Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC3069 Computación Paralela y Distribuida
Catedrático: Miguel Novella
Ciclo 1 de 2023

Proyecto 2 - Programación Paralela con MPI

Bryann Alfaro 19372
Raul Jimenez 19017
Donaldo Garcia 19683

Link al repo: https://github.com/bryannalfaro/Proyecto2Paralela
Guatemala, 14 de mayo de 2023

ÍNDICE

INTRODUCCIÓN

El poder verificar la vulnerabilidad de un algoritmo criptográfico se hace cada vez más importante en un mundo en donde los datos son la principal fuente de información. El comprender el funcionamiento detrás de ellos puede ayudar a crear mejores algoritmos, que presenten una menor vulnerabilidad.

En este proyecto se utiliza el concepto de memoria distribuida por medio de la librería MPI para realizar un ataque de fuerza bruta, en donde se itera sobre las posibles claves del algoritmo DES (Data Encryption Standard) repartiendo cierta cantidad de claves a cada proceso.

Se busca además, poder mejorar el proceso de fuerza bruta Naive, por medio de la implementación de dos versiones mejoradas que permitan recorrer el espacio de claves de una manera más efectiva.

Para el cifrado y descifrado de la información se utiliza la librería openssl la cual brinda ciertas funciones para poder utilizar distintos modos de encriptación y creación de claves.

ANTECEDENTES

¿Qué es DES?

DES es la abreviación de “Data Encryption Standard”, este es un algoritmo de cifrado simétrico el cual fue desarrollado por IBM en la década de 1970. Este fue de los primeros algoritmos de cifrado utilizados a nivel mundial y fue adoptado por el gobierno de los Estados Unidos como un estándar.

Al ser un algoritmo de cifrado este ha sido parte de muchas investigación y análisis a lo largo de los años. Este ha sido utilizado en varias aplicaciones, enfocadas en la protección de datos confidenciales en sistemas de seguridad informática, bancarios y financieros así como en muchos otros.

DES utiliza un sistema monoalfabético. Este algoritmo de cifrado se basa en varias permutaciones y sustituciones aplicadas sucesivamente al texto en claro que se desea cifrar. El proceso de cifrado comienza con una permutación del bloque de entrada de 64 bits o múltiplos de 64. Luego, el texto permutado es sometido a la acción de dos funciones principales, una función de permutación con entrada de 8 bits y otra de sustitución con entrada de 5 bits. Este proceso de permutación y sustitución se repite 16 veces para obtener el texto cifrado. La clave simétrica utilizada en DES es de 64 bits, de los cuales 56 se usan para la encriptación y los 8 restantes son de paridad y se utilizan para la detección de errores. Debido a que la clave efectiva es de 56 bits, hay un total de

72.057.594.037.927.936 claves posibles, lo que equivale a 72.000 billones de claves. Por lo tanto, la probabilidad de romper el sistema utilizando ataques de fuerza bruta o diccionario es sumamente baja, aunque no imposible si se dispone de una gran potencia de cálculo y suerte. (NEO, s.f.)

Hoy en día como la tecnología ha avanzado mucho este algoritmo ya se usa con gran frecuencia debido a que existen otros que brindan mayor seguridad para enfrentarse a los ataques de hoy en día y proteger la información.

¿Qué es 3-DES?

El algoritmo 3DES es la siguiente versión de DES. Se creó esta modificación también llamada triple DES, la cual consiste en encriptar tres veces consecutivas los datos de entrada del bloqueo con 3 claves de 56 bits distintas. Este utiliza bloques de 64 bits pero cuenta con algunas mejoras que aumentan su seguridad:

Tamaño del bloque	64 bits
Tamaño de la clave	168 bits
Bits de seguridad	112 bits

Aunque este es una gran mejora de seguridad en comparación del sistema DES, este ya no se considera tan seguro, sin embargo este todavía se puede encontrar en algunos sistemas antiguos. Hoy en día los nuevos algoritmos de cifrado han reemplazado a estos debido a que son más efectivos y brindan más seguridad. (Keep Coding, 2023)

DESARROLLO

Parte A

1. Investigue sobre DES y describa los pasos requeridos para cifrar/descifrar un texto. Incluya esto en su reporte.

El algoritmo DES (Data Encryption Standard) es un algoritmo de cifrado por bloques. El cual permite transformar un bloque brindado en texto plano a un texto cifrado, esto por medio de la utilización de una clave única. Este algoritmo usa una clave de tamaño 56

bits, con la cual toma un bloque de texto plano de 64 bits como input y genera un bloque de texto cifrado de 64 bits. (Simplilearn, 2022)

Este proceso toma varios pasos, los cuales son denominados “rondas”. Estas rondas dependerán del tamaño de la key. Ya que, por ejemplo una llave de 128 bits requiere alrededor de 10 rondas mientras que una de 192 bits requiere 12 rondas. (Simplilearn, 2022)

Entre las características que se pueden mencionar del algoritmo se encuentran:

1. El tipo de cifrado que realiza es simétrico, esto refiere que utiliza la misma clave secreta tanto para el proceso de encriptación como desencriptación.
2. Tiene la característica de poder reutilizar la misma clave, en otros algoritmos esto no es posible.
3. En cuanto al tamaño de la llave, estas pueden ser de un tamaño reducido. Sin embargo, en otros algoritmos , la clave debe tener el mismo tamaño del mensaje que se desea transmitir.

(KeepCoding, 2023)

El algoritmo DES presenta distintos modos de operación. Entre ellos se pueden mencionar:

1. **Electronic Codebook (ECB)**. En este modo cada bloque de 64 bit se encripta y desencripta de manera independiente.

2. **Cipher Block Chaining (CBC)**. Cada bloque de 64 bits depende del bloque anterior y utiliza además un vector de inicialización. (IV).

3. **Cipher Feedback (CFB)**. En este modo de operación, el texto que se encuentra cifrado se convierte ahora en la entrada del algoritmo, en donde ahora la salida es de forma pseudoaleatoria, esta salida ahora se le aplica una operación XOR con el texto plano y de esta manera se produce el nuevo mensaje cifrado.

4. **Output Feedback (OFB)**. El proceso es similar a lo que ocurre con CFB solamente que el input del algoritmo es el output del DES anterior.

5. **Counter (CTR).** En este proceso cada bloque del texto o mensaje es operado por medio de XOR con un contador encriptado, durante cada bloque subsecuente el contador se incrementa.

(Simplilearn, 2022)

Pasos de DES para cifrar/descifrar

DES (Estándar de cifrado de datos) es un algoritmo de cifrado de clave simétrica que se utiliza para cifrar y descifrar datos. Fue desarrollado en la década de 1970 por IBM y adoptado por el gobierno de EE. UU. como estándar para la comunicación segura.

El algoritmo DES utiliza una clave de 56 bits para cifrar y descifrar datos en bloques de 64 bits. El algoritmo utiliza una serie de operaciones matemáticas, incluidas la sustitución y la permutación, para transformar el texto sin formato en texto cifrado.

Para cifrar datos usando DES, el texto sin formato se divide primero en bloques de 64 bits. Luego, la clave se usa para realizar una serie de operaciones en cada bloque, lo que da como resultado un bloque correspondiente de texto cifrado. Para descifrar el texto cifrado, se utiliza la misma clave para revertir las operaciones y recuperar el texto sin formato original.

Si bien DES se consideró seguro cuando se desarrolló por primera vez, los avances en la potencia informática lo han hecho vulnerable a los ataques de fuerza bruta. Como resultado, el algoritmo ha sido reemplazado en gran medida por estándares de cifrado más seguros, como AES (Estándar de cifrado avanzado). (Geek For Geeks, 2023)

