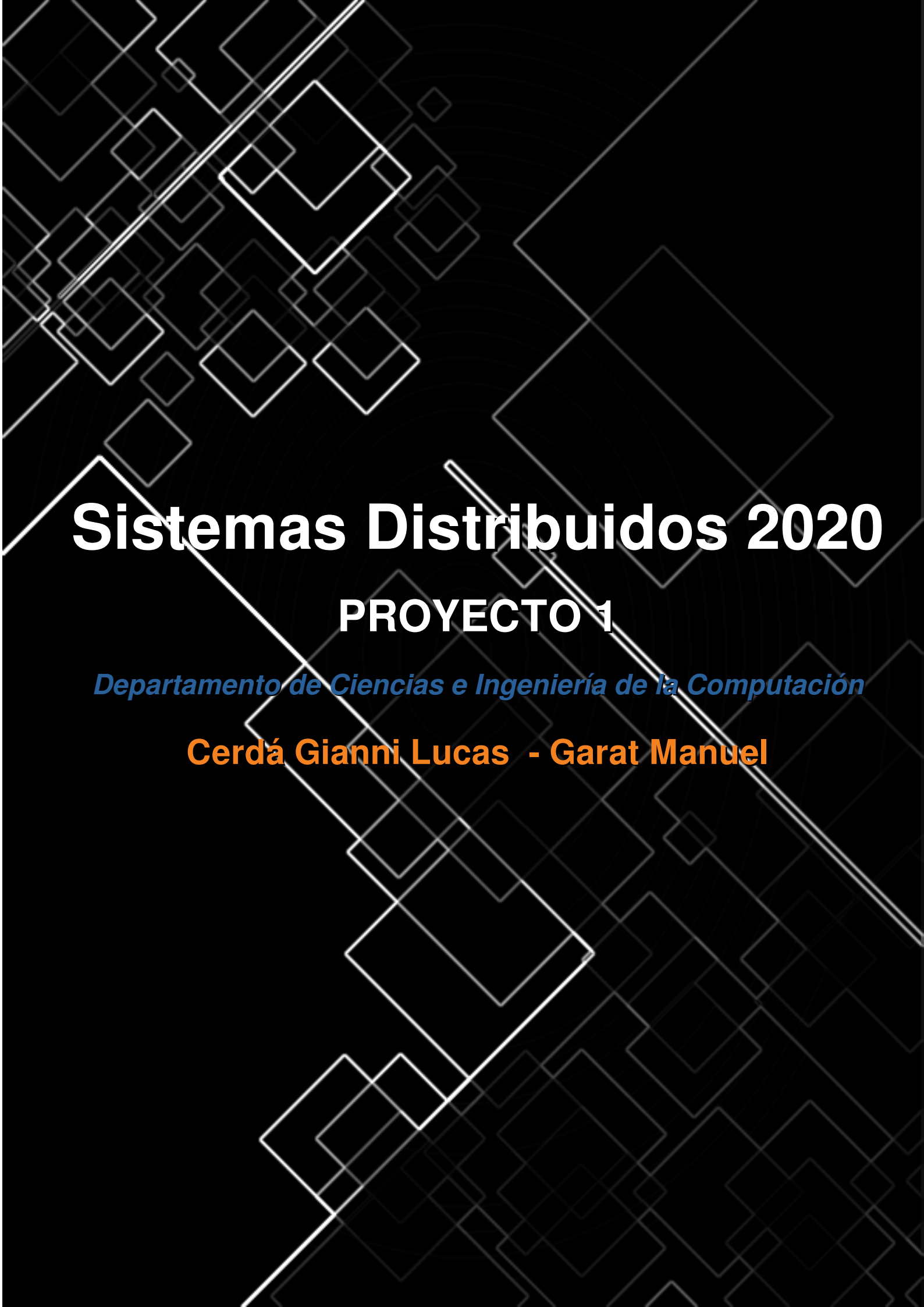


Sistemas Distribuidos 2020

PROYECTO 1

Departamento de Ciencias e Ingeniería de la Computación

Cerdá Gianni Lucas - Garat Manuel

Aclaración: *Los experimentos realizados en este proyecto se hicieron con distintas distribuciones, incluida la provista por la cátedra:*

