Instituto Tecnológico de Costa Rica
Sistemas Operativos
Documentación - Tarea 3

Walter Benavides Alfaro - 2016136272
Johan Herrera Gutiérrez - 2016133925

Jun 2020

1 Introducción

En la actualidad muchos de los webservers se encuentran subutilizados, pero en periodos de tiempo cortos, es posible que se sobre utilicen debido a la demanda del servicio. Esto sucede con muchos proveedores de entradas a eventos masivos. La configuración normalmente no contempla la limitación de los recursos. Existen dos técnicas que pueden ayudar a administrar mejor los recursos de un servidor, ellas son pre-thread y pre-forked.

Estas técnicas consisten en levantar un servidor web con recursos ya asignados, para el caso de querer utilizar hilos para atender las solicitudes de los usuarios podemos utilizar la función pre-thread, que nos permite levantar el servidor con una cantidad "n" de hilos listos y preparados para atender lo que sea necesario, esto nos evita el proceso de crear hilos durante el tiempo de ejecución. Por otra parte, tenemos la posibilidad de levantar el servidor utilizando procesos ya asignados y creados con anticipación, esta técnica se conoce como pre-fork.

Algunas funcionalidades del servidor web son: poder cambiar de protocolo, es decir, el usuario es capaz de levantar el servidor utilizando los protocolos: FTP, SSH, SMTP, DNS, TELNET, SNMP. Adicionalmente el servidor web implementa los métodos PUT, GET, POST y DELETE.
El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.
El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
El método DELETE borra un recurso en específico.
Adicionalmente el proyecto cuenta con dos clientes http, el primero creado

en Python utilizando la biblioteca Curl, este permite conectarse al servidor y obtener los datos principales del servidor al que se conectó, el segundo creado en C de igual forma se conecta al servidor y recopila los datos principales.
Para poder hacer uso de los hilos y verificar su funcionamiento el proyecto cuenta con un programa llamado StressCMD que se encarga de lanzar un ataque para saturar el servidor web.

2 Ambiente de desarrollo

Las herramientas utilizadas en el desarrollo del proyecto fueron las siguientes:
Sistema operativo GNU/Linux para poder ejecutar nuestro programa desde la terminal utilizando un compilador.
GCC como compilador del lenguaje de programación C, el cual nos permite tomar código programado en C, compilarlo y ejecutarlo desde la terminal de GNU/Linux.
Geany como editor de textos, es decir, nos permite escribir nuestro código en el programa de una forma ordenada y clara. Este editor fue utilizado para desarrollar los servidores web al igual que ambos clientes, uno desarrollado en Python y el otro en C.
Python como intérprete de nuestro cliente http.

3 Estructuras de datos y funciones

Para ambos servidores las funciones más relevantes son:
void createSocket(); /*Función que se encarga de crear el socket que sera utilizado por el web server.*/
void bindSocket(); /*Avisamos al sistema operativo que hemos creado un socket y queremos que una nuestro programa a el.*/
void listeningConnections(); /*Función que se encarga de escuchar las conexiones entrantes al servidor./*
void initializeWebServer(int argc, char *argv[]);/*Función principal para levantar el servidor /*
void acceptIncomingConnections(); /*Función encargada de aceptar conexiones entrantes y crear un nuevo proceso para cada una de ellas.*/
void attendIncomingRequest(int pConnection, char *pRootPath, int pPort, int pProcessMax, int pActiveProcess); /* Función encargada de revisar que tipo de puerto donde se encuentra el servidor/* void attendIncomingHttpRequest(int pConnection, char *pServerRootPath); /*Funcion que se encarga de atender una nueva peticion*/

Fuciones dependiendo del puerto.

Nota: a los puertos se les agregó el 80 al inicio ya que no reconocía un puerto

con dos dígitos.
void attendIncomingFtpRequest(int pConnection, int pActiveProcess, int pProcessMax);
void attendIncomingTelnetRequest(int pConnection);
void attendIncomingDNSRequest(int pConnection);
void attendIncomingSSHRequest(int pConnection);
void attendIncomingSMTPRequest(int pConnection);
void attendIncomingSNMPRequest(int pConnection);
void attendOutputGetRequest(int pConnection, struct RequestInfo *pRequestInfo, char *pServerRootPath, char *pContentType); /*Función principal que se encarga de atender una patición de tipo GET /*

