Evaluador de servidores con diferentes procesos

Arturo Córdoba-Villalobos, Fabián González-Araya, José Carlos Núñez-Valverde
email: arturocv16@gmail.com, fagonar96.tec@gmail.com, jcnv21@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Abstract*—El proyecto consiste en implementar una arquitectura cliente-servidor de tres formas diferentes, mejor dicho, con un mecanismo distinto en cada uno de ellos de procesar las necesidades del cliente. Los servidores aplican un filtro Sobel en imágenes de cualquier tamaño, la misma una vez filtrada se guardara 100 veces (las primeras 100 procesadas), esto en una folder/directorio correspondiente para cada instancia y cada servidor inicializado.**

***Palabras clave*—Cliente, Servidor, Filtro, Sobel.**

I. Introducción

El procesamiento en paralelo es el método mediante el cual una serie de tareas e instrucciones se ejecutan de forma simultánea. Como cualquier trabajo en paralelo, se trata de dividir el trabajo en trozos más simples, que actualmente solemos llamar hilos, threads o subprocesos [1]. Por lo tanto, se trata de manejar partes separadas de una tarea general y esto ayudará a reducir la cantidad de tiempo para ejecutar un programa.

Por lo general, un ingeniero en computadores o informática dividirá una tarea compleja en varias partes con una herramienta de software y asignará cada parte a una unidad de procesamiento, luego cada unidad resolverá su parte y una herramienta de software se encargará de volver a ensamblar los datos para leer la solución o ejecutar la tarea. Este documento presenta tanto el diseño del programa como las instrucciones de utilización de una arquitectura de cliente-servidor de tres maneras distintas.

El proyecto consiste en la creación de tres servidores independientes cuya función sea recibir consultas y procesar lo que los clientes desean. Cada servidor tiene su propio ejecutable, sin embargo, todos cumplen la misma función que consiste en recibir una imagen de cualquier tamaño y aplicar el filtro Sobel, algoritmo detector de bordes, a dicha imagen. Una vez aplicado el filtro, se almacenan las primeras 100 imágenes procesadas en un directorio distinto para cada servidor. El primer servidor utiliza un esquema FIFO para atender las solicitudes. El segundo servidor, denominado heavy process, crea un nuevo proceso por cada solicitud recibida. El tercer servidor, denominado pre heavy process, crea al inicio de la ejecución una cantidad n de subprocesos, los cuales se encargan de atender las solicitudes, el proceso principal es el responsable de asignar los clientes a aquellos subprocesos que se encuentren en un estado de disponible.

II. Ambiente de Desarrollo

En el desarrollo del proyecto se utilizaron diferentes herramientas para llevar a cabo el diseño e implementación de la solución al problema planteado. Como sistema operativo se utilizó Ubuntu 20.04, con el fin de utilizar las llamadas al sistema del estándar POSIX. Se utilizó el lenguaje de programación C, el cual fue compilado con la ayuda del GNU Compiler Collection (GCC), cuyas instrucciones fueron automatizadas con un Makefile para simplificar este proceso. Se utilizó Visual Studio Code para la escritura y prueba de los archivos fuente desarrollados. Se utilizó la biblioteca libpng, la cual permite el procesamiento de imágenes cuyo formato es exclusivamente PNG. Para la conexión entre el cliente y el servidor se utilizó la biblioteca nativa de C para el desarrollo de sockets, y se utilizaron unix domain sockets para la trasferencia de file descriptors. Se utilizó Git para llevar un control de versiones del código desarrollado, en conjunto con GitHub y GitKraken, este último permite el manejo de Git a través de una interfaz gráfica. Se utilizó Octave para realizar el visualizador.

III. Diseño del programa

En esta sección se detallará el diseño de la solución implementada. Se utilizarán diagramas de flujo para explicar los pasos realizados por cada tipo de servidor, se empleará un diagrama de secuencia para detallar la interacción entre el cliente y los servidores, y con el uso de un diagrama de arquitectura se desarrollará la estructura de la solución.