Cifrado

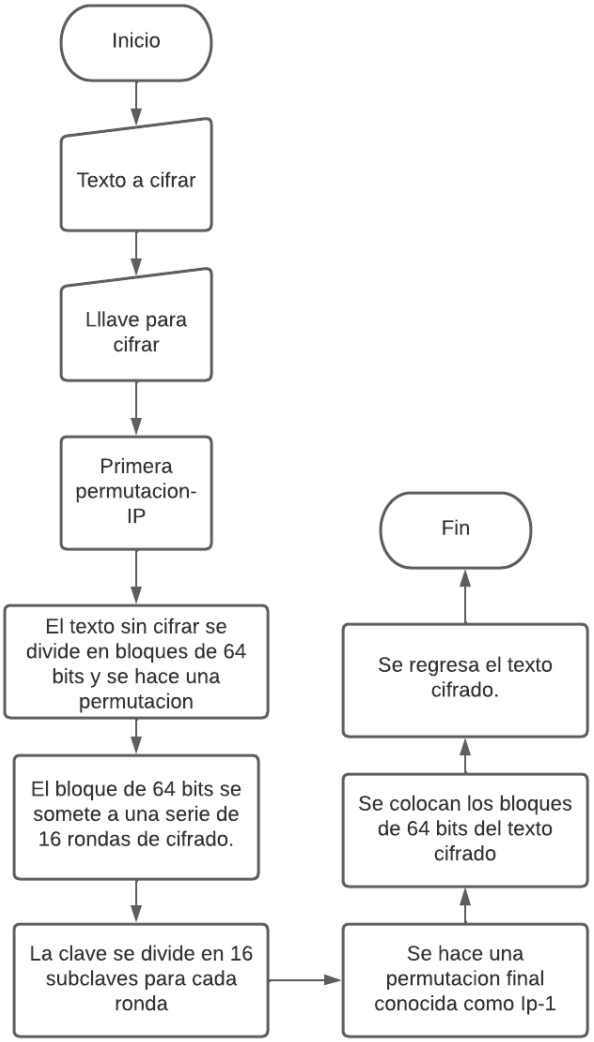
1. Generación de clave: Genere una clave de 56 bits para usarla para el cifrado y descifrado. Esta clave debe mantenerse en secreto.
2. Relleno: si el texto a cifrar no es un múltiplo de 64 bits, debe rellenarse con bits adicionales para que sea un múltiplo de 64 bits.
3. Divide el texto sin formato en bloques: divide el texto sin formato acolchado en bloques de 64 bits.
4. Permutación inicial: aplique una permutación inicial a cada bloque de texto sin formato.
5. Función de ronda: realice una serie de 16 rondas en cada bloque de texto sin formato utilizando la clave de 56 bits. Cada ronda implica una combinación de operaciones de sustitución y permutación.
6. Permutación final: aplique una permutación final a la salida de la última ronda.
7. Concatenación: concatene la salida de cada bloque para formar el texto cifrado final.

Descifrado:

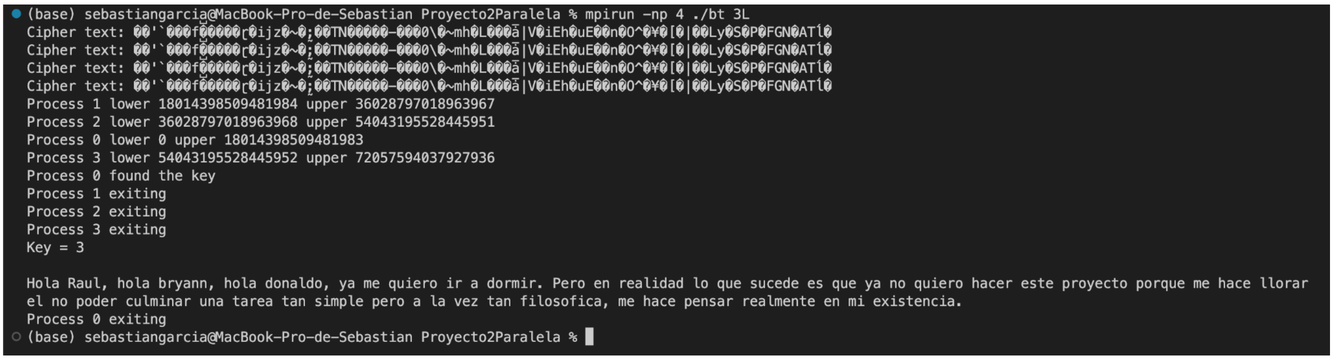
1. Generación de claves: obtenga la misma clave de 56 bits que se utilizó para el cifrado. Esta clave debe mantenerse en secreto.
2. Divida el texto cifrado en bloques: Divida el texto cifrado en bloques de 64 bits.
3. Permutación inicial: aplique una permutación inicial a cada bloque de texto cifrado.

4. Función de ronda: realice una serie de 16 rondas en cada bloque de texto cifrado utilizando la clave de 56 bits. Cada ronda implica una combinación de operaciones de sustitución y permutación que son inversas a las que se utilizan en el cifrado.
5. Permutación final: aplique una permutación final a la salida de la última ronda.
6. Concatenación: concatene la salida de cada bloque para formar el texto sin formato final.
7. Eliminar relleno: si se agregó relleno durante el cifrado, elimínelo del texto sin formato final.

2. Dibuje un diagrama de flujo describiendo el algoritmo DES



3. Luego de trabajar un poco con DES, haremos una prueba de código como introducción y base para el resto del desarrollo. Haga funcionar el programa bruteforce.c Es probable que necesite una biblioteca que reemplace a “rpc/des_crypt.h” en caso su computadora/instalación/sistema operativo lo requiera, para ello puede utilizar cualquier librería que desee/encuentre.



4. Una vez funcionando su programa base, explique, mediante diagramas, texto, dibujos, etc., como funcionan las rutinas (o la equivalente si uso otra librería en caso de decrypt/encrypt):
    a. decrypt (key, *ciph, len)

La función "decrypt" descifra un texto dado una llave utilizando el algoritmo DES (Data Encryption Standard) en modo ECB (Electronic Code Book).

Lo primero que realiza es establecer la paridad de la llave. Para esto se hace por medio de un for en el cual desplaza un bit a la izquierda en cada iteración y luego aplica una máscara para asegurar que el bit más significativo sea cero. Por lo cual el resultado de esta “k” es una nueva llave con paridad par. Esto quiere decir que buscamos que la cantidad de “1” en nuestra llave binaria sea par.

Luego de esto se inicia la clave DES usando la “k” y se establece un paridad impar usando la función “DES_set_odd_parity”.

Una vez se tenga esta llave se procede a analizar el mensaje. Se calcula la longitud del mensaje y se agrega el relleno necesario para que su longitud sea un múltiplo de 8 bytes.

Por último para terminar se descifra el mensaje usando el algoritmo DES en modo ECB mediante la función "DES_ecb_encrypt". Se hace un for para que lo haga en bloques de 8 bytes.

Por último se chequea el padding en la misma.

b. Encrypt (key, *ciph, len)

La función "encryptText" cifra un texto dado una llave utilizando el algoritmo DES (Data Encryption Standard) en modo ECB (Electronic Code Book).

Lo primero que realiza es establecer la paridad de la llave. Para esto se hace por medio de un for en el cual desplaza un bit a la izquierda en cada iteración y luego aplica una máscara para asegurar que el bit más significativo sea cero. Por lo cual el resultado de esta “k” es una nueva llave con paridad par. Esto quiere decir que buscamos que la cantidad de “1” en nuestra llave binaria sea par.

Luego de esto se inicia la clave DES usando la “k” y se establece un paridad impar usando la función “DES_set_odd_parity”.

Una vez se tenga esta llave se procede a analizar el mensaje. Se calcula la longitud del mensaje y se agrega el relleno necesario para que su longitud sea un múltiplo de 8 bytes.

Por último para terminar se cifra el mensaje usando el algoritmo DES en modo ECB mediante la función "DES_ecb_encrypt". La función toma como argumentos el mensaje cifrado, la variable que va a tomar el output, la clave DES y la dirección de una función de cifrado DES. La función cifra el mensaje en bloques de 8 bytes y reemplaza cada bloque con el bloque cifrado correspondiente.

En resumen, la función "encryptText" cifra un mensaje utilizando el algoritmo DES en modo ECB y lo rellena para que su longitud sea un múltiplo de 8 bytes. La función utiliza una llave con paridad par y un bloque de cifrado DES para cifrar el mensaje.

c. tryKey (key, *ciph, len)

Esta función se encarga de probar la llave que se le pasa y de hacer una copia del puntero de donde viene el texto cifrado para evitar cualquier condición de carrera la cual pueda causar que un proceso esté intentando descifrar un texto cifrado doblemente o inclusive ya descifrado. De último retorna si se pudo o no romper el programa.

Este método recibe los siguientes atributos:

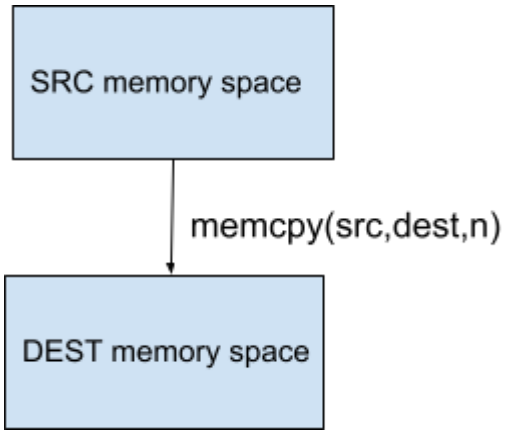
1. key que es de tipo long.
2. ciph que es donde viene el texto cifrado que es un puntero de char.
3. len que es de tipo int y denota el número de caracteres en el texto cifrado.