- *Linux Mint 19.3 Tricia x86_64*
  Arquitectura: *x86_64 (32-bit, 64-bit)*
  CPU(s): *4*
  Memoria: *16 GiB*

- *Arch Linux x86_64*
  Arquitectura: *x86_64 (32-bit, 64-bit)*
  CPU(s): *4*
  Memoria: *8 GiB*

- *Linux Mint 19.3 Tricia i386 (máquina virtual)*
  Arquitectura: *i386 (32-bit)*
  CPU(s): *1*
  Memoria: *2 GiB*

- *Raspbian 10 (Buster) x86_64 (máquina virtual)*
  Arquitectura: *x86_64 (32-bit, 64-bit)*
  CPU(s): *1*
  Memoria: *1 GiB*

Ejercicio 1

Realizar un experimento en el cual se obtenga el tiempo que tarda en crear un proceso.

Ubicación:

Proyecto1 > Fuentes > 1

Fuentes:

Makefile
medir_tiempo.c

Compilación:

make

Ejecución:

./ejecutable

- ***Experimentos realizados***

Sistema Operativo	Linux Mint	Raspbian (máquina virtual)	Arch Linux
Promedio tiempo de creación de procesos (5 mediciones)	**150 µs**	**172.8 µs**	**116.4 µs**

- ***Resultados***

Los resultados obtenidos indican que el tiempo de ejecución promedio para la creación de procesos en Linux Mint y Arch es menor en comparación al Raspbian provista por la cátedra. Estas diferencias pueden deberse a que las características propias de cada distribución varían en su arquitectura y rendimiento (procesador, memoria, etc). También los distintos tiempos tomados pueden deberse a que hay procesos que se están ejecutando en background y las aplicaciones activas en el momento de capturar los tiempos de creación de procesos en cada una de ellas.

- ***Respuestas: ¿Cuántos procesos se pueden crear?***

La cantidad de procesos que un usuario puede crear depende de:

1. Restricciones o privilegios, que pueden modificarse y leerse en el siguiente archivo:

   `/etc/security/limits.conf`

   Ese archivo, en Linux Mint, contiene la siguiente información:

```
#<domain>      <type>  <item>         <value>
#

#*               soft    core            0
#root            hard    core            100000
#*               hard    rss             10000
#@student        hard    nproc           20
#@faculty        soft    nproc           20
#@faculty        hard    nproc           50
#ftp             hard    nproc           0
#ftp             -       chroot          /ftp
#@student        -       maxlogins       4
```

   El <domain> se refiere a un usuario, grupo o todos los usuarios, se puede usar un nombre de grupo si lo antepone con un @. Si usa * sería el predeterminado para todos los usuarios

   El <tipo> especifica el tipo límite (suave, hard) donde el tipo soft es un limite de advertencia mientras que el limite hard es un límite real. El núcleo mantiene los límites estrictos mientras que el shell impone los límites flexibles.

   Las distintas restricciones nos dan la oportunidad de utilizarlas como plantillas para imponer nuestros propios limites.

2. Por otro lado, otra manera de ver la cantidad de procesos que se pueden crear es mediante viendo el contenido del archivo:

   `/proc/sys/kernel/pid_max`

   En este se encuentra el la máxima cantidad de procesos que se pueden crear en el sistema. Tanto en Linux Mint como en Raspbian el límite es 32768 procesos. En Arch Linux es 4194304.

Linux no tiene un límite de hilos por proceso (dado que los hilos procesan en un espacio de direcciones compartido). El archivo

```
/proc/sys/kernel/threads-max
```

contiene la cantidad máxima de hilos en ejecución.

Ejercicio 2

Realizar un experimento para medir el tiempo que tarda en crear un proceso hijo que cargue una imagen ejecutable con una tarea. La tarea debe realizar un ciclo de N iteraciones y en cada iteración dos operaciones aritméticas y para finalizar guardar un texto en un archivo.

a) El proceso hijo es creado en la misma máquina.

