempo. Se debe tener en cuenta que los puertos de ambos sitios deben ser diferentes.

Para el primer sitio utilizamos la configuración de Prueba 1 cambiando los puertos por:

Puerto: 2028

Puerto Control: 10003

Se debe iniciar este servidor.

El segundo sitio debe tener la configuración que mostramos a continuación:

Configuración básica:

Puerto: 2029

Puerto Control: 10004

Máximo conexiones: 100

Máximo conexiones cliente: 10

Raíz: Seleccionar “ejemplos/1/html” utilizando el buscador.

Protegido: Sin marcar

Time out: 20 segundos

Tipos estáticos:

Extensión	Tipo
html	text/html
png	image/png

Tipos dinámicos:

No son necesarios

Usuarios:

No son necesarios

Errores:

Agregar los errores que se encuentran en “ejemplos/1/errores/”

Código	Error
404	404.html
503	503.html

Se debe utilizar el buscador de la interfaz.

Una vez realizada esta configuración debemos iniciar este servidor.

Ingresando con el un navegador a localhost:2028 veremos que se devuelve la misma página que en la prueba 1. Ingresando a localhost:2029 vemos que se abre una página que contiene una imagen.

Luego se deben cerrar ambos servidores. El que se encuentra activo con puerto de control 10003 y el otro con puerto 10004.

Prueba 5

El objetivo de esta prueba es probar la integración de PHP dentro del servidor. Para lograr esto se debe agregar un comando que se encarga de resolver las peticiones de este tipo.

Usamos la siguiente configuración:

Configuración básica:

Puerto: 2030

Puerto Control: 10005

Máximo conexiones: 100

Máximo conexiones cliente: 10

Raíz: Seleccionar “ejemplos/2/html” utilizando el buscador.

Protegido: Sin marcar

Time out: 20 segundos

Tipos estáticos:

Extensión	Tipo
html	text/html
png	image/png

Tipos dinámicos:

Extensión	Comando
php	parserPHP.cgi

Usuarios:

No son necesarios

Errores:

Agregar los errores que se encuentran en “ejemplos/2/errores/”

Código	Error
404	404.html
503	503.html

Se debe utilizar el buscador de la interfaz.

Una vez realizada esta configuración debemos iniciar este servidor y procedemos con un navegador a probarlo. Ingresamos localhost:2030 en la barra de direcciones, luego ingresamos un nombre válido y el sistema nos devuelve un mensaje con nuestro nombre. Por último cerramos este servidor.

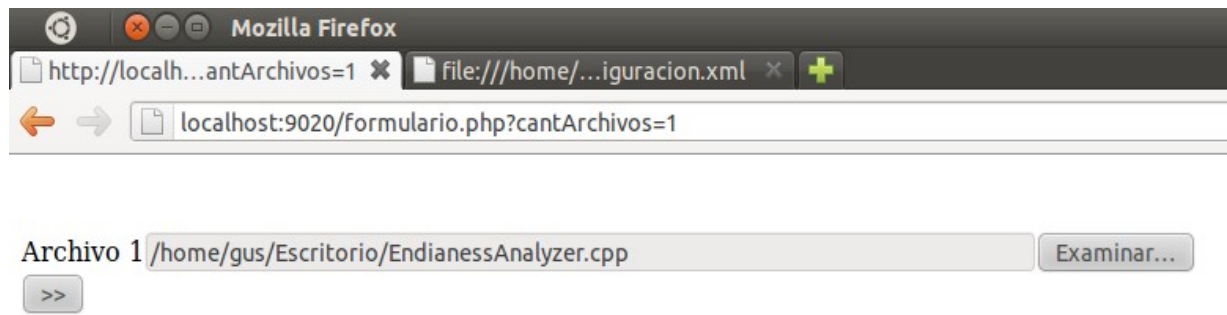
Apéndice II

Pruebas del Precompilador:

En index.html, insertamos la cantidad de archivos que procesaremos y aceptamos:



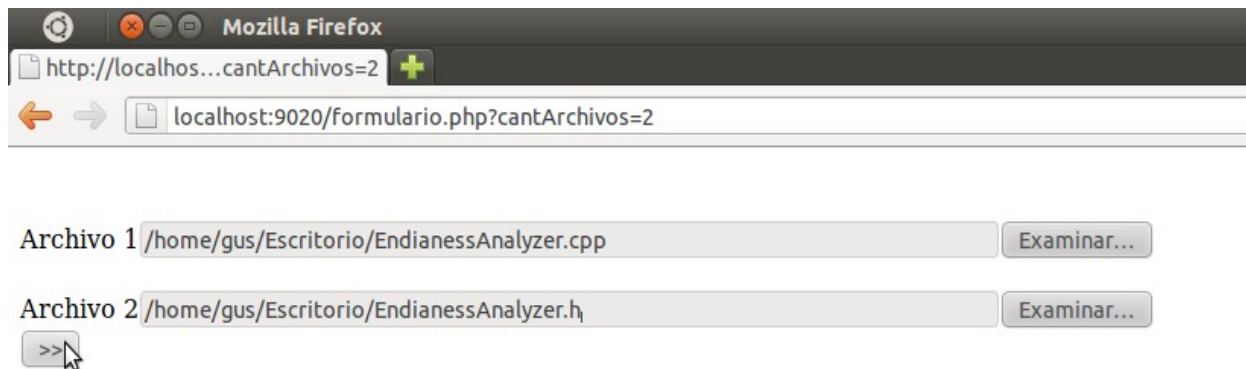
Seleccionamos el recurso principal (en este caso, EndianessAnalyzer.cpp).



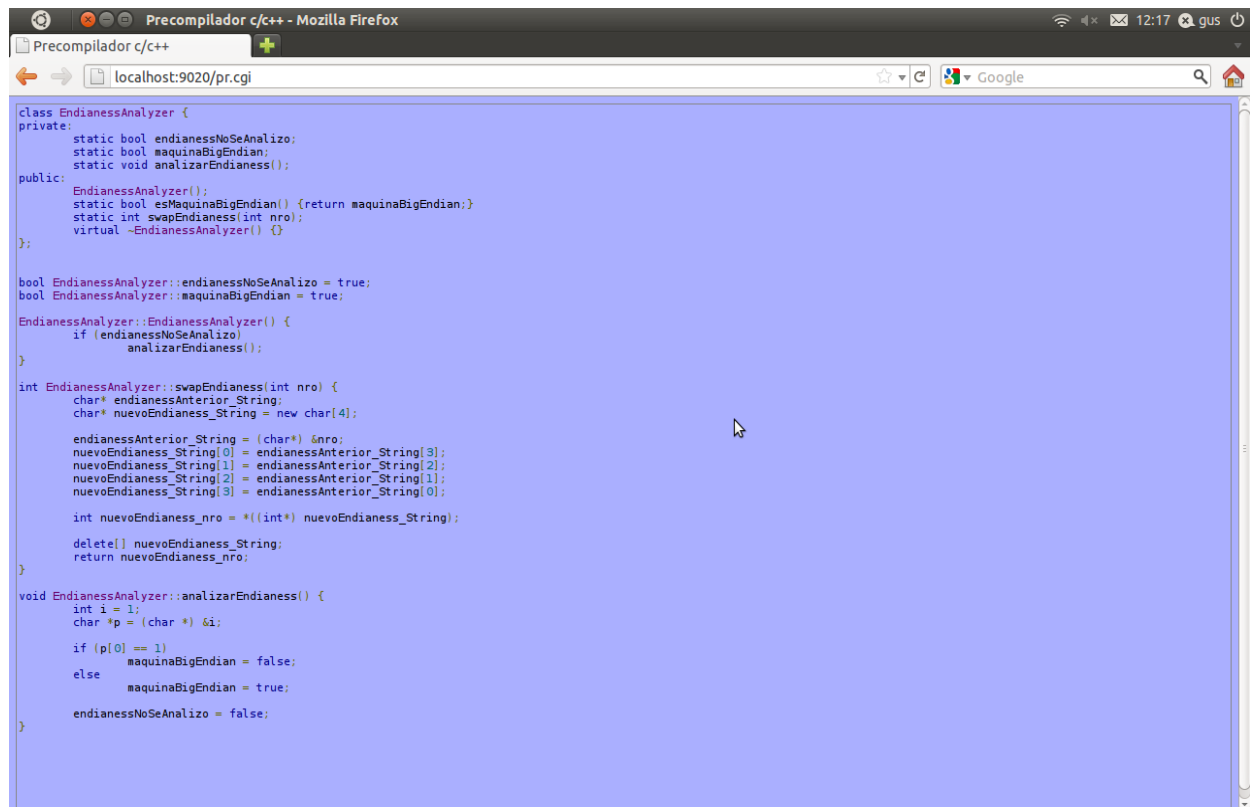
Y obtenemos como salida:



Se nos informa basicamente que nos falto incluir el archivo EndianessAnalyzer.h. Entonces, para obtener la salida deseada, esta vez si lo incluimos:



Y la salida es...



```
class EndianessAnalyzer {
private:
    static bool endianessNoSeAnalizo;
    static bool maquinaBigEndian;
    static void analizarEndianess();
public:
    EndianessAnalyzer();
    static bool esMaquinaBigEndian() {return maquinaBigEndian;}
    static int swapEndianess(int nro);
    virtual ~EndianessAnalyzer() {}
};

bool EndianessAnalyzer::endianessNoSeAnalizo = true;
bool EndianessAnalyzer::maquinaBigEndian = true;

EndianessAnalyzer::EndianessAnalyzer() {
    if (endianessNoSeAnalizo)
        analizarEndianess();
}

int EndianessAnalyzer::swapEndianess(int nro) {
    char* endianessAnterior_String;
    char* nuevoEndianess_String = new char[4];

    endianessAnterior_String = (char*) &nro;
    nuevoEndianess_String[0] = endianessAnterior_String[3];
    nuevoEndianess_String[1] = endianessAnterior_String[2];
    nuevoEndianess_String[2] = endianessAnterior_String[1];
    nuevoEndianess_String[3] = endianessAnterior_String[0];

    int nuevoEndianess_nro = *((int*) nuevoEndianess_String);

    delete[] nuevoEndianess_String;
    return nuevoEndianess_nro;
}

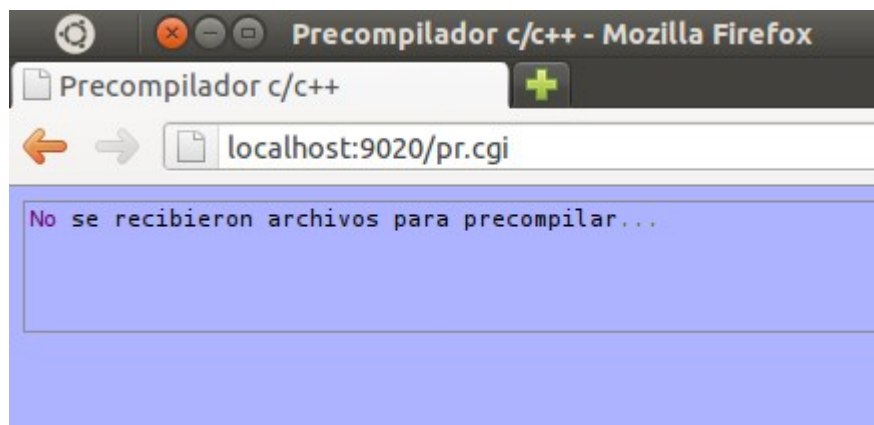
void EndianessAnalyzer::analizarEndianess() {
    int i = 1;
    char *p = (char *) 0;

    if (p[0] == 1)
        maquinaBigEndian = false;
    else
        maquinaBigEndian = true;

    endianessNoSeAnalizo = false;
}
```

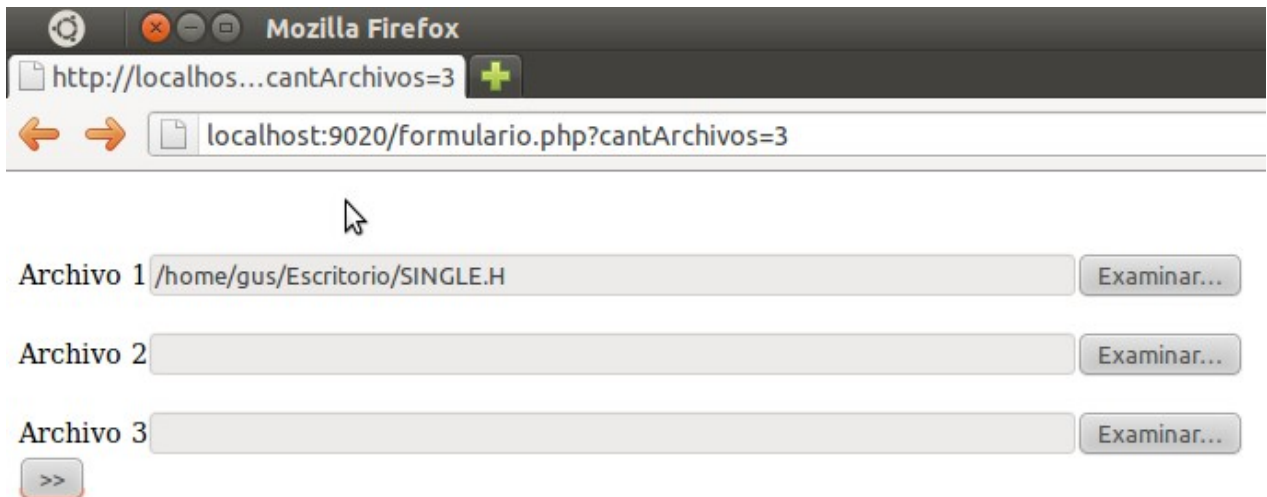
Caso especial: ningun archivo fue enviado.