El pseudocódigo de este método es el siguiente:

- Se define un puntero char llamado “temp” que sea de tamaño “len” + 1
- Se copia el valor de “ciph” dentro de la variable “temp”
- Se manda a llamar a la función decrypt con los parámetros “key”, “temp”, “len”
- Retorna 1 o 0 dependiendo si logró o no romper el programa

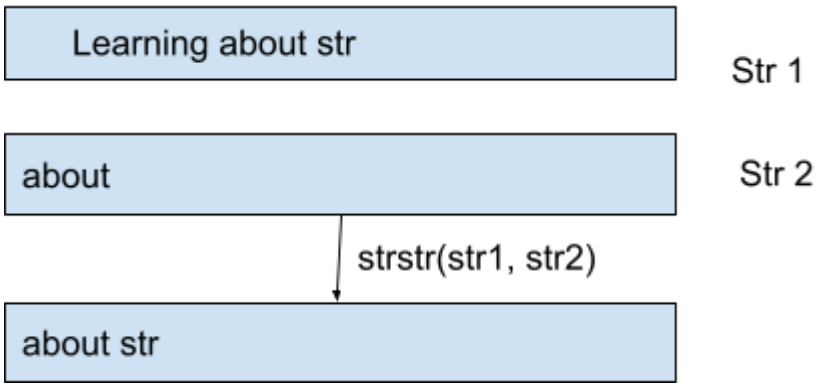
d. memcpy

Esta función se encarga básicamente de copiar cierta cantidad n de elementos que se encuentran en el espacio en memoria del elemento fuente hacia el espacio en memoria del elemento destino. (Tutorialspoint, s.f)



e. strstr

Esta función se utiliza para poder encontrar la primera ocurrencia de un substring que se coloca como parámetro en un string dado. (Tutorialspoint,s.f)



5. Describa y explique el uso y flujo de comunicación de las primitivas de MPI:
   a. MPI_Irecv

MPI_Irecv es una función de no bloqueo en la biblioteca MPI, que se utiliza para recibir mensajes de otros procesos en un programa paralelo. Proporciona una forma de recibir mensajes de forma asincrónica, lo que permite que el proceso de recepción continúe con otros cálculos mientras espera que llegue un mensaje.

El flujo de comunicación mediante MPI_Irecv implica varios pasos:

1. El proceso de recepción llama a MPI_Irecv para publicar una solicitud para recibir un mensaje. La función toma varios parámetros, incluido el búfer donde se almacenará el mensaje, el tipo de datos del mensaje y la fuente y la etiqueta del mensaje.

2. La biblioteca MPI agrega la solicitud de recepción a una cola de solicitudes pendientes e inmediatamente devuelve el control al proceso de recepción. El proceso de recepción puede continuar con otros cálculos mientras espera que llegue un mensaje.

3. Cuando llega un mensaje, la biblioteca MPI hace coincidir el mensaje con la solicitud de recepción pendiente y copia los datos del mensaje en el búfer de recepción. Si no hay ninguna solicitud de recepción coincidente pendiente, el mensaje se almacena en un búfer hasta que se publica una solicitud de recepción coincidente.

4. La biblioteca MPI envía una notificación al proceso de recepción, indicando que el mensaje ha llegado y la operación de recepción está completa.

5. El proceso de recepción puede entonces llamar a MPI_Test o MPI_Wait para comprobar si la operación de recepción se ha completado y recuperar el mensaje recibido del búfer de recepción.

MPI_Irecv es particularmente útil en los casos en que el proceso de recepción necesita realizar otros cálculos mientras espera que llegue un mensaje, como en el paralelismo. Al usar MPI_Irecv, el proceso de recepción puede superponerse a la comunicación y el cálculo, mejorando así el rendimiento del programa paralelo.

b. MPI_Send

MPI_Send es una función de suma importancia de MPI, esta es la que permite que se envíen datos entre procesos. El flujo de comunicación se basa en la preparación de datos, la especificación del proceso destino, envío de mensaje, recepción del mensaje y procesamiento de datos recibido.

La parte de preparación de datos se refiere a que el proceso original prepara los datos, variables, que desee dar a conocer. Estos pueden ser vectores, cadenas de caracteres, números, etc. La parte de especificación del destino hace referencia en dar a conocer el identificador del proceso destino al que se le quieren dar a conocer los datos, este identificador se conoce como “rango”. Siguiente a esto se hace el envío del mensaje, se usa la función MPI_Send para enviar los datos, esta función usa seis argumentos importantes; 1) el puntero al buffer de los datos. 2) El número de elementos que se van a enviar. 3) el tipo de datos. 4) El destino (rango). 5) La tag para identificar el mensaje. 6) El comunicador. Una vez se envía con esta función el hilo destino tiene que recibir la información con MPI_recv y por último solo tiene que procesar la información para utilizarla.

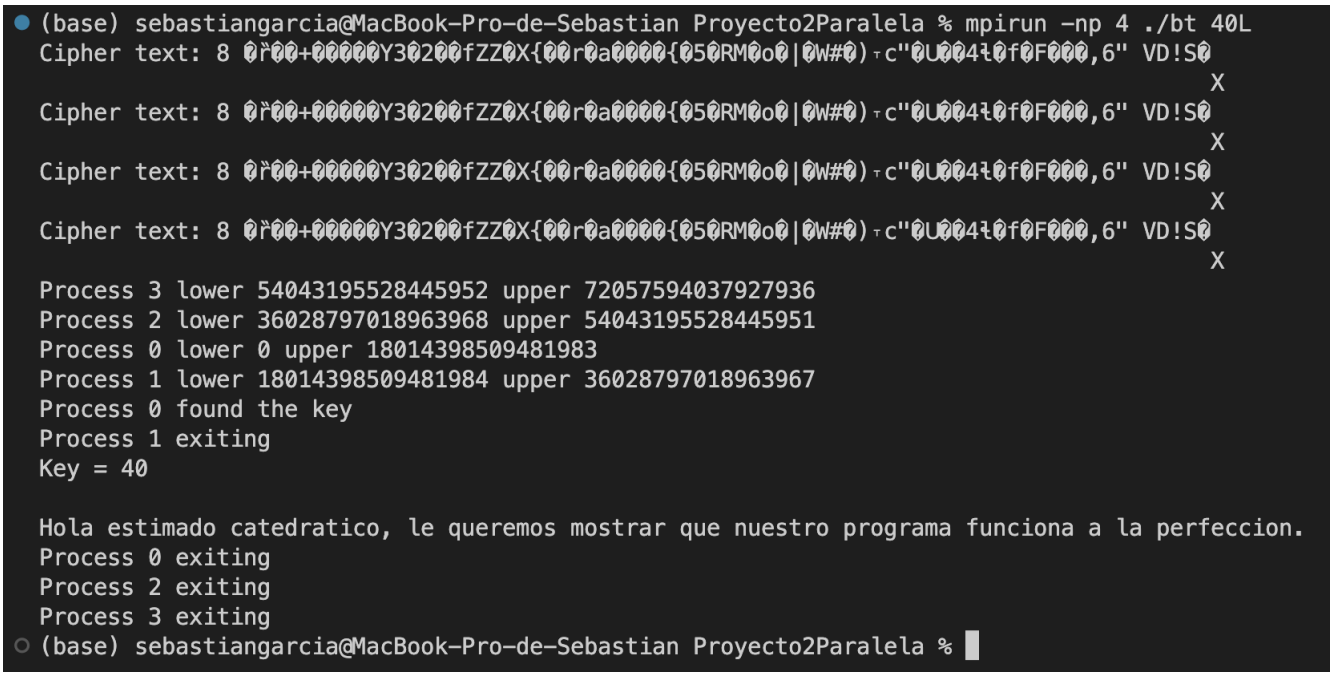
c. MPI_Wait

MPI_wait es una función de MPI que se utiliza para esperar la finalización de una operación de comunicación asíncrona. El flujo del proceso consta del inicio de la operación asíncrona, la llamada a MPI wait, la espera, la liberación de recursos y la continuación de la ejecución. De igual manera cabe resaltar que esta función se debe de llamar una vez para cada operación asíncrona iniciada. Si dado caso hay varias operaciones asíncronas en curso el mpi wait esperará hasta que la operación más antigua en cola sea completada.

La parte de inicio de operación asíncrona se refiere a que se inicia alguna operación de envío o recepción, puede ser mpi send o mpi recv, esta se ejecuta en segundo plano y permite que se continúe con otras tareas. Seguido de esto se corre la llamada mpi wait, la cual consta de dos parámetros. 1) Puntero al objeto de solicitud devuelto por la operación asíncrona (MPI_Request). 2) Status el cual es un objeto con los datos de envío. (MPI_Status). Una vez llamada la función se espera a que la función asíncrona se haya completado. Cuando se completa la solicitud asíncrona se procede a librar los recursos, esto quiere decir que la función de mpi wait libera cualquier recurso asociado con la solicitud, buffer de envío o recepción, y se marca la solicitud como completada. Una vez se marca como completada se continúa la ejecución del código normal.