- **Fork misma máquina**

Ubicación:

Proyecto1 > Fuentes > 2 > fork > local

Fuentes:

Makefile
tarea.c
padre.c

Compilación:

make

Ejecución:

./padre *<cantidad de iteraciones>*

Experimentos realizados

Sistema Operativo	Linux Mint	Raspbian (máquina virtual)	Arch Linux
Promedio tiempo de ejecución (5 mediciones con 10000 iteraciones)	231 µs	337.2 µs	681.4 µs

- **System misma máquina**

Ubicación:

Proyecto1 > Fuentes > 2 > system > local

Fuentes:

Makefile
tarea.c

padre.c

Compilación:

make

Ejecución:

./padre *<cantidad de iteraciones>*

Experimentos realizados (system)

Sistema Operativo	Linux Mint	Raspbian (máquina virtual)	Arch Linux
Promedio tiempo de ejecución (5 mediciones con 10000 iteraciones)	**4289 µs**	**3403.4 µs**	**8216.4 µs**

b) El proceso hijo es creado en una máquina remota.

- **Fork máquina remota**

Ubicación:

Proyecto1 > Fuentes > 2 > fork > remoto

Fuentes:

Makefile
proceso.x
proceso_client.c
proceso_server.c
tarea.c

Compilación:

make

Ejecución (cliente)

./client *<IP servidor> <cantidad de iteraciones>*

Ejecución (servidor)

./server

Experimentos realizados

Sistema Operativo	Linux Mint	Raspbian (máquina virtual)	Arch Linux
Promedio tiempo de ejecución (5 mediciones con 10000 iteraciones)	**389 µs**	**3078 µs**	**899.2 µs**

- ## System máquina remota

Ubicación:

Proyecto1 > Fuentes > 2 > system > remoto

Fuentes:

Makefile
proceso.x
proceso_client.c
proceso_server.c
tarea.c

Compilación:

make

Ejecución (cliente)

./client *<IP servidor> <cantidad de iteraciones>*

Ejecución (servidor)

./server

Experimentos realizados (system)

Sistema Operativo	Linux Mint	Raspbian (máquina virtual)	Arch Linux
Promedio tiempo de ejecución (5 mediciones con 10000 iteraciones)	**4355.2 µs**	**8690.8 µs**	**4936 µs**

Comportamiento:

Se puede observar que el tiempo de ejecución aumenta cuando se utilizan máquinas remotas. Esto es causado especialmente por las latencias propias del envío y recepción de mensajes en la red. Por otra parte, es evidente que el tiempo de ejecución crece aún más con la llamada a *system*. La principal razón de esta diferencia es que *system* ejecuta una shell sobre la que se crea un proceso en donde se carga una imagen ejecutable. Es decir, se agrega un paso más en la ejecución del proceso correspondiente a la creación de la shell.

Ejercicio 3

Se deben realizar los siguientes experimentos para comparar el tiempo de ejecución de procedimientos que se ejecutan en forma remota (RPC - Remote Procedure Call) con respecto a los procedimientos que se ejecutan en forma local (LC - Local Call). La llamada al procedimiento se realiza con argumento de entrada de 2048 bytes y un argumento de respuesta de 2048 bytes.
Considere que el procedimiento obtiene la suma de dos arreglos de 100 lugares y asigna valores a los argumentos de respuesta.

Consideraciones que se tuvieron en cuenta para la implementación:

Para este experimento, se definió una estructura con un arreglo de caracteres de 2048 lugares (el tamaño del tipo char en C ocupa un byte en memoria). El procedimiento recibe una de estas estructuras y retorna otra. El primer arreglo de cien elementos son las primeras cien posiciones de la estructura de entrada [0, ... 99], mientras que el segundo arreglo son las segundas cien posiciones de la estructura de entrada [100, ... 199]. Así, se suman ambos arreglos y se asigna el resultado a las primeras cien posiciones de la estructura de salida.

- **LC - Local Call**

Ubicación:
Proyecto1 > Fuentes > 3 > local

Fuentes:
Makefile
suma_arreglos.c

Compilación:
make

Ejecución:
./ejecutable

- **RPC – Remote Procedure Call**

Ubicación:
Proyecto1 > Fuentes > 3 > remoto

Fuentes:
Makefile
suma_arreglos.x
suma_arreglos_client.c
suma_arreglos_server.c

Compilación:
make

Ejecución (cliente)

./client *<IP servidor>*

Ejecución (servidor)

./server

a) El procedimiento remoto se encuentra en la misma máquina.

