Instituto Tecnológico de Costa Rica

Facultad de Ingeniería en Computación

Mi DDoS
Sistemas Operativos

Tarea 2

Fabrizio Alvarado Barquero
2017073935
Kevin Ledezma Jiménez
2017109672

Noviembre 2020

0.1 Introdución

El propósito del presente trabajo es aprender sobre el uso de servidores en lenguaje C, donde el objetivo es aprender sobre el concepto de thread y fork, como estos funcionan y también su comportamientos, todo esto mediante la implementación de los servidores utilizando las técnicas prethread y preforked para manejar las conexiones además de solicitudes por parte de clientes.

Ambos servidores tendrán el mismo comportamiento, donde su función principal es el manejo de archivos tales como imagenes, videos, paginas web, entre otros.

Estos servidores tendrán características específicas según estos sean invocados, donde algunas de sus variantes son la dirección del sistema de archivos, el número de procesos o hilos máximos en ejecución y demás.

0.2 Ambiente de desarrollo

- Oracle Virtual Box:
Sistema emulador de Ubuntu.

- Geany:
Editor de texto.

- IntelliJ:
Editor de texto.

- GitHub:
Control de versiones.

- Mozilla Firefox:
Navegador web utilizado para accesar a los puestos de los servidores.

0.3 Estructuras de datos usadas y funciones

- PreThread
  - Función: connection_handler: Encargada de las operaciones a realizar para cada thread creado.
  - Función: processParameters: Encargada de obtener los datos de los argumentos en consola y asignarlos a variables locales para el uso durante la ejecución.
  - Función: main: Encargada de la ejecución principal de la asignación.
- Arrays
  - Para la pre-creación de hilos con PreThread
  - Para los file descriptors de los clientes en el PreForked
  - Para el contenido html (el string")
- PreForked
  - Función: spaceAvailable: Encargada de verificar si permite más conexiones entrantes o si se llenaron los espacios.
  - Función: startPreforkWebServer: Encargada de crear el servidor con la técnica "Pre Forked" para recibir conexiones de clientes.
  - Función: createServer: Encargada de obtener los parámetros de los argumentos en consola y configurar las variables globales para el uso dentro de la ejecución.
  - Función: ImageOpen: función para obtener los datos de los recursos (imagen) del enlace provisto.
  - Función: main: Encargada de la ejecución principal de la asignación.
- httpClientPython
  - Curl: Función y biblioteca escrita en Python, encargada de solicitar y obtener un recurso a partir de un enlace provisto.
- protocols.c
  - Archivo con la intención de manejar la múltiples funciones correspondientes para los protocolos de los distintos puertos web, entre ellos:
    - HTTP
    - FTP
    - SSH
    - TELNET
    - SMTP

        * DNS
        * SNMP

- stressClient
    - Encargado de crear multiples solicitudes a partir de un cliente en formato binario, con la respectiva configuración desde los argumentos de terminal
    - Función: attack: Encargada de recibir una instrucción previamente con formato para terminal, invocar al os.cmd y ejecutarla en varios threads.

0.4 Instrucciones para ejecutar programa

- Generación de binario de PreForkWebServer:
gcc -o PreForkWebServer PreForkWebServer.c

- Ejecución de binario de PreForkWebServer:
./PreForkWebServer -n [cantidad procesos] -w [path de archivos] -p [puerto]

- Generación de binario de PreThreadWebServer:
gcc -o PreThreadWebServer PreThreadWebServer.c -pthread

- Ejecución de binario de PreThreadWebServer:
./PreThreadWebServer -n [cantidad procesos] -w [path de archivos] -p [puerto]

- Generación de binario de httpClientC:
gcc -o httpClientC httpClientC.c

- Ejecución de binario de httpClientC:
./httpClientC -u [ruta de recurso a obtener]

- Ejecución de httpClient.py:
python httpClient.py -u [ruta de recurso a obtener]

- Ejecución de stressClient:
python stressClient.py -n [cantidad de hilos] httpClientC -u [parametros del cliente]

0.5 Actividades realizadas por estudiante

NOTA: Se da por hecho que los tiempos y las tareas realizadas en esta bitácora son realizadas por ambos estudiantes.

Sábado 24 de octubre - 3 horas
-Investigación sobre clientes y servidores en C.
-Investigación sobre hilos y procesos en C.
-Preparación de ambiente de trabajo, así como instalación de python, git y demás.

Problemas: Ninguno hasta el momento, la sesión fue únicamente una introducción al tema.

Domingo 25 de octubre - 4 horas
-Búsqueda de ejemplo de servidores y clientes tanto en c como en python, para darnos una idea de como funcionan este tipo de sistemas en los lenguajes.
-Interacción con ejemplo de prueba encontrado en internet, link 1.

Problemas: Muchas dudas en cuanto al funcionamiento de algunos métodos de las bibliotecas. Poca práctica últimamente con respecto al lenguaje C.

Lunes 26 de octubre - 3 horas
-Creación del repositorio de git.
-Investigación a fondo de los métodos mencionados anteriormente.
-Continua interacción con el ejemplo de prueba para ir comprendiendo mejor el funcionamiento de los sistemas.

Problemas: Nos sentimos un poco desorientados en cuanto a las partes del servidor y del cliente (socket, binding, listener, etc).

Martes y miércoles 27-28 de octubre - 5 horas
-Investigación sobre clientes tanto en python como en C
-Investigación de uso de argumentos
-Inicio a partir del cliente de prueba de C que teníamos previamente
-Documentación de código

Problemas: Sorpresivamente no tuvimos muchos problemas, hay bastante documentación en internet sobre el tema.