Parte B

1. Ya que estamos familiarizados con el temario, vamos a analizar el problema más a fondo. Modifique su programa para que cifre un texto cargado desde un archivo (.txt) usando una llave privada arbitraria (como parámetro). Muestra una captura de pantalla evidenciando que puede cifrar y descifrar un texto sencillo (una oración) con una clave sencilla (por ejemplo 42).



2. Una vez listo el paso anterior, proceder a hacer las siguientes pruebas, evidenciando todo en su reporte. Para todas ellas utilice 4 procesos (-np 4). El texto a cifrar/descifrar: “Esta es una prueba de proyecto 2”. La palabra clave a buscar es: “es una prueba de”:
    a. Mida el tiempo de ejecución en romper el código usando la llave 123456L

```
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 4 ./bt 123456L
Cipher text: Y!X@@@@@@200000
K@
Cipher text: Y!X@@@@@@200000
K@
Cipher text: Y!X@@@@@@200000
K@
Cipher text: Y!X@@@@@@200000
K@
Process 0 lower 0 upper 18014398509481983
Process 1 lower 18014398509481984 upper 36028797018963967
Process 2 lower 36028797018963968 upper 54043195528445951
Process 3 lower 54043195528445952 upper 72057594037927936
Process 0 found the key
Key = 123456

Esta es una prueba de proyecto 2

Time to break the DES 0.055645
Process 0 exiting
Process 1 exiting
Process 2 exiting
Process 3 exiting
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela %
```

b. Mida el tiempo de ejecución en romper el código usando la llave: O sea 18014398509481983L. [spoiler: se tardará mucho, si es que(256/4) termina, no se ofusquen si no termina].

```
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 4 ./bt 18014398509481983L
Cipher text: @ard@@%
                    @@@M@@I@P@@#@@@z'@k@@>@P
Cipher text: @ard@@%
                    @@@M@@I@P@@#@@@z'@k@@>@P
Cipher text: @ard@@%
                    @@@M@@I@P@@#@@@z'@k@@>@P
Cipher text: @ard@@%
                    @@@M@@I@P@@#@@@z'@k@@>@P
Process 0 lower 0 upper 18014398509481983
Process 1 lower 18014398509481984 upper 36028797018963967
Process 2 lower 36028797018963968 upper 54043195528445951
Process 3 lower 54043195528445952 upper 72057594037927936
```

c. Mida el tiempo de ejecución en romper el código usando la llave: . O sea 18014398509481984L.(256/4) + 1

```
n@nwRHO@@Ut: r><@@Lp@NG@@1$@@@0@@Y)u@_@B@@@
n@nwRH@text: r><@@Lp@NG@@1$@@@0@@Y)u@_@B@@@
        @U
n@nwR@HyUxt: r><@@Lp@NG@@1$@@@0@@Y)u@_@B@@@
n@nwR@14@Ut: r><@@Lp@NG@@1$@@@0@@Y)u@_@B@@@
Process 3 lower 54043195528445952 upper 72057594037927936
Process 0 lower 0 upper 18014398509481983
Process 1 lower 18014398509481984 upper 36028797018963967
Process 2 lower 36028797018963968 upper 54043195528445951
Process 3 exiting
Process 1 found the key
Process 1 exiting
Key = 18014398509481984

Esta es una prueba de proyecto 2@
Time to break the DES 0.000000
Process 0 exiting
Process 2 exiting
```

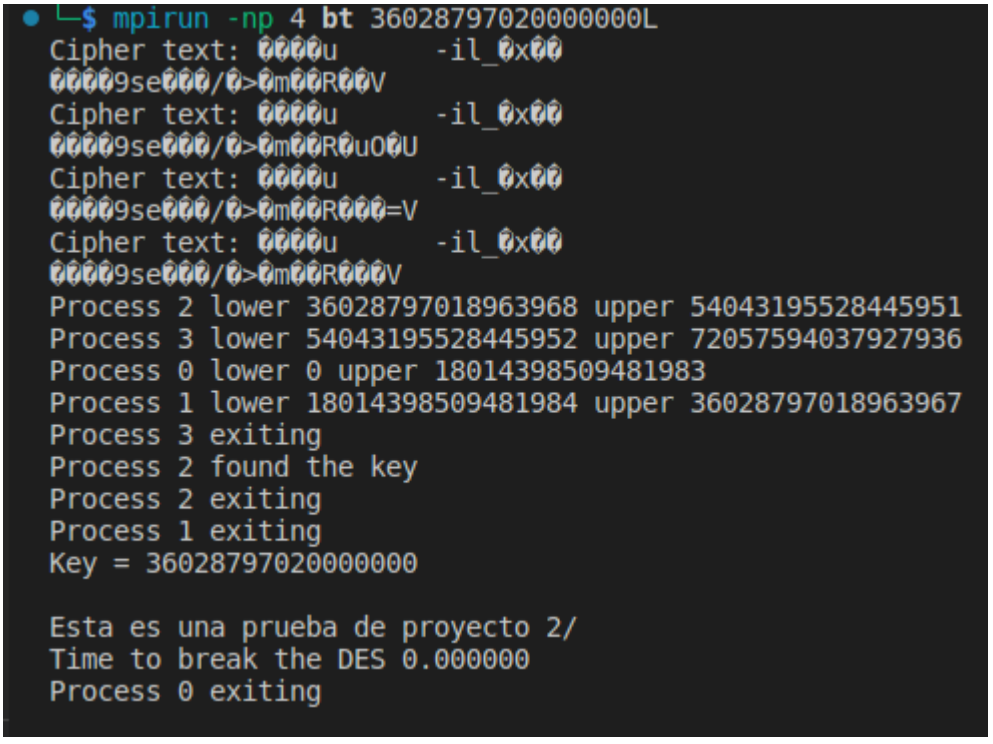
d. Reflexione lo observado y el comportamiento del tiempo en función de la llave.

Se puede observar que el tiempo depende de la posición del bloque en el que se encuentre la llave elegida. Por ejemplo, en la llave elegida del inciso B, el programa luego de cierto tiempo continúa ejecutándose, esto debido a que la llave se encuentra en la última posición del primer proceso por lo que la encontrará en la última iteración. Por otro lado, con el inciso C la llave se encuentra en la primera posición del bloque del segundo proceso y la encuentra en la primera iteración y el tiempo es prácticamente nulo.

Para poder ver y comprender tal fenómeno podemos realizar cambios a la llave privada y analizar el desempeño en función de ellas; asumiendo 4 procesos (ojo, adaptar dependiendo de los procesos que usen):

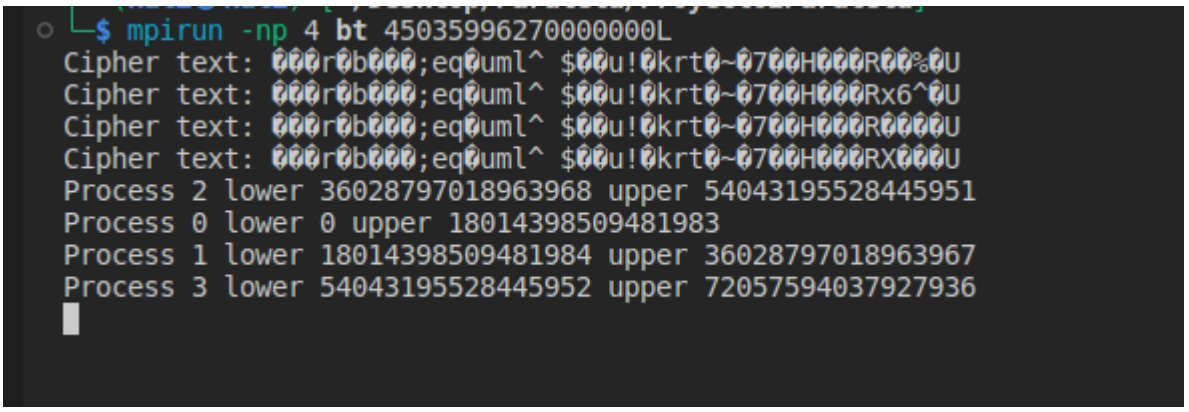
- **Una llave fácil de encontrar, por ejemplo, con valor de (2^56) / 2 + 1**