Experimentos realizados

Sistema Operativo	Raspbian (máquina virtual)
Promedio tiempo de ejecución (5 mediciones)	491.6 µs

b) El procedimiento remoto se encuentra en otra máquina con la misma arquitectura y distribución.

Experimentos realizados

Sistema Operativo	Raspbian (máquina virtual)
Promedio tiempo de ejecución (5 mediciones)	1264.8 µs

c) El procedimiento remoto se encuentra en otra máquina con diferente arquitectura y distribución.

Experimentos realizados

Sistema Operativo	Raspbian (máquina virtual)
Promedio tiempo de ejecución (5 mediciones)	1736.8 µs

Respuestas

En los resultados obtenidos anteriormente se puede ver de forma clara como los tiempos de ejecución promedio aumentan en gran medida si vamos de una llamada a llamada de procedimiento remote, incrementándose aún más si se trata de clientes y servidores con diferentes arquitecturas y distribuciones.

Ejercicio 4
Realizar un experimento para medir cuánto tiempo se requiere para atender una solicitud de cliente a un servidor utilizando comunicación mediante sockets con TCP.

Cuánto requiere si el tamaño del dato que requiere el cliente es de 4 bytes.

Ubicación:

Proyecto1 > Fuentes > 4 > 4bytes

Fuentes:

Makefile
client.c
server.c

Compilación:

make

Ejecución (cliente)

./client *<IP servidor>*

Ejecución (servidor)

./server

Experimentos realizados

Sistema Operativo	Raspbian (máquina virtual)
Promedio tiempo de ejecución (5 mediciones)	95.6 µs

Cuánto requiere si el tamaño del dato que requiere el cliente es de 2048 bytes.

Ubicación:

Proyecto1 > Fuentes > 4 > 2048bytes

Fuentes:

Makefile
client.c
server.c

Compilación:

make

Ejecución (cliente)

./client *<IP servidor>*

Ejecución (servidor)

./server

Experimentos realizados

Sistema Operativo	Raspbian (máquina virtual)
Promedio tiempo de ejecución (5 mediciones)	144 µs

Respuestas

Se observa en los resultados anteriores como los tiempos de ejecución promedio aumentan al incrementarse la cantidad de datos a transmitirse al cliente.

Ejercicio 5

Realizar un servidor "Mini Operaciones".
El cliente solicita el servicio aritmético que desea y luego invoca al procedimiento remoto para que se lo brinde.
El proceso servidor tiene las siguientes funciones: conversión de un número decimal a binario, conversión de un número binario a hexadecimal, suma, resta, multiplicación y división. Las operaciones de suma y resta pueden tener hasta 4 operandos y el resto de las operaciones tienen 2 operandos.

a) Utilizar la llamada de un procedimiento que se encuentre en otro proceso en la misma máquina. Utilizando comunicación entre procesos (memoria compartida, archivos, colas de mensajes, etc.).

Ubicación:

Proyecto1 > Fuentes > 5 > ipc

Fuentes:

Makefile
mini_operaciones.c

Compilación:

make

Ejecución

mini_operaciones

b) Utilizar llamadas a procedimientos remotos para las funciones requeridas (RPC).

Ubicación:

Proyecto1 > Fuentes > 5 > rpc

Fuentes:

Makefile
mini_operaciones.x
mini_operaciones_client.c
mini_operaciones_server.c

Compilación:

make

Ejecución (cliente)

./client *<IP servidor>*

Ejecución (servidor)

./server

Sistema Operativo	Suma	Resta	Multiplicación	División	Decimal a binario	Binario a hexadecimal
IPC	46.4 µs	53.8 µs	33.6 µs	42.2 µs	33.6 µs	35.2 µs
RPC a sí misma (Raspbian)	237 µs	250.2 µs	253.6 µs	277.8 µs	213 µs	261.4 µs
RPC a otra máquina virtual (Rapsbian)	505.2 µs	519.8 µs	483.4 µs	504.8 µs	628 µs	493.4 µs
RPC a otra máquina virtual (Linux Mint 32 bits)	706.8 µs	832.2 µs	550.2 µs	708.8 µs	685.4 µs	628.8 µs

Ejercicio 6
Comunicación de Agencias.