El cliente fue construido de tal manera que puede interactuar con cualquiera de los tres servidores sin importar su tipo. Se reciben como parámetros la dirección ip y puerto del servidor, la ruta de la imagen que debe ser enviada, la cantidad de hilos que deben crearse y la cantidad de ciclos que realiza cada hilo. La interacción entre el cliente y el servidor se explicará más adelante por medio de un diagrama de secuencia.

En la figura 1 se muestra el diagrama de flujo del servidor secuencial. Este tipo de servidor se encarga de atender las solicitudes con un esquema FIFO. Al iniciar la ejecución se crea una carpeta exclusiva para la instancia del servidor, y se procede a esperar las solicitudes de conexión por parte de los clientes, cuando se recibe una se realiza la recepción de la imagen y su filtrado, si el número de solicitudes totales es menor a 100, entonces se guarda la imagen. Se aumenta el contador de solicitudes, y se vuelve a esperar la conexión de un nuevo cliente.

En la figura 2 se puede observar un diagrama de flujo que explica los pasos que sigue el servidor heavy process. Cuando se inicia el programa se crea una carpeta exclusiva para la instancia del servidor donde se almacenan las primeras 100

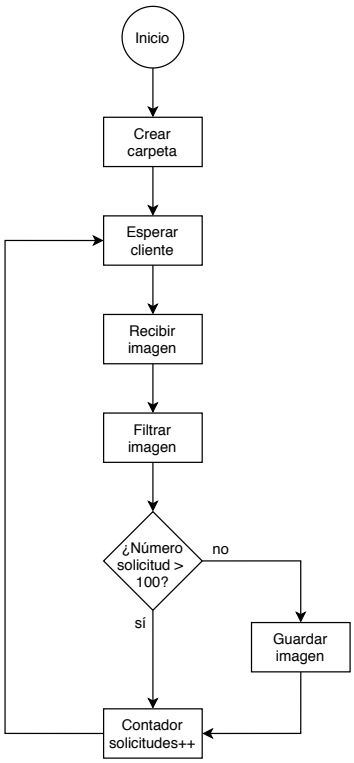

Fig. 1: Diagrama de flujo del servidor secuencial

imágenes recibidas. Posteriormente, se espera por la conexión de nuevos clientes, cuando se recibe una nueva solicitud, se crea un nuevo proceso utilizando la llamada al sistema *fork*, el proceso padre vuelve a esperar por nuevas conexiones y el proceso recién creado se encarga de atender la solicitud del cliente. El proceso hijo es el responsable de recibir la imagen, realizar el filtrado y determinar si se debe almacenar el resultado en disco, una vez finalizados estos pasos el proceso hijo muere.

En la figura 3 se muestra el diagrama de flujo del proceso principal del servidor pre heavy process. El proceso principal es el encargado de crear la carpeta de uso exclusivo para la instancia del servidor, y de crear los subprocesos encargados de atender las solicitudes realizadas por los clientes. La cantidad de subprocesos que deben crearse se recibe como argumento cuando se ejecuta el programa. Una vez creados todos los subprocesos, el proceso principal queda a la espera de nuevas conexiones, cuando se recibe una se espera a que uno de los subprocesos se encuentre disponible para atender la solicitud, posteriormente, el cliente es asignado, se despierta el subproceso, y se incrementa el contador de solicitudes. Para determinar si se encuentran subprocesos disponibles, se utilizó un semáforo, el cual es incrementado por los subprocesos cuando finalizan la atención de un cliente. Para realizar la transferencia del cliente del proceso padre al subproceso, se utilizaron sockets de unix (*unix domain sockets*), los cuales permiten transferir el archivo descriptor del cliente para que el subproceso pueda atender la solicitud. Por cada subproceso se tiene un semáforo asignado, de esta forma se despiertan en el momento que deben atender una solicitud.