36028797020000000L



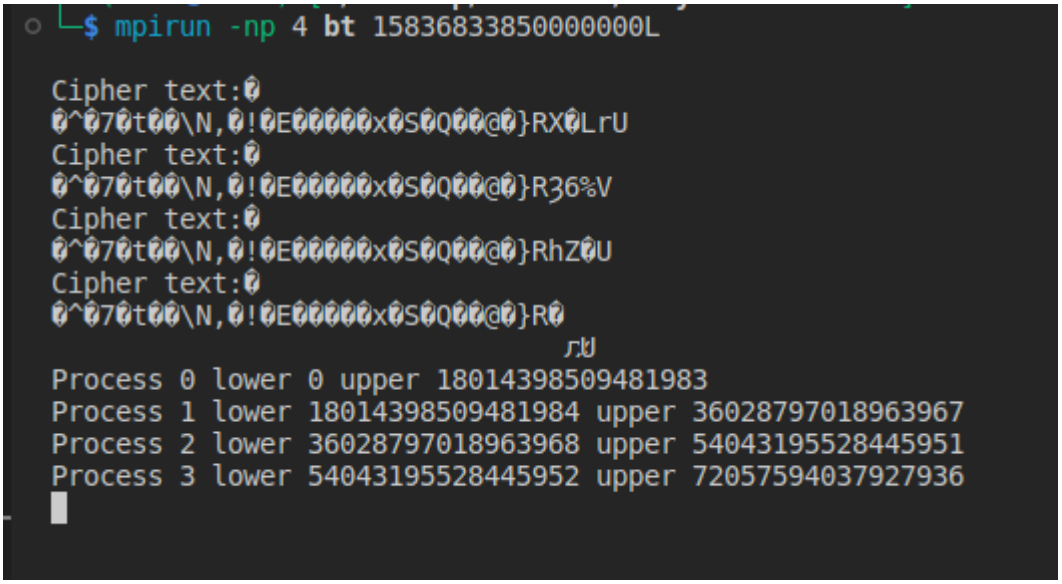
- **Una llave medianamente difícil de encontrar, por ejemplo, con valor de (2^56) / 2 + (2^56) / 8**

45035996270000000L



- **Una llave difícil de encontrar, por ejemplo, con valor de (2^56) / 7 + (2^56) / 13 aproximados al entero superior**

  15836833850000000L



Como podemos ver, el approach “naive” no es el mejor posible. Proponga, analice, e implemente 2 opciones alternativas al acercamiento “naive”. Tenga como objetivo en mente encontrar un algoritmo que tenga mejor “tiempo paralelo esperado” que la versión “naive” demostrada en ecuación (1). Para cada una no olvide:

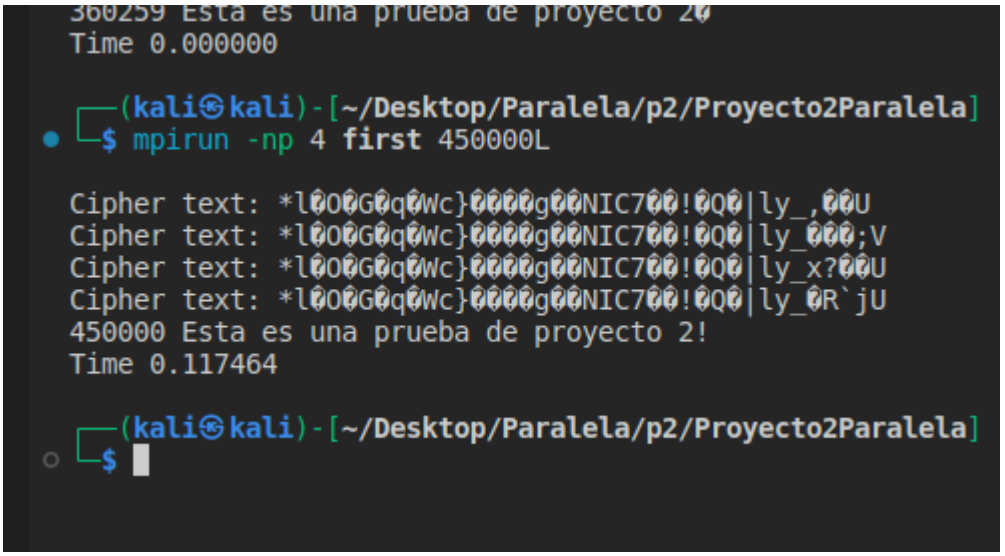
Adaptación 1 - Saltos (first_mpi.c)

Pseudocódigo:

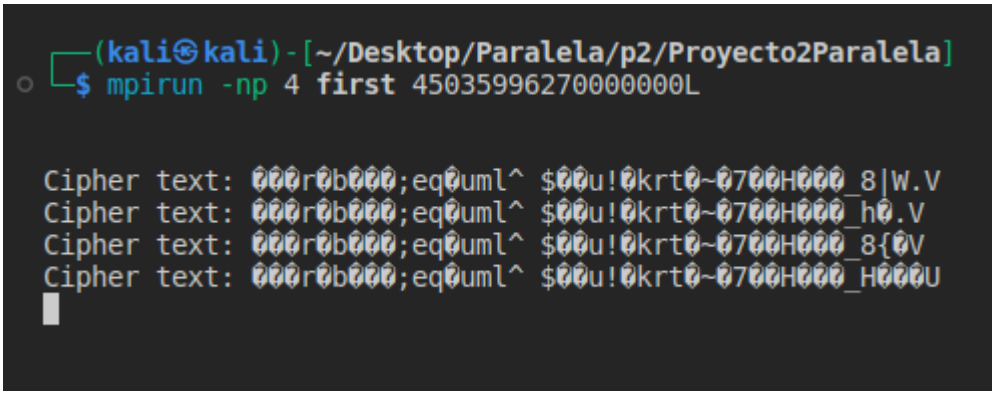
- Asignar N cantidad de pruebas a cada proceso
- i igual a id,
  - Si probar llave con puntero izquierdo
    - Mandarle a todos los procesos una señal que ya se descifró
  - Si probar llave con puntero derecho
    - Mandarle a todos los procesos una señal que ya se descifró
  - Si ya probó con todos entonces mpi test y salir

- Puntero izquierdo más igual N
- Puntero derecho menos igual N

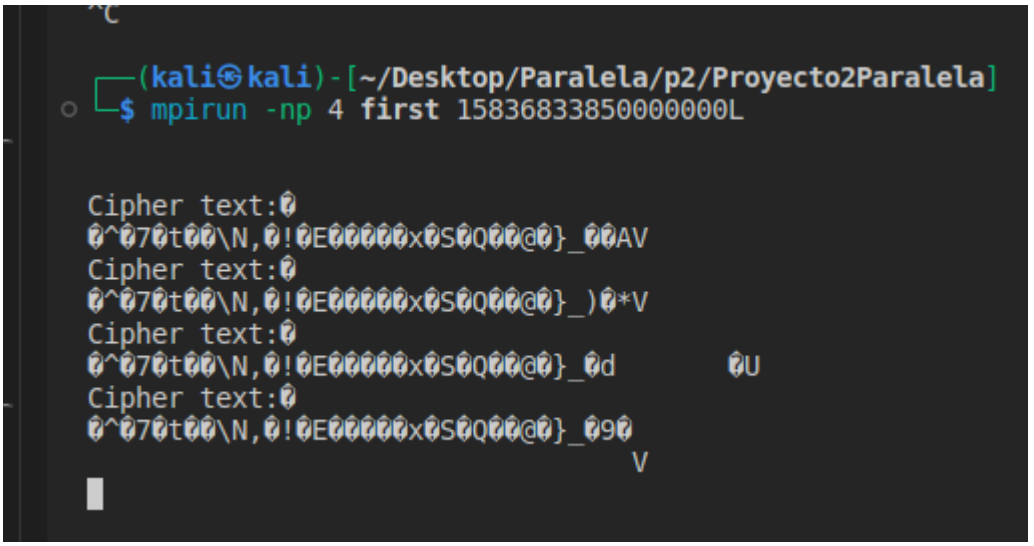
Llave fácil



Llave mediana



Llave difícil

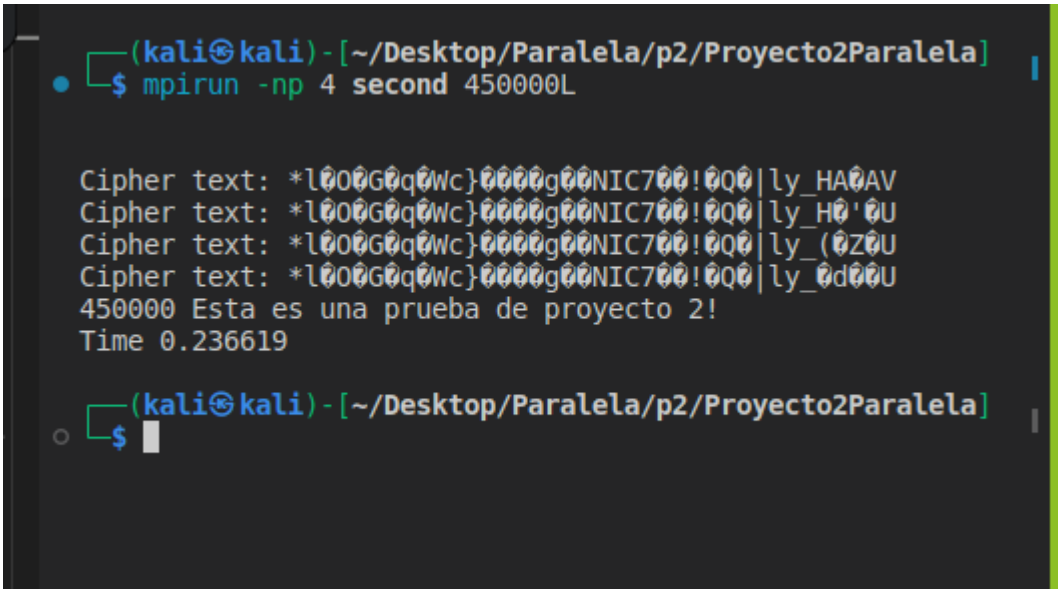


Adaptación 2 - Punteros (second_mpi.c)