4 Instrucciones para la ejecución del programa

Para la ejecución del programa se necesita el sistema operativo GNU/Linux.
Es necesario descargar el código fuente del proyecto, este se puede encontrar en el repositorio de GitHub ubicado en https://github.com/johanhg/Operativos_tarea1
Tras descargar el código fuente, hay que proceder a extraer el proyecto en alguna carpeta de nuestro sistema operativo.
Seguidamente utilizamos la terminal y los comandos correspondientes para movernos hasta la dirección donde se encuentra nuestro proyecto. Este proyecto consta de dos carpetas principales, la que contiene los servidores y la que contiene los clientes.
Luego nos ubicaremos en la carpeta que contiene los servidores, vamos a proceder a compilar el proyecto utilizando el makefile, para ello debemos abrir la terminal en con la dirección donde se encuentra el makefile y escribir **make** en la terminal y teclear enter, esto nos va a compilar todos los archivos .c y nos va a crear los ejecutables. Seguidamente podemos levantar nuestro propio servidor de dos formas:

En modo pre-thread utilizaremos la sintaxis:

./WebServerThread -n [cantidad-hilos] -w [path] -p [port]

En modo pre-forked la sintaxis será:

./WebServerFork -n [cantidad-hilos] -w [path] -p [port]

Es importante destacar que para nuestro proyecto el [path] indica la carpeta del proyecto "Files", es decir, se debe escribir la ruta del proyecto que apunta a esa carpeta en específico.
Utilizando los comandos anteriores podemos levantar nuestros propios servidores web lo que nos permitiria levantar los clientes que se conecten al servidor y puedan hacer alguna solicitud, para esto debemos abrir una terminal nueva y ubicarnos en la carpeta del proyecto donde se encuentran los clientes.

Luego de movernos hasta la carpeta en la terminal, procedemos a compilar los clientes utilizando el comando del compilador GCC:

gcc httpclient.c -o httpclient

Esto nos crea un ejecutable del cliente, podemos ejecutarlo utilizando el comando:

./httpclient localhost [port]

Tras utilizar ese comando el cliente se conectara con el servidor y nos mostrara en la terminal los datos del servidor.
Adicionalmente tenemos un cliente desarrollado en python y que a través de la biblioteca [curl] nos permite conectarnos al servidor y obtener de igual forma los datos principales. Para esto en la misma terminal y en la misma carpeta de Clientes procedemos a escribir el comando:

python httpclient.py.

Por último tenemos en la misma carpeta un archivo llamado StressCMD.c que necesita ser compilado de igual forma utilizando el comando:

gcc StressCMD.c -o StressCMD

Tras compilarlo podemos ejecutarlo utilizando el comando:

StressCMD [cantidad-hilos] httpclient [parametros del cliente].

Esto nos permite bombardear el servidor con solicitudes para verificar que seamos capaces de atender todas las entradas o requests que le hacemos al servidor.
NOTA:
Para el correcto funcionamiento del proyecto es necesario que se modifiquen dos lineas de codigo:
La primera se encuentra en la linea 56 del archivo httpclient.c en la función [void Get()] debemos escribir la ruta donde localizamos la carpeta Files del proyecto como se menciono anteriormente.

La segunda se encuentra en el archivo AttendRequest.c que se encuentra en la carpeta de los servidores. Ahí es necesario modificar la línea 8 para de igual forma escribir la ruta donde encontramos la misma carpeta Files mencionada anteriormente.

5 Actividades realizadas por los estudiantes

06.06.2020 Reunión grupal, buscar información. Duración: 1.5h
07.06.2020 Llamada para la creación del Kick-Off Duración: 2h
09.06.2020 Ajustes en el Kick-Off Duración: 0.5h
11.06.2020 Investigar sobre Thread y Fork Duración: 5h
12.06.2020 Buscar información sobre servidores(sockets) en C y generar código de prueba Duración: 5h
13.06.2020 Se logró levantar un servidor Duración: 3h
14.06.2020 Implementación del servidor en modo prefork y prethread, junto con los dos tipos de clientes Duración: 7h
16.06.2020 Implementación del GET para devolver un "Login" en html a los clientes Duración: 2h
17.06.2020 Gerenerar el archivo StressCMD Duración: 2h
18.06.2020 Afinar detalles del código Duración: 2h
19.06.2020 Generar el documento final y generar varias pruebas al servidor. Duración: 6h

Total de horas: 36 aproximadamente.
Es importante mencionar que todas las actividades fueron realizadas simultaneamente por ambos integrantes del grupo.