En la figura 4 se puede observar el diagrama de flujo de

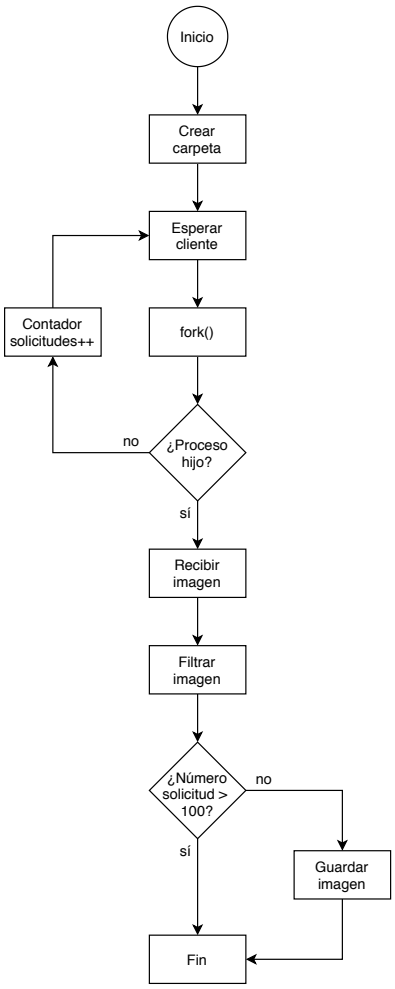

Fig. 2: Diagrama de flujo del servidor heavy process

los subprocesos del servidor pre heavy process. Cuando estos son creados, esperan por la señal de atender una solicitud, la cual es generada por el proceso padre a través de un semáforo. Una vez esta señal es captada, se procede a recibir el archivo descriptor del cliente, y se empieza con la recepción de la imagen. Después de filtrar la imagen recibida, se determina si se debe almacenar en disco, y se avisa al proceso principal que la atención de la solicitud ha finalizado, a través de un semáforo.

En la figura 5 se muestra el diagrama de secuencia de la interacción entre el cliente y el servidor. El usuario inicia el programa del cliente, ingresando como argumentos la ip y puerto del servidor, la ruta de la imagen que desea filtrar, y la cantidad de hilos y ciclos que se deben ejecutar. El cliente establece conexión con el servidor, y envía la cantidad de solicitudes totales que serán realizadas, el servidor empieza a tomar el tiempo en el momento en el que se recibe este mensaje, y termina la medición cuando se alcanza la cantidad de solicitudes indicada por el cliente, el tiempo obtenido es guardado en un archivo de texto junto con la cantidad de solicitudes atendidas. El cliente abre la imagen indicada por el usuario, y procede a crear los hilos encargados de realizar el envío. Por cada envío de una imagen, se establece una nueva