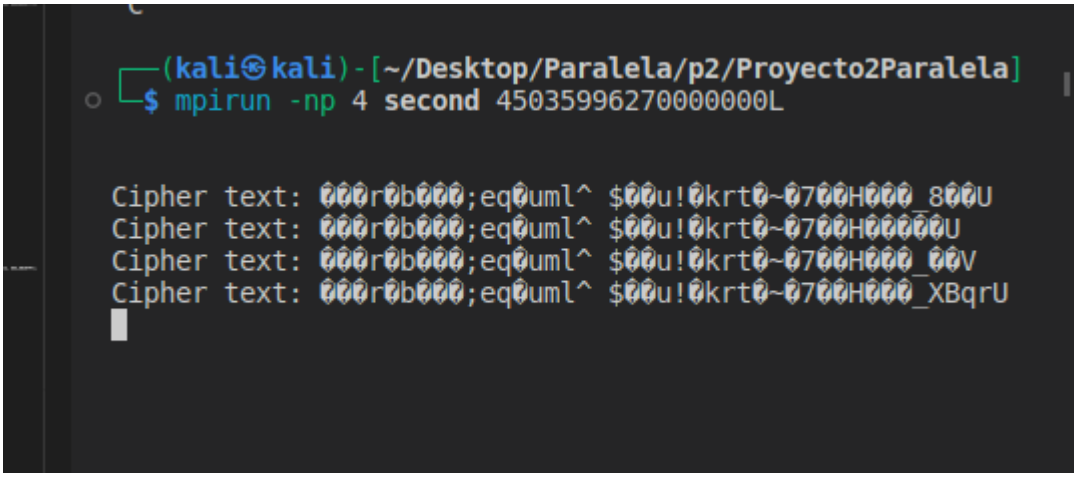
Pseudocódigo:

- Asignar N cantidad de pruebas a cada proceso
- Definir punteros izquierda y derecha
- Igualar el puntero izquierdo a id y el derecho a upper - 1
- Mientras puntero izquierdo sea distinto a puntero derecho
    - Si probar llave con puntero izquierdo
        - Mandarle a todos los procesos una señal que ya se descifró
    - Si probar llave con puntero derecho
        - Mandarle a todos los procesos una señal que ya se descifró
    - Si ya probó con todos entonces mpi test y salir
    - Puntero izquierdo más igual N
    - Puntero derecho menos igual N

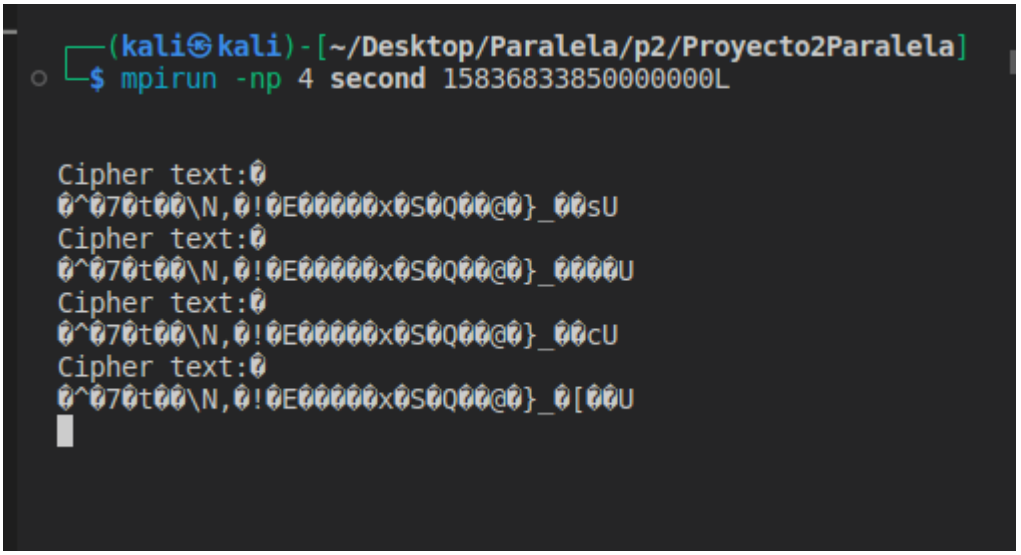
Llave facil



Llave mediana



Llave dificil



Valor esperado
Para el valor esperado del naive se agarran intervalos de llaves y se van probando. Sin embargo para el primer y segundo intento se hizo de forma t/2.

CONCLUSIONES

1. El proceso de quebrar un algoritmo y encontrar la llave por medio de fuerza bruta puede ser un proceso complejo computacionalmente hablando por lo que se requiere un buen análisis para dividir el proceso de búsqueda y evitar inconsistencias en las mediciones de desempeño.
2. Al paralelizar esta secuencia de datos tan grande reduce el tiempo de solución para la llave de la encriptación. Debido a que cada proceso tiene un rango de números para probar.
3. El tamaño del espacio clave y la longitud del texto cifrado determinarán en última instancia la viabilidad y eficacia del enfoque de fuerza bruta.