Tras haber establecido la cantidad de archivos a incluir, si dejamos en blanco todos los campos en donde se fijan los paths de los archivos fuente, se genera la siguiente salida:



Ahora probemos con el archivo SINGLE.H. Establecemos que incluiremos 3 archivos, aun cuando finalmente solo subamos uno solo. El recurso principal siempre sera aquel que se encuentre en el primer campo no vacio.

Colocar el archivo SINGLE.H en la posicion 1 o 3, genera la misma salida:



Mozilla Firefox

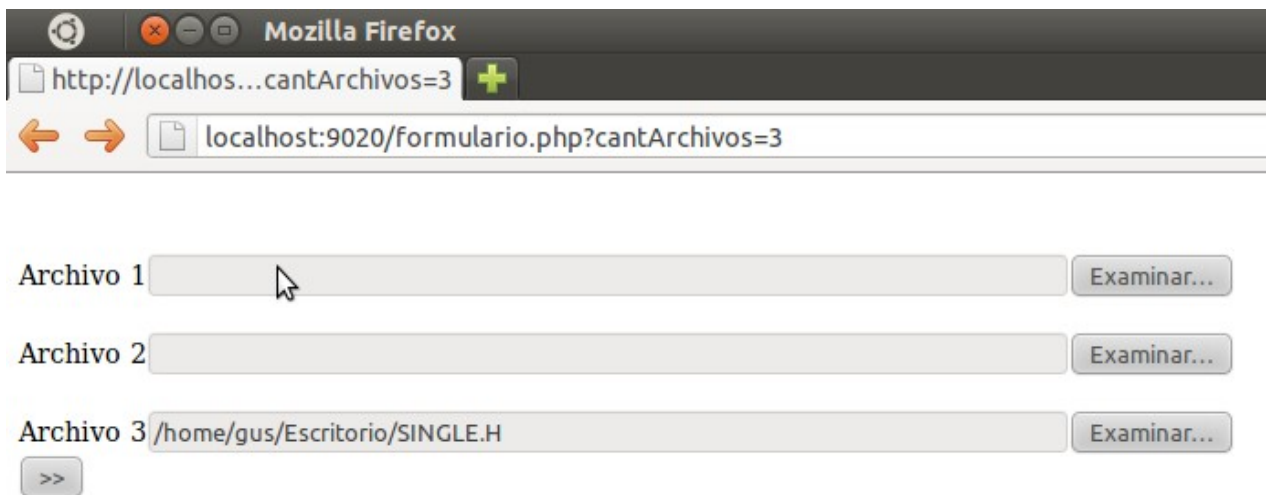
http://localhost:9020/formulario.php?cantArchivos=3

Archivo 1 Examinar...

Archivo 2 Examinar...

Archivo 3 Examinar...

>>



Mozilla Firefox

http://localhost:9020/formulario.php?cantArchivos=3

Archivo 1 Examinar...

Archivo 2 Examinar...

Archivo 3 Examinar...

>>

La salida es:



```
Precompilador c/c++
localhost:9020/pr.cgi

pitufo = 2
Area_Circulo1 = 3.14159*r*r
Area_Circulo2 = r*3.14159*r
Area_Circulo3 = r*3.14159*r
Area_Circulo4 = r*3.14159*r
Area_Circulo5 = r*r*3.14159
Area_Circulo6 = r*r*api
Area_Circulo7 = r*r*3.14159

Autor="Gonzalo Merayo"
```

CGI Precompilador ZIP

Tiene la misma funcionalidad que el precompilador basico, pero se añade la posibilidad de enviar un .zip con todos los fuentes a precompilar. Esto evita tener que seleccionar uno por uno los archivos deseados. Todos los fuentes deben estar en el directorio raiz del archivo zip.

Modo de uso y ejemplos

En la carpeta Pruebas Precompilador, podemos encontrar varios archivos comprimidos. Cada uno de ellos puede ser seleccionado en indexZIP.html. El programa cgi que lo procesa (pr-zip) descomprimira el archivo y precompilara su contenido.

Código fuente