- **Cada agencia quiere comunicarse con otra agencia para intercambiar servicios y/o información.**
- **Considere para esta primera solución que la comunicación está formada por dos agencias pares, la agencia A y la agencia B. La agencia A brinda turnos para sacar la licencia de matrimonio, información sobre la partida de nacimiento y turno para la inscripción de un bebé recién nacido. La agencia B brinda turnos para patentar el auto, turnos para la transferencia de un vehículo e información sobre el dominio de un vehículo.**

a) Realizar los procesos necesarios para cumplir las tareas especificadas.

b) Explique las consideraciones de diseño que utilizaron para modelar el problema, detallando especialmente las consideraciones para iniciar y finalizar la comunicación.

Consideraciones que se tuvieron en cuenta para la implementación:

Para este experimento, cada agencia se divide en dos partes o módulos: una que brinda servicios a la otra (servidora) y otra que pide sus servicios (cliente). Se pensó en la creación de un único módulo con dos procesos, uno servidor y otro cliente; sin embargo, se descartó esta idea para evitar que la salida por pantalla de cada proceso se mezcle, resultando en una visión confusa de lo que está sucediendo.

También se definió un archivo con constantes comunes para ambas agencias y una interfaz (.h) para las operaciones de cada agencia, que son llamadas desde los módulos servidores mencionados en el párrafo anterior, según el pedido del cliente. De esta manera, los módulos de comunicación se abstraen de la implementación de las operaciones de cada agencia.

Las operaciones de cada agencia se implementaron de una manera sencilla y demostrativa. El foco de la implementación se puso en la comunicación por sockets, no tanto en las operaciones que realiza cada agencia per se.

La comunicación por sockets es orientada a conexión y se realiza de la siguiente forma:

1. El cliente establece una conexión con el servidor, a partir de un **listen()**, **accept()** para el servidor, y un **connect()** para el cliente.
2. El servidor crea un proceso que maneje esta conexión.
3. El cliente envía un mensaje con el código de la operación que pide.
4. El proceso manejador recibe este mensaje, ejecuta la operación pedida y responde con otro mensaje que contenga el resultado.
5. Se repiten los pasos 3 y 4 hasta que el cliente envíe una operación de terminación de conexión.

Ubicación:

Proyecto1 > Fuentes > 6

Fuentes:

clienteA.c
clienteB.c
constantes.h
Makefile
operacionesA.c
operacionesA.h
operacionesB.c
operacionesB.h
servidorA.c
servidorB.c

Compilación:

make

Ejecución (Agencia A)

- ***Ejecución (cliente), que pide servicios a la agencia B***

  ./clienteA *<IP servidor>*

- ***Ejecución (servidor), que ofrece servicios***

  ./servidorA

Ejecución (Agencia B)

- ***Ejecución (cliente), que pide servicios a la agencia A***

  ./clienteB *<IP servidor>*

- ***Ejecución (servidor), que ofrece servicios***

  ./servidorB

Ejercicio 7

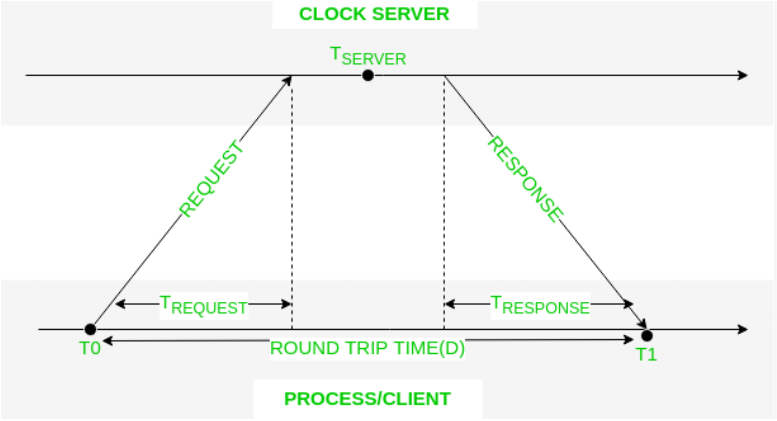
El proceso cliente quiere conocer la variación que tiene el reloj de su máquina con respecto a la máquina del proceso servidor. Inicialmente considere que sólo quiere conocer la variación con respecto a una máquina y luego con respecto a varias máquinas. Realice los procesos necesarios para resolver este problema.