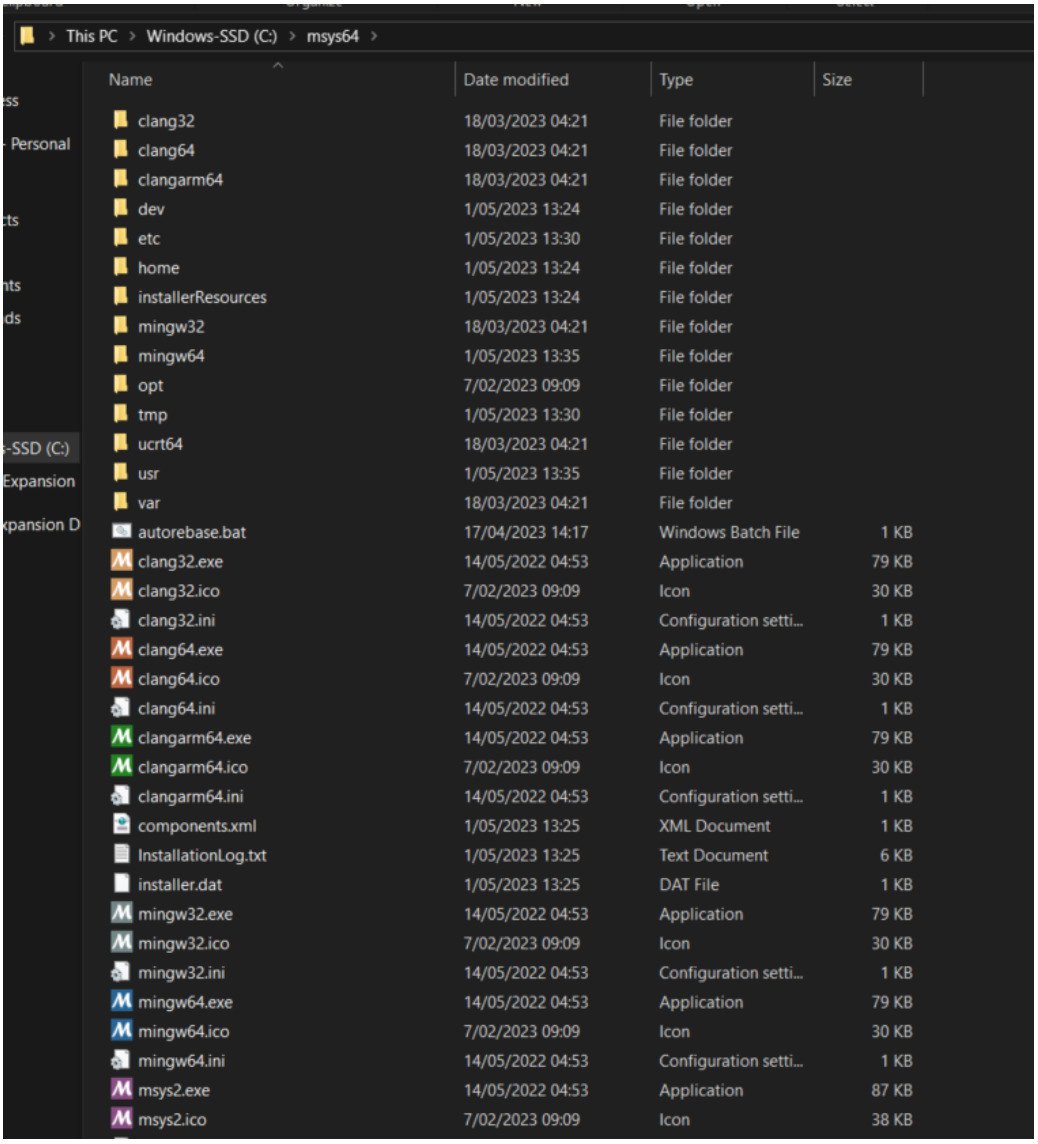
RECOMENDACIONES

1. Dividir el código en distintas funciones permite que se modularize de mejor manera el proceso y de esta manera se pueda analizar más rápido cualquier inconsistencia, especialmente en el proceso de encriptación y desencriptación.
2. La utilización de MPI es de gran ayuda para procesos como este de fuerza bruta para intentar descifrar la llave, pues es una librería que nos ayuda a optimizar procesos y operaciones en el código.
3. Se debe llevar una exhaustiva investigación antes de programar siempre para evitar errores en ejecución por falta de teoría. Un ejemplo podría ser el limpiar el buffer cada vez que se utilizara, ya que este a veces trae caracteres raros al final.

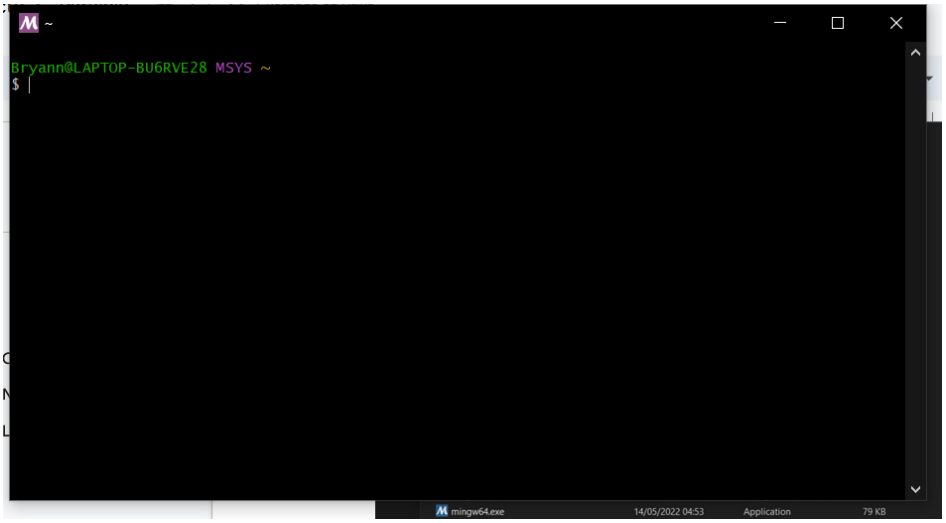
APÉNDICE

Instalación de openssl en windows

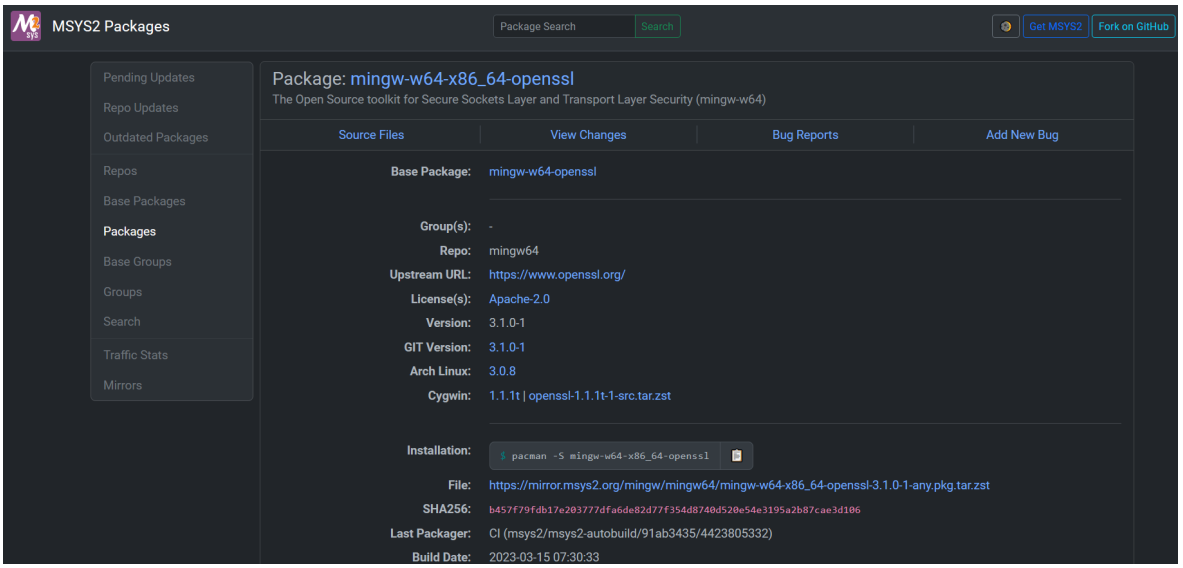
1. Como primer paso se puede verificar que se cuenta con msys2 instalado para poder utilizar el comando que instalará la librería.



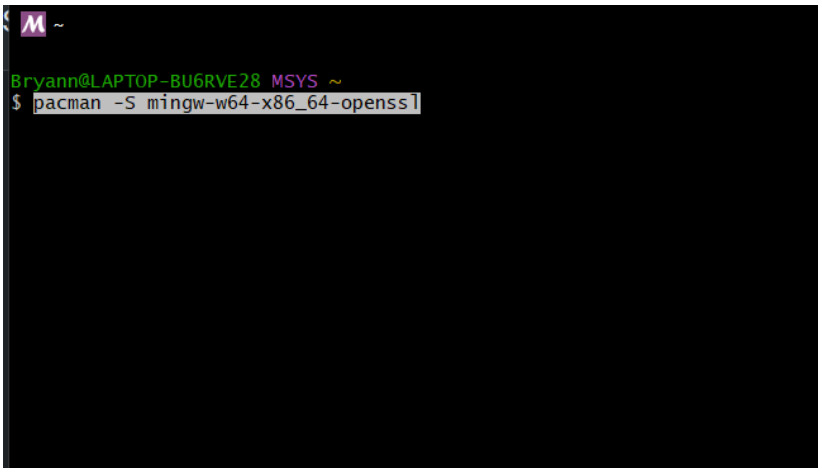
2. Luego se abre una consola de msys2. Esto se puede hacer haciendo doble click en el archivo msys2.exe



3. Seguidamente se puede buscar el paquete openssl en la página oficial de Msys2. En este link se encuentra el paquete a instalar: https://packages.msys2.org/package/mingw-w64-x86_64-openssl



4. Al ejecutar el comando en la consola, el paquete quedará listo para poder utilizarse.



5. En código la forma de incluir la librería es de la siguiente manera:

   #include “openssl/des.h”

6. Para realizar la compilación se puede utilizar las siguientes flags:

   -lssl -lcrypto

ANEXO 1 - Catálogo de funciones y estructuras

readFile

Descripción: Función que se utiliza para realizar la lectura de un archivo de texto. Toma el nombre del archivo y se aloja un espacio de memoria para el buffer que contendrá el texto, luego por medio de la función **fread** se copia el contenido del archivo al buffer.

Entradas:

1. filename
   a. Tipo: char
   b. Descripción: Contiene el nombre del archivo de texto a leer para realizar la encriptación.

Salidas:

1. buffer
   a. Tipo: char
   b. Descripción: Se utiliza para guardar el contenido del archivo de texto y poder ser utilizado en el proceso de cifrado y fuerza bruta.

decrypt

Descripción: Función que se utiliza para poder realizar la desencriptación de un texto. Toma la llave utilizada, el texto cifrado y la longitud del texto para aplicar la desencriptación por medio de bloques utilizando la función **DES_ecb_encrypt** de openssl con la flag DES_DECRYPT.

Entradas:

1. key
   a. Tipo: long
   b. Descripción: Variable que contiene la llave a utilizar para la desencriptación
2. ciph
   a. Tipo: char

b. Descripción: Variable que contiene el texto que se intentará descifrar con la key brindada.

