Tarea Corta 2: NetNeutralityMac

Pamela Ortiz Fuentes

28 de Marzo 2018

1 Introducción

NetNeutralityMac consiste en crear una simulación del comportamiento del Net-Neutrality en un restaurante de comida rápida de hamburguesas. El objetivo de esta tarea en implementar dicha simulación con una interfaz para visualizar las ordenes para un programa en lenguaje C. Para la tarea se piensa usar prethreads y preforkeds para distriburi las ordenes de hamburgesas.

2 Ambiente de Desarrollo

Para poder desarrollar esta tarea se necesitan de las siguientes herramientas:

- Geany: Editor de código.
- Lenguaje C: Lenguaje a usar para el desarrollo.
- GCC: Compilador para el lenguaje C.
- gtk/gtk.h: Biblioteca para desarrollar la interfaz.
- pthread.h: Biblioteca para el desarrollo de threads.
- fork: Metodo para crear procesos.

3 Estructuras de datos usadas y funciones

El comportamiento general del programa de simulación es la creación de un servidor con prethreads o preforks que estén precargadas en conjunto con la ejecución de la interfaz, y un cliente que mandara solicitudes sobre ordenes de hamburguesas. Dentro de el sistema destacan los siguientes de archivos:

- thpool.c: Es una piscina de hilos pre-cardos, esperando que se le asigne un tarea.
- servidorPrethread.c: Es un servidor creado por configuración, y donde se escucharan las peticiones del cliente; inicializa una cantidad de hilos, y ejecuta el programa de interfaz con un hilo independiente.

- servidorForks.c: Es un servidor creado por configuración, y donde se escucharan las peticiones del cliente; inicializa una cantidad de procesos, y ejecuta el programa de interfaz con un hilo independiente.
- cliente.c: Contiene la estructura del cliente, que envía mensajes a un servidor por configuración.
- ordenes.c: Es el 'main' del cliente, agarra los componentes del cliente y los usa para enviar un mensaje al un servidor por un puerto determinado por configuración, un cliente solo puede pedir una hamburgesa a la vez.
- hamburgesa.c: Es la estructura basica de las solicitudes de hamburguesas, con clasificación de prioridad (ALTA/MEDIA/BAJA), esta clasificación es para separar el 'tiempo de cocimiento' para las hamburguesas.
- interfaz.c: Contiene la estructura de la interfaz en GTK, la compone: un botón para refrescar los mensajes de ordenes desde un archivo, una transición de imágenes para mostrar el progreso de la hamburguesa, y un label que muestra el mensaje escrito por el servidor acerca de las ordenes.
- archivos.c: Tiene métodos para poder escribir y leer de un archivo (hamb_mc.txt); este archivo se usa como medio para intercambiar información entre el servidor y la interfaz, así mostrar las hamburguesas que se están en procesos y cuando están listas.

Dentro del sistema destacan las funciones principales:

- inicializar_servidor_prethreads() :: servidorPrethread.c: Función donde se tiene el servidor escuchando al cliente, y donde la interfaz se inicio a correr, se le asigno la función inicial de la interfaz a un hilo extra para su ejecución, también, donde se cargan los prethreads con una cantidad indicada.
- inicializar_servidor_forks() :: servidorForks.c: Función donde se tiene el servidor escuchando al cliente, y donde la interfaz se inicio a correr, se le asigno la función inicial de la interfaz a un hilo extra para su ejecución, también, donde se cargan los preforks con una cantidad indicada.
- inicializar_cliente() :: cliente.c: Carga configuraciones para iniciar el cliente, se le es pasado el puerto y el ip de conexión.
- envia_orden :: cliente.c: Envía el mensaje de texto de orden al servidor conectado.
- main() :: ordenes.c: Es donde recibe los parametros de configuración para el cliente, y tambien donde procesa la entrada para decidir si es un cliente o un stress, para la ejecición de ordenes al servidor.
- cocinar_hamburgesa_alta() :: hamburgesa.c: Es un metodo que al igual cocinar_hamburgesa_media() y cocinar_hamburgesa_baja, ejecutan un sleep

personalizado para cada función de menor a mayor tiempo correspondiente a la prioridad, también, antes y despues del sleep escriben mensajes al final del archivo hamb_mc.txt, para indicar las acciones que realizan.

- interfaz() :: interfaz.c: Funcion principal que ejecuta y carga la ventana interfaz, y que mediante un botón se muestra información del archivo hamb_mc.txt.
- leer_fichero() y escribir_fichero() :: archivos.c: Manejan la entrada y salida de caracteres del archivo hamb_mc.txt .

4 Instrucciones para ejecutar el programa

Para ejecutar el programa se debe estar en le fichero /src donde esta el código del programa.

- Compilar el programa $ make all.
- Ejecutar un servidor de prethreads, como por ejemplo: $ ./prethread-Server -p 6667 -n 7 -r /home/usuario/Imagenes/hamburgesas
- Ejecutar un servidor de preforks, como por ejemplo: $ ./preforked-Server -p 6667 -n 7 -r /home/usuario/Imagenes/hamburgesas
- Luego de ejecutar alguno de los dos servidores, ejecutar el cliente, ejemplo: $ ./client -h 127.0.0.1 -c media -p 6667
- Un ejemplo para ejecutar el cliente de Stress:, ejemplo: $ ./stress -n 10 -h 127.0.0.1 -c media -p 6667

Para ejecutar el Cliente o el Stress se debe tener primero creado el servidor con los hilos o los procesos.