Viernes y sábado 30-31 de octubre - 3 horas
-Cambios menores sobre cantidad de conexiones en el servidor.
-Inicio de investigación sobre stress client en python.
-Investigacion sobre biblioteca Treq

Problemas: Treq.

Domingo, lunes, martes 1-2-3 de noviembre - 9 horas:
-Pruebas con el servidor pre thread
-Entendiendo el funcionamiento de sus métodos y cómo se manejan los hilos
-Se inició con el stress cliente.

Problemas: Muchos en realidad, pero esencialmente, ninguno meramente importante.

Miércoles 4 de noviembre - 6 horas:
-Cambios menores sobre cantidad de conexiones en el servidor.
-Inicio de investigación sobre stress client en python.
-Investigacion sobre biblioteca Treq

Problemas: Treq.

Miércoles 4 de noviembre - 6 horas:
-Pruebas con el servidor pre forked
-Entendiendo el funcionamiento de sus métodos y cómo se manejan los procesos

Problemas: Errores en cuanto a manejo de archivos.

Jueves 5 de noviembre - 8 horas:
-Finalización y corrección de problemas tanto el server prethread como preforked
-Corrección del stress client.
-Documentación

Problemas: Archivos a obtener.

0.6 Autoevaluación

- Estado final:
El estado final de los servidores es tal como se pidió en la especificación, con los metodos PreThread y PreFork un sistema de servidores ejecutables que permite obtener recursos de un sistema de archivos.

- Problemas:
Tiempo y conocimiento del tema por lo que nos resultó imposible la finalización del CGI.

- Limitaciones:
El tiempo fue un factor muy importante, además de

- Evaluación:

Característica	Valor	Fabrizio	Kevin
WebServer	40	30	30
Protocolos	10	10	10
httpClient-C	5	5	5
httpClient-PY	5	5	5
stressClient-PY	10	10	10
CG- Injector	10	0	0
Documentación	20	20	20
Total	100	80	80

-Repositorio de GitHub:
https://github.com/faoalvarado5/MyDDos

-Reporte de commits:
https://github.com/faoalvarado5/MyDDos/commits?
Al igual que se encuentra un archivo de txt llamado "commitlog.txt" con el registro de los commit del repositorio de git.

0.7 Lecciones aprendidas

Fabrizio:

La verdad me gustaría decir que es una tarea bastante complicada, porque hay que aprender muchos tecnisismos sobre todo en realidad, sobre los hilos, procesos, servidores, clientes y demás, cosa que al menos en mi caso no estoy acostumbrado a trabajar a este nivel.
Por otra parte, la verdad creo que si logré entender como es que estos funcionan y se comportan a la hora de utilizarlos.

Kevin:

Esta asignación fue desafiante, por que nos hace estudiar un concepto nuevo que no es fácil de comprender. Me hubiera gustado tener mayor tiempo (no se confíen, puede ser sencilla pero consumirán horas estudiando lo que deben hacer) y tener también mayor claritud en cuanto a lo que sucede en estos procesos y aún más claridad en cuanto a su implementación. Me siento "satisfecho" con el progreso que hicimos, pero siento que podemos demostrar más, solo nos queda seguir trabajando duro.

0.8 Bibliografía

- Socket Programming in C/C++ - GeeksforGeeks. GeeksforGeeks. (2020). Retrieved 5 November 2020, from https://www.geeksforgeeks.org/socket-programming-cc/.

- Using cURL in Python with PycURL. Stack Abuse. (2020). Retrieved 5 November 2020, from https://stackabuse.com/using-curl-in-python-with-pycurl/.

- Multiple Client Server Program in C using fork — Socket Programming(2020). Retrieved 5 November 2020, from https://www.youtube.com/watch?v=BIJGSQEipEE.

- nikhilroxtomar/Multiple-Client-Server-Program-in-C-using-fork. GitHub. (2020). Retrieved 5 November 2020, from https://github.com/nikhilroxtomar/Multiple-Client-Server-Program-in-C-using-fork/blob/master/tcpServer.c.

- Python, P., KR, K. (2020). Process Multiple Requests Simultaneously and return the result using Klein Module Python. Stack Overflow. Retrieved 5 November 2020, from https://stackoverflow.com/questions/41392710/process-multiple-requests-simultaneously-and-return-the-result-using-klein-modul/41406549.

- POSIX Threads Programming. Computing.llnl.gov. (2020). Retrieved 5 November 2020, from https://computing.llnl.gov/tutorials/pthreads/Pthread.

- $\text{pthread}_c reate(3) - Linux manual page. Man7.org.(2020). Retrieved 5 November 2020, from https : //man7.org/linux/man - pages/man3/\text{pthread}_c reate.3.html.$

$-C, m., Deep, S., Deep, S.(2020). multithreadserver/clientimplementationinC. StackOverflow. Retrieved 5 November 2020, from https : //stackoverflow.com/questions/21405204/multithread - server - client - implementation - in - c.$

$-RedAndBlueEraser/c - multithreaded - client - server. GitHub.(2020). Retrieved 5 November 2020, from https : //github.com/RedAndBlueEraser/c - multithreaded - client - server/blob/master/server.c.$

$-14 commonnetworkportsyoushouldknow. Opensource.com.(2020). Retrieved 5 November 2020, from https : //opensource.com/article/18/10/common - network - ports.$