6 Autoevaluación

Rúbrica:
[1] [2] [3] [4] [5] Aprendizaje de pthreds(1).
[1] [2] [3] [4] [5] Aprendizaje de forks.(2)
[1] [2] [3] [4] [5] Aprendizaje de comunicación entre procesos.(3)
[1] [2] [3] [4] [5] Aprendizaje de sockets.(4)

TOTAL: 20

Johan Herrera:
(1) - [3]
(2) - [3]
(3) - [3]
(4) - [4]

TOTAL: 13

Walter Benavides:

(1) - [3]
(2) - [3]
(3) - [3]

(4) - [4]

TOTAL: 13

7 Lecciones aprendidas

Nosotros como estudiantes del presente curso, podemos decir que el aprendizaje fue bastante intenso pues la mayoría de conceptos antes de iniciar el proyecto no eran bien conocidos a fondo por nosotros y verdaderamente no teniamos idea de como funcionan estos conceptos.
Como ejemplo podemos brindar los conceptos de pre-thread, pre-forked, sockets, servidor web y clientes http.
Podemos comentar que aprender sobre esos conceptos y como aplicarlos fue bastante interesante y posiblemente de gran ayuda pues para cursar nuestra carrera es necesario conocer bien el funcionamiento de los servidores web y como interactuar con ellos.

Recomendamos a los futuros estudiantes del presente curso tomarse el tiempo para comprender a fondo los conceptos necesarios y no comenzar a programar solo con una base del conocimiento, es necesario analizar y tener claro lo que se debe hacer.

8 Bibliografía

prefork.c is not shown in httpd -l. (2018, 26 noviembre). Recuperado de https://stackoverflow.com/questions/53489233/prefork-c-is-not-shown-in-httpd-l

M. (2018, 29 agosto). Diferencias entre los módulos de procesamiento de Apache: Prefork, Worker y Event. Recuperado de https://soyadmin.com/diferencias-entre-apache-mpm-prefork-worker-y-event/

Multithreading in C. (2018, 10 octubre). Recuperado de https://www.geeksforgeeks.org/multithreading-c-2/

Thread functions in C/C++. (2019, 13 marzo). Recuperado de https://www.geeksforgeeks.org/thread-functions-in-c-c/

Programación en C/Sockets - Wikilibros. (2020, 6 junio). Recuperado de https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Sockets

fork(). Parte I: ,Creación de un nuevo proceso, hijos, padres, zombies y huérfanos. (2019, 29 abril). [Archivo de vídeo]. Recuperado de https://www.youtube.com/watch?v=VwjP-KFuZCM&t=648s

Fork - Creando Procesos en C. (2018, 24 enero). [Archivo de vídeo]. Recuperado de https://www.youtube.com/watch?v=Xm9Eah8t0d4

Y. (2015, 23 agosto). YeisonCL/WebServerC. Recuperado de https://github.com/YeisonCL/WebServerC

Program your own web server in C. (sockets). (2018, 20 noviembre). [Archivo de

vídeo]. Recuperado de https://www.youtube.com/watch?v=esXw4bdaZkc&t=604s
How to build a web client? (sockets). (2018, 12 octubre). [Archivo de vídeo].
Recuperado de https://www.youtube.com/watch?v=bdIiTxtMaKA&t=186s
cliente.c. (2011). Recuperado de https://gist.github.com/movihus/444832/29de448ccf76f98e2dd3ede3e27b1
Programacin Bsica de Sockets en Unix para Novatos. (2016). Recuperado de
http://es.tldp.org/Tutoriales/PROG-SOCKETS/prog-sockets.html
PYTHON SOCKETS - CHAT TUTORIAL. (2016, 2 julio). [Archivo de vídeo].
Recuperado de https://www.youtube.com/watch?v=McbFhgG_VE0
Redes de Computadores. Cliente - Servidor en Python. (2015, 15 junio).
[Archivo de vídeo]. Recuperado de https://www.youtube.com/watch?v=62OgxwGCjmM
Multiple Client Server Program in C using fork — Socket Programming. (2017,
18 diciembre). [Archivo de vídeo]. Recuperado de https://www.youtube.com/watch?v=BIJGSQEipEE
How to write a multithreaded server in C (threads, sockets). (2019, 20 septiembre). [Archivo de vídeo]. Recuperado de https://www.youtube.com/watch?v=Pg_4Jz8ZIH4&t=89s
Simple socket server in C using threads (pthread library) Compiles on linux.
(2017). Recuperado de https://gist.github.com/oleksiiBobko/43d33b3c25c03bcc9b2b
multithread server/client implementation in C. (2014, 28 enero). Recuperado de
https://stackoverflow.com/questions/21405204/multithread-server-client-implementation-in-c