5 Actividades realizadas por el estudiante

12-03-18 — 2 hr — Investigación sobre manejo de hilos.

12-03-18 — 3 hr — Construcción con manejo de hilos.

13-03-18 — 1 hr — Investigación sobre manejo de forks.

13-03-18 — 5 hr — Construcción del manejo del forks.

13-03-18 — 1 hr — Investigación y prueba del servidor.

15-03-18 — 2 hr — Investigación y prueba del servidor con un cliente.

15-03-18 — 1 hr — Realizar construcción del cajero para ordenes.

15-03-18 — 1 hr — Conexión cliente-servidor con ordenes de hamburguesas.

17-03-18 — 3 hr — Asignar ordenes a prethreads.

17-03-18 — 4 hr — Investigación y prueba la asignación de funciones a forks.

18-03-18 — 4 hr — Investigación y prueba la asignación de funciones a forks.

19-03-18 — 3 hr — Prueba y contrucción de la aplicación Stress.

19-03-18 — 3 hr — Investigar y prueba la asignación de funciones a forks.

19-03-18 — 1 hr — Construcción del Server con Forks.

22-03-18 — 2 hr — Modificación del Client y el Stress.

22-03-18 — 2 hr — Investigar ejemplos del GTK con C.

24-03-18 — 2 hr — Investigar y probar la interfaz GTK.

24-03-18 — 4 hr — Construir base de interfaz GTK.

25-03-18 — 4 hr — Prueba de modificación sobre el manejo de prethreads.

26-03-18 — 4 hr — Prueba de modificación sobre el manejo de prethreads.

26-03-18 — 2 hr — Modificar la interfaz GTK.

27-03-18 — 3 hr — Prueba de modificación sobre el manejo de prethreads.

27-03-18 — 4 hr — Investigación e integración sobre modificación de imágenes en GTK.

28-03-18 — 1 hr — Investigación e integración de refrescar información en GTK.

28-03-18 — 4 hr — Investigación, construcción e integración sobre manejo de archivos.

28-03-18 — 4 hr — Realización de la documentación.

Total: 70 hr

6 Comentarios Finales

El programa llego a funcionar solo con el server de prethreads, con los forks se logro hacer que funcione, faltan correcciones a este servidor de forks del porque ya no escucha al cliente. También, los servidores no se lograron conectar por el puerto 80 con el protocolo HTTP, se conectan el cliente y el servidor por medio que cualquier puerto que sea diferente al 80.

7 Conclusión y Recomendaciones del proyecto

La tarea en primera entrada se ve compleja a realizar, por por el manejo de threads, y en esspecial de forks. El punto de partida que se pudo identificar es el poder manejar los prethreads y los forks, iniciando con la búsqueda de de la biblioteca y ejemplos sobre su uso, en un inicio no se veia complicado manejar los threads y los forks, sin embargo, con forme se avanzaba en el desarrollo de

la tarea se noto que era mas complicado de lo que se notaba, por lo que se perdió tiempo en la búsqueda del manejo los hilos y forks. Al notarse que era complicado iniciar con los hilos y forks, se paso al manejo de cliente-servidor con la simulación del cajero, que fue mas sencilla de desarrollas a comparación de los forks, existió muchas pruebas y errores, llegando hasta un estado en donde servía la tarea sin contar los forks. Como recomendaciones para esta tarea serian puntualmente uno; dedicar tiempo para comprender el funcionamiento de los forks, ya que es la parte mas complicada de esta tarea, y si no se investiga bien se pierde el control sobre esta.

8 Bibliografía

- Pubs.opengroup.org. (2018). ¡pthread.h¿. Recuperado de: http://pubs.opengroup.org/onlinepubs/79087
- Paszniuk, R. (2013). Creación y duplicación de procesos en C (Parte I) – Linux – Programación. Programacion.com.py. Recuperado de: https://www.programacion.com.py/escrit y-duplicacion-de-procesos-en-c-linux
- Pithikos, M. (2017). Pithikos/C-Thread-Pool. [online] GitHub. Recuperado de: https://github.com/Pithikos/C-Thread-Pool
- Felice, M. (2015). sisoputnfrba/so-test-sockets. GitHub. Recuperado de: https://github.com/sisoputnfrba/so-test-sockets/blob/master/Servidor/src/Server.c
- Felice, M. (2014). sisoputnfrba/so-test-sockets. GitHub. Recuperado de: https://github.com/sisoputnfrba/so-test-sockets/blob/master/Cliente/src/Cliente.c
- Developer.gnome.org. (2015). Getting Started with GTK+: GTK+ 3 Reference Manual. Recuperado de: https://developer.gnome.org/gtk3/stable/gtk-getting-started.html
- Gale, T. (2000). GTK v1.2 Tutorial. Gtk.org. Recuperado de: https://www.gtk.org/tutorial1.2/gtk$_t ut.ht$

- ARORA, H. (2012). File Handling in C with Examples (fopen, fread, fwrite, fseek). Thegeekstuff.com. Recuperado de: https://www.thegeekstuff.com/2012/07/c-file-handling