Consideraciones que se tuvieron en cuenta para la implementación:

Para este experimento, se utiliza un algoritmo basado en el de Cristian para conocer la variación que tiene el reloj del cliente con respecto al del servidor. Esto es:

1. El cliente envía un mensaje al servidor: T0.
2. El servidor responde con su hora local: Ts.
3. La respuesta llega al cliente: T1.
4. El cliente calcula el RTT: (T1-T0)/2 = Tr

5. La diferencia de tiempo es: Ts - T0+Tr. Si la diferencia es negativa, el cliente está adelantado. Sino, está atrasado.



Algoritmo de Cristian

Para consultar con muchas máquinas, el cliente realiza una consulta igual a la anterior por cada una de las máquinas que se especifican al ejecutarlo. Luego, realiza un promedio de cada una las mediciones.

- ***Una máquina***

Ubicación:
Proyecto1 > Fuentes > 7 > una-máquina

Fuentes:
Makefile
offset.x
offset_client.c
offset_server.c

Compilación:
make

- ***Ejecución (cliente)***
./client *<IP servidor>*

- ***Ejecución (servidor)***
./server

- ***N máquinas***

Ubicación: Proyecto1 > Fuentes > 7 > n-máquinas

Fuentes:
Makefile
offset.x
offset_client.c
offset_server.c

Compilación:
make

- ***Ejecución (cliente)***
./client

- ***Ejecución (servidor)***
./server

Ejercicio 8
Explique el concepto de Blockchain en una carilla máximo.

Blockchain es un sistema distribuido cuyo objetivo es proporcionar una plataforma confiable y segura para el manejo de transacciones. El concepto de blockchain fue puesto por primera vez en práctica en la criptomoneda Bitcoin. De hecho, la confiabilidad y confidencialidad que posee esta moneda, se debe a que utiliza el sistema de blockchain para sus transacciones.

Como su nombre lo indica, blockchain consiste en una cadena de bloques donde cada bloque es un conjunto de transacciones y a su vez estos bloques se enlazan con otros bloques en un orden único e inalterable. Debido a este orden único, toda transacción presente en un bloque queda completamente validada y no hay forma de que se pueda falsificar por razones que se explicarán más adelante.

Cada bloque del sistema se conforma principalmente de tres componentes: Datos (en la mayoría de los casos transacciones), un Hash del bloque y un Hash del bloque anterior. El hash del bloque se forma a partir de la los datos que contiene en su interior. Es por ello que, si alguien intenta modificar alguna transacción, el hash del bloque cambiaría completamente. Si esto ocurriera, el bloque siguiente en la cadena tendría un hash incorrecto de su bloque anterior, por lo que el bloque modificado quedaría invalidado. Esta medida de seguridad podría sortearse con facilidad si se re-calculan todos los hash de los bloques en base al primer bloque modificado, pero esto no es suficiente gracias a la principal característica de blockchain: se trata de un Sistema Distribuido peer-to-peer. Esto significa que el sistema está conformado de miles de usuarios que tienen una copia exacta de la cadena de bloques y todos en conjunto son los que validan mediante la resolución de un problema criptográfico. Un bloque se da por válido si más del 50% de los usuarios lo confirman. Esto quiere decir que si alguien logra cambiar la cadena de bloques alterando todos los hash, aun así, no llegaría a validar el bloque debido a que tiene que ser validado por la mayoría de los usuarios. Esto significa que mientras más
usuarios tengan el sistema, más confiable y seguro es.

a) Se podría decir que el principal beneficio de blockchain es que es una forma completamente confiable de hacer transacciones importantes (como transferencias de dinero/bitcoins) sin la necesidad de un intermediario. La transacción es completamente directa y miles de usuarios verifican que dicha transacción es válida sin saber la identificación de las personas que están realizado dicha transacción

b) Algunas limitaciones tecnológicas que posee blockchain son:

- Escalabilidad limitada debido al “sistema de consenso” el cual restringe la cantidad de transacciones procesadas por segundo.