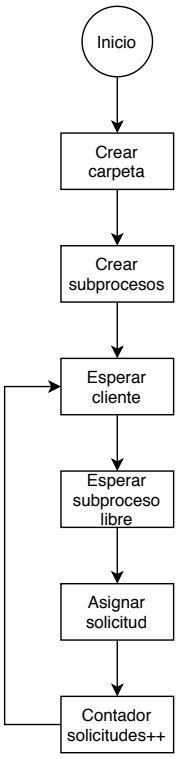

Fig. 3: Diagrama de flujo del proceso principal del servidor pre heavy process

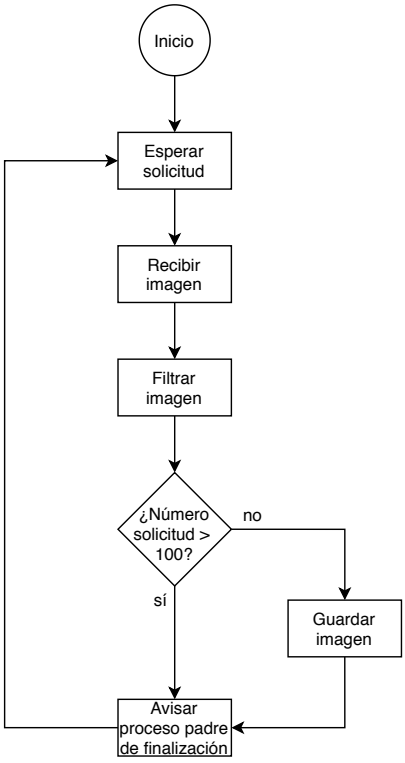

Fig. 4: Diagrama de flujo de los subprocesos del servidor pre heavy process

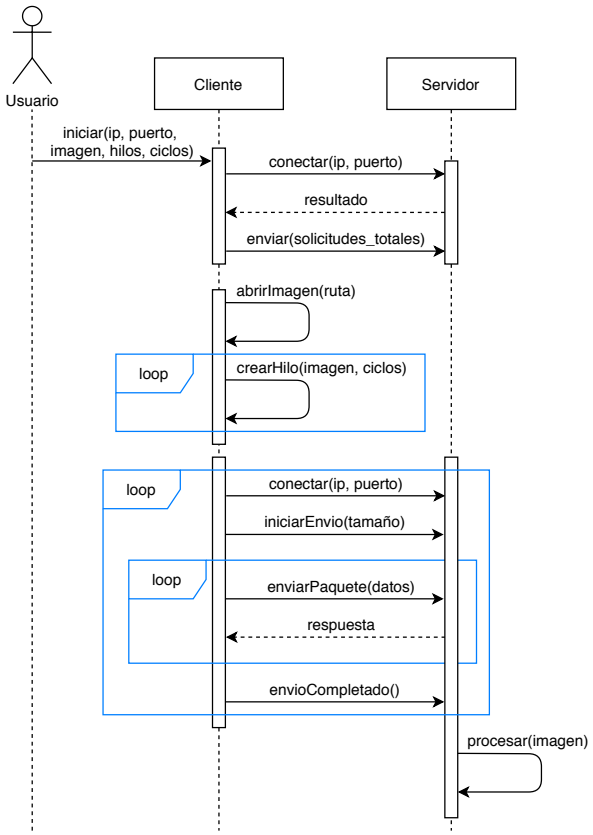

Fig. 5: Diagrama de secuencia del sistema

conexión con el servidor, y se envía un mensaje indicando el tamaño del archivo, posteriormente, se envía la imagen por partes, hasta completar el envío. El servidor entonces realiza el procesamiento de la imagen.

En la figura 6 se aprecia el diagrama de arquitectura del sistema. Se puede observar que el cliente no sabe con qué tipo de servidor está interactuando, si no que para él todos tienen una misma interfaz. Sin embargo, se tienen tres tipos de servidores diferentes, pero todos utilizan el mismo Request Handler, el cual se encarga de atender la solicitudes de los clientes. Este utiliza el Receiver Handler para manejar la recepción de la imagen, el Sobel Filter para realizar el procesamiento, y el Image Handler para realizar el guardado de la imagen en un archivo png. El visualizador utiliza los archivos de texto generados por los servidores para realizar las gráficas con las estadísticas de las consultas.

El filtro Sobel fue implementado manualmente en C, por lo que se requirió programar el proceso de convolución entre dos matrices, el resultado fue escrito en un archivo png utilizando la biblioteca libpng.

IV. Instrucciones de utilización

A continuación se detallarán las instrucciones de cómo utilizar el proyecto:

A. *Compilación*

Las instrucciones para la compilación del programa se detallan a continuación:

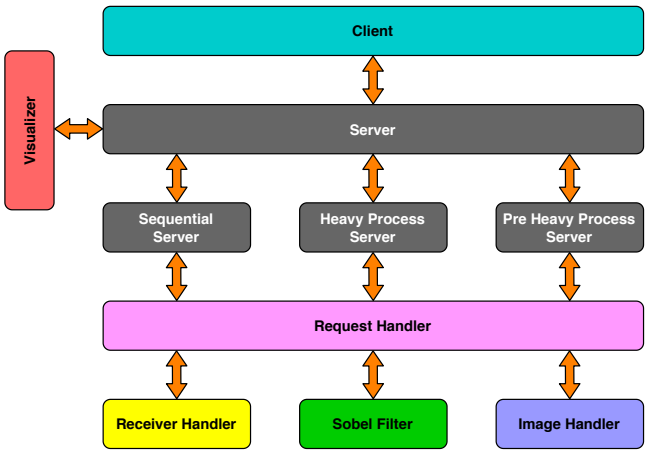

Fig. 6: Diagrama de arquitectura del sistema

- Descargar el código fuente como zip - Git: https://github.com/ArturoCordoba/P2-Operativos-IIS2020.git
- Descomprimir el archivo descargado.
- Ingresar a la carpeta `P2-Operativos-IIS2020` donde se encuentra el código descomprimido.
- Abrir una terminal en el directorio mencionado en el punto anterior y ejecutar el comando `> make`
- Si la compilación se realiza correctamente, se crearán 4 archivos, uno para el cliente (`client`), el servidor secuencial (`server_seq`), servidor heavy process (`server_hp`) y servidor pre heavy process (`server_php`).

B. Uso del programa

Una vez que la compilación se realiza de manera correcta, para ejecutar el proyecto debe realizar los siguientes pasos:

Cliente
$./client<ip><puerto><imagen><Nthreads><Nciclos>$

- **ip:** Corresponde a la dirección ip del servidor al cual se va a conectar.
- **puerto:** Corresponde al numero de puerto por el cual se va a comunicar con el servidor.
- **imagen:** Corresponde a la dirección de la imagen a procesar.
- **Nthreads:** Corresponde a la cantidad de hilos a crear.
- **Nciclos:** Corresponde a la cantidad de ciclos que se va a ejecutar en cada hilo.

Servidor secuencial
$./server\_seq$

- - Sin Argumentos -

Servidor Heavy Process
$./server\_hp$

- - Sin Argumentos -

Servidor Pre Heavy Process
$./server\_php<procesos>$

- **procesos:** Corresponde a la cantidad de procesos a crear.

Visualizador
Para utilizar el visualizador se debe tener instalado GNU Octave, abrir el archivo visualizer.m y ejecutarlo. Antes de utilizar el programa, es necesario que se ejecute al menos una vez cada servidor.

V. Bitácora de actividades

En la Tabla I se muestra la lista de actividades realizadas para el desarrollo del proyecto, junto al integrante que realizó la tareas y la cantidad de horas dedicadas. (A: Arturo ,F: Fabián, J: José)

Fecha	Integrante(s)	Actividad	Tiempo
05/11/20	A, F y J	Lectura y distribución del proyecto	2 horas
08/11/20	A y J	Implementación del Cliente	2 horas
09/11/20	A y J	Investigación e Implementación del filtro Sobel	3 horas
12/11/20	F	Implementación del Servidor Secuencial	3 horas
13/11/20	A y F	Integración y mejoras entre Cliente y Servidor S.	2 horas
16/11/20	J	Implementación del Servidor Heavy Process:	4 horas
18/11/20	A y J	Integración y mejoras entre Cliente y Servidor H. P.	3 horas
19/11/20	A	Implementación del Servidor Pre Heavy Process:	4 horas
20/11/20	A y J	Integración y mejoras entre Cliente y Servidor P. H. P.	3 horas
21/11/20	A	Estadísticas	3 horas
24/11/20	A, F y J	Documentación	4 horas
		Total	**33 horas**

TABLE I: Bitácora de actividades realizadas.

VI. Resultados

En esta sección se mostrarán los datos obtenidos para los tres servidores. Se realizaron 100, 225, 400, y 625 consultas en cada ejecución. El servidor pre heavy process tuvo disponible un total de 10 subprocesos para atender las solicitudes.

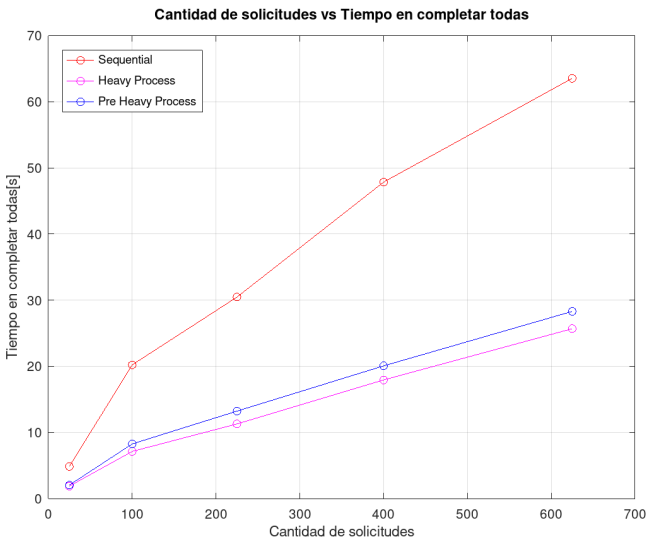

Fig. 7: Cantidad de solicitudes vs Tiempo en ejecutar todas

En la figura 7 se puede observar el tiempo que tardaron los servidores en ejecutar todas las solicitudes enviadas por el cliente. Es claro que el servidor secuencial es el debe durar más, ya que solo atiende una solicitud a la vez. El servidor heavy process tiene un mejor rendimiento que el servidor pre heavy process, esto se debe a que el primero puede crear una cantidad indefinida de procesos para atender cada solicitud en el momento que se recibe, mientras que el servidor pre heavy process debe esperar a que alguno de sus procesos hijos se encuentre disponible, la cantidad de procesos hijos se encuentra limitada a un número establecido, lo que el rendimiento es un poco menor.

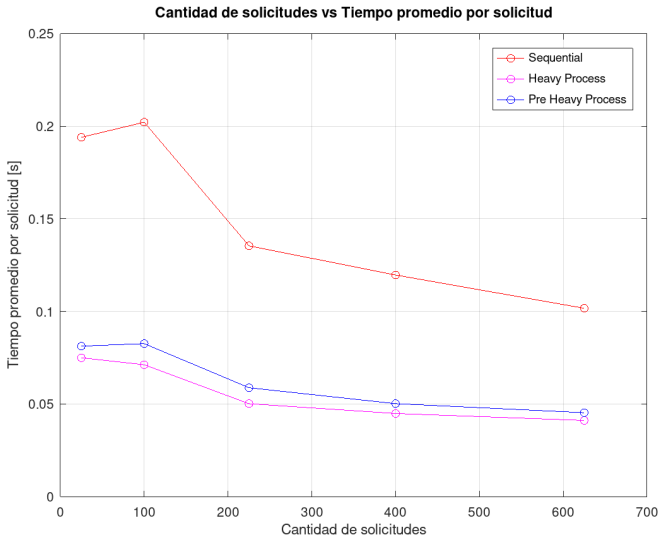

Fig. 8: Cantidad de solicitudes vs Tiempo promedio por solicitud

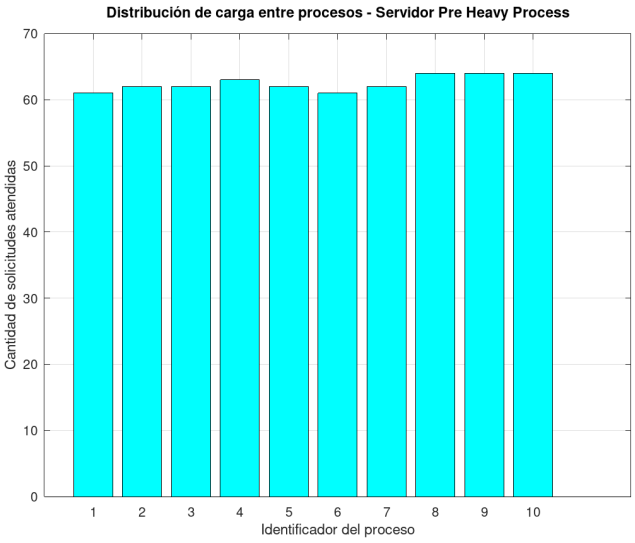

Fig. 9: Cantidad de solicitudes vs Tiempo promedio por solicitud

En la figura 8 se muestra el tiempo promedio por solicitud para cada servidor. Se puede observar que el rendimiento de los tres servidores se comporta de manera similar a los datos mostrados en la figura 7, con el servidor secuencial como el más lento y el servidor heavy process como el más rápido. Las primeras 100 solicitudes son las que tienen un mayor promedio, esto se debe a que estas deben escribir la imagen en disco perjudicando el rendimiento.