- Aunque la identidad del usuario es privada, las transacciones son públicas y pueden ser fácilmente rastreadas. Elevado consumo de energía producto del procesamiento de miles de nodos durante el proceso de validación (Bitcoin tiene un consumo de energía anual de 29.05 TW/h, representando el 0,13% del consumo energético mundial).

Ejercicio 10

En este ejercicio, usted deberá elegir entre el problema 5 y 6 e implementarlo en Java con sockets con conexión. Estime el tiempo de ejecución y compare con los resultados obtenidos.

Aclaración: Para este experimento se incorporó un archivo README en el que explica al usuario paso por paso cómo es el proceso de compilación y ejecución del programa por terminal en la distribución de Linux que utilice y que paquetes necesarios debe tener previamente instalado.

Ubicación del archivo: Proyecto1 > Fuentes > 10 > README

- ***Monocliente***

  Ubicación: Proyecto1 > Fuentes > 10 > 1 > MonoCliente

  Fuentes:

  Server.java
  Cliente.java
  Servicios.java
  Servicios2.java

  Compilación:

  make (Este archivo se encuentra en *Proyecto1 > Fuentes > 10 > 1 y debe ser ejecutado desde esta ubicación)*

  Ejecución Cliente:

  java MonoCliente.Cliente *<IP> <PORT>*

  Ejecución Servidor:

  java MonoCliente.Server *<PORT>*

  Experimentos Realizados: (Tiempo Promedio en 5 muestras)

Sistema Operativo	Suma
Linux Mint	**1262014.8 ns ≈ 1262.0148 µs**
Raspbian (Máquina Virtual)	**4544408.8 ns ≈ 4544.4088 µs**

Sistema Operativo	Resta
Linux Mint	**1922567.8 ns ≈ 1922.5678 µs**
Raspbian (Máquina Virtual)	**324732 ns ≈ 324.732 µs**

Sistema Operativo	Multiplicación
Linux Mint	315358 ns ≈ 315.358 µs
Raspbian (Máquina Virtual)	1922567.8 ns ≈ 1922.5678 µs

Sistema Operativo	División
Linux Mint	235317.8 ns ≈ 235.3178 µs
Raspbian (Máquina Virtual)	975818.8 ns ≈ 975.8188 µs

Sistema Operativo	Decimal a binario
Linux Mint	249227.8 ns ≈ 249.2278 µs
Raspbian (Máquina Virtual)	414038.2ns ≈ 414.0382µs

Sistema Operativo	Binario a hexadecimal
Linux Mint	2019999.4 ns ≈ 2019.9994 µs
Raspbian (Máquina Virtual)	22729852.4 ns ≈ 22729.8524 µs

Resultados:

Java posee solamente funciones para medir el tiempo en nanosegundos o en milisegundos. En este caso, se utilizó la función que retorna el valor en nanosegundos y, en la tabla, se hizo una aproximación a microsegundos, para respetar el enunciado.
A partir de los tiempos promedios obtenidos en las diferentes distribuciones y arquitecturas de las tablas anteriores se puede deducir que los cálculos que realiza la máquina virtual en promedio es mayor en comparación con los cálculos que realiza Linux Mint como sistema operativo anfitrión, esto se debe nuevamente que las arquitecturas de ambas poseen características diferentes y por lo tanto afecta su rendimiento en la ejecución de los servicios aritméticos.

Extra:

Además de realizar el ejercicio 10 con conexión para un único cliente se desarrollo el ejercicio para múltiples clientes para observar cómo se implementaba ésta modificación.

- ***Multicliente***

  Ubicación: Proyecto1 > Fuentes > 10 > 2 > MultiCliente

Fuentes:

Server.java
Cliente.java
ClientHandler.java
Servicios.java
Servicios2.java

Compilación:

make (Este archivo se encuentra en *Proyecto1 > Fuentes > 10 > 2 y debe ser ejecutado desde esta ubicación)*

Ejecución Cliente:

java MultiCliente.Cliente *<IP> <PORT>*

Ejecución Servidor:

java MultiCliente.Server *<PORT>*