En la figura 9 se muestra la distribución de carga que tuvieron los subprocesos del servidor pre heavy process para la ejecución con un total de 625 solicitudes. Es claro que la carga se distribuyó de manera uniforme entre los 10 subprocesos disponibles.

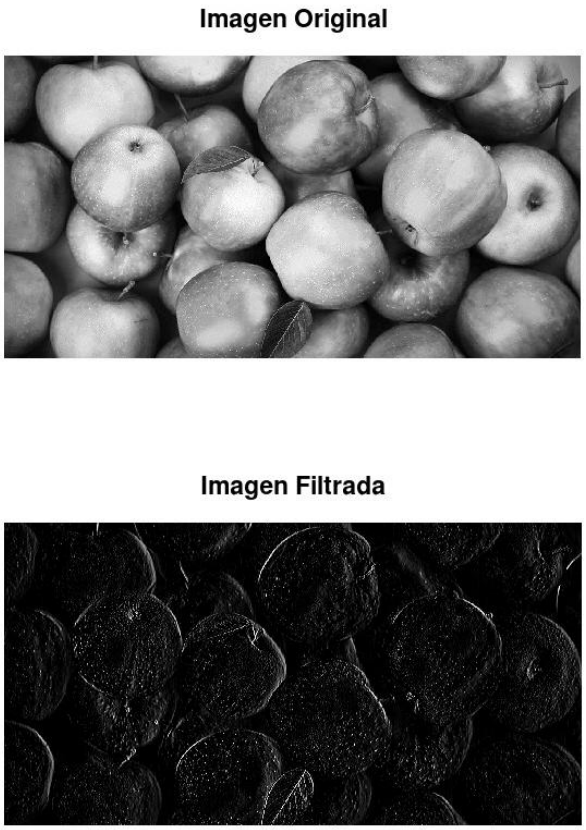

Fig. 10: Resultado obtenido al aplicar el Filtro Sobel

En la figura 10 se puede observar el resultado obtenido al aplicarle el filtro a una imagen de unas manzanas. El Filtro Sobel implementado se encarga de detectar los bordes horizontales.

VII. Conclusiones

La paralelización de las tareas de un proceso mejora drásticamente el rendimiento de un sistema, es evidente que el servidor secuencial tardó muchísimo más en procesar todas las solicitudes en comparación a los otros dos servidores, con un mayor tiempo promedio por solicitud en todos los casos. Aprovechar de manera correcta los recursos que ofrece un sistema es una característica que debe ser explorada por un Ingeniero en Computadores, con el objetivo de realizar una solución óptima a un problema planteado.

El servidor heavy process tuvo un mejor rendimiento que su contraparte pre heavy process, sin embargo, crear subprocesos a la libre puede ocasionar que un sistema se sature, provocando errores que pueden interrumpir el servicio del sistema. El enfoque utilizado por el servidor pre heavy process es mucho más atractivo, porque se mantiene un control de los recursos utilizados obtienendo además un rendimiento muy bueno, incluso cercano al obtenido con el servidor heavy process.

REFERENCES

[1] J. A. Castillo (2020, May 10). Procesador en paralelo: que es y para qué sirve. [Online] Disponible en: https://www.profesionalreview.com/2020/05/10/procesador-en-paralelo/

[2] R. Macias, J. A. Rodríguez,J. N. Alba H. Taud (2016, Jul 1). EXTRACCIÓN DE BORDES; OPERADORES SOBEL, PREWITT Y ROBERTS [Online] Disponible en: http://www.boletin.upiita.ipn.mx/index.php/ciencia/669-cyt-numero-55/1293-extraccion-de-bordes-operadores-sobel-prewitt-y-roberts