3. len
    a. Tipo: int
    b. Descripción: Contiene la longitud del mensaje que se desea descifrar.

Salidas:

1. La función es de tipo void, por lo que no hay salida.

encryptText

Descripción: Función que se utiliza para poder realizar la encriptación de un texto. Toma la llave utilizada, el texto y la longitud del texto para aplicar la encriptación por medio de bloques utilizando la función **DES_ecb_encrypt** de openssl con la flag DES_ENCRYPT.

Entradas:

2. key
    c. Tipo: long
    d. Descripción: Variable que contiene la llave a utilizar para la encriptación
4. ciph
    a. Tipo: char
    b. Descripción: Variable que contiene el texto que se encriptará con la key proporcionada.
5. len
    a. Tipo: int
    b. Descripción: Contiene la longitud del mensaje que se desea encriptar

Salidas:

2. La función es de tipo void, por lo que no hay salida.

tryKey

Descripción: Función que se utiliza para probar la llave con el texto y verificar si el substring que se busca se encuentra en el texto encriptado para validar la key.

Entradas:

3. key
    e. Tipo: long
    f. Descripción: Variable que contiene la llave a utilizar para la desencriptación
6. ciph
    a. Tipo: char

   b. Descripción: Variable que contiene el texto que se desencripta con la llave proporcionada.
7. len
   a. Tipo: int
   b. Descripción: Contiene la longitud del mensaje que se desea descifrar

Salidas:

1. tipo: int
2. Descripción: Entero que contiene el valor de la evaluación del substring con el texto desencriptado para verificar si coincide.

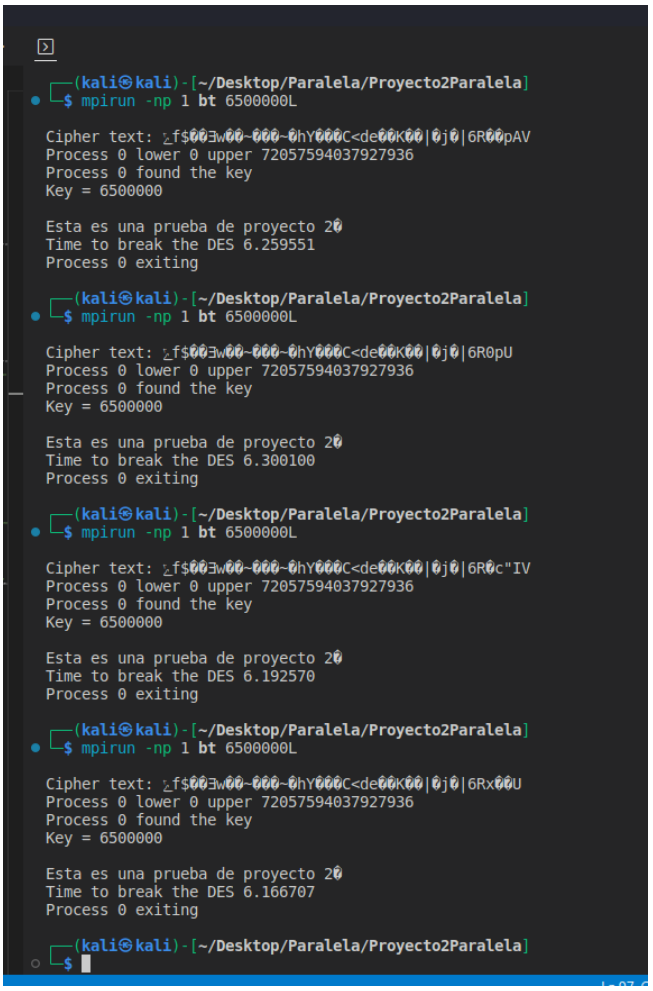
ANEXO 2 - Bitácora

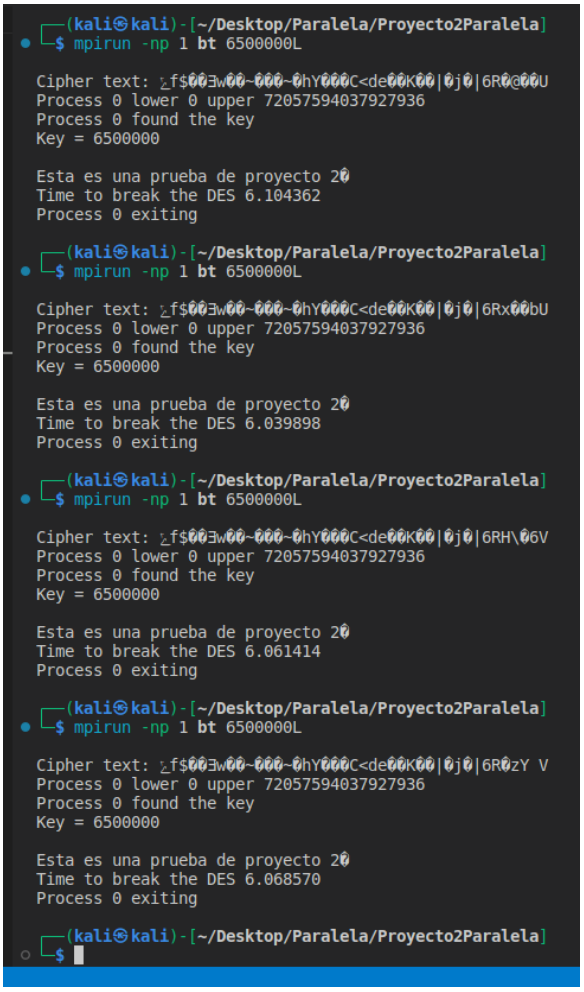
Programa Naive - naive.c

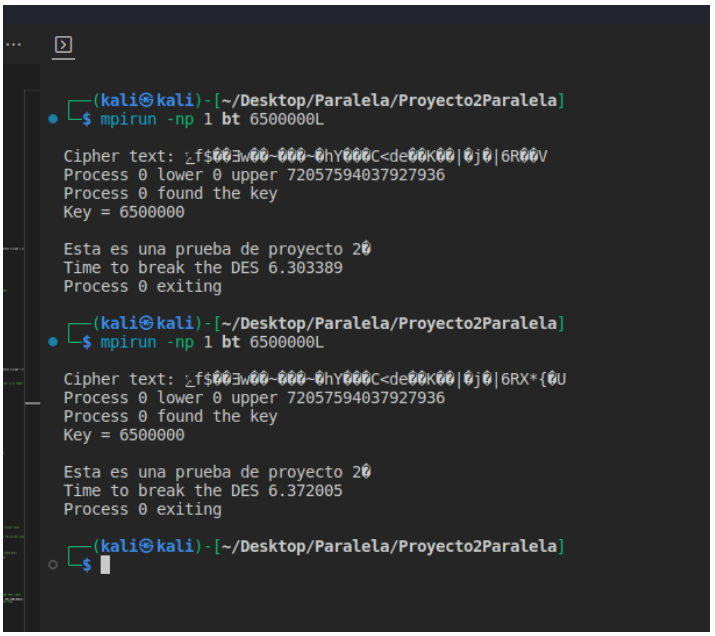
Llave: 6500000L

Secuencial

Iteración	Tiempo (s)
1	6,269551
2	6,3001
3	6,19257
4	6,166707
5	6,104362
6	6,039898
7	6,061414
8	6,06857
9	6,303389
10	6,372005
Promedio	6,19


(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
$ mpirun -np 1 bt 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6R@@pAV
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000
Esta es una prueba de proyecto 2@
Time to break the DES 6.259551
Process 0 exiting
(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
$ mpirun -np 1 bt 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6R0pU
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000
Esta es una prueba de proyecto 2@
Time to break the DES 6.300100
Process 0 exiting
(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
$ mpirun -np 1 bt 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6R@c"IV
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000
Esta es una prueba de proyecto 2@
Time to break the DES 6.192570
Process 0 exiting
(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
$ mpirun -np 1 bt 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6Rx@@U
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000
Esta es una prueba de proyecto 2@
Time to break the DES 6.166707
Process 0 exiting
(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
$


(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
$ mpirun -np 1 bt 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6R@@@@U
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000
Esta es una prueba de proyecto 2@
Time to break the DES 6.104362
Process 0 exiting
(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
$ mpirun -np 1 bt 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6Rx@@bU
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000
Esta es una prueba de proyecto 2@
Time to break the DES 6.039898
Process 0 exiting
(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
$ mpirun -np 1 bt 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6RH\@6V
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000
Esta es una prueba de proyecto 2@
Time to break the DES 6.061414
Process 0 exiting
(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
$ mpirun -np 1 bt 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6R@zY V
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000
Esta es una prueba de proyecto 2@
Time to break the DES 6.068570
Process 0 exiting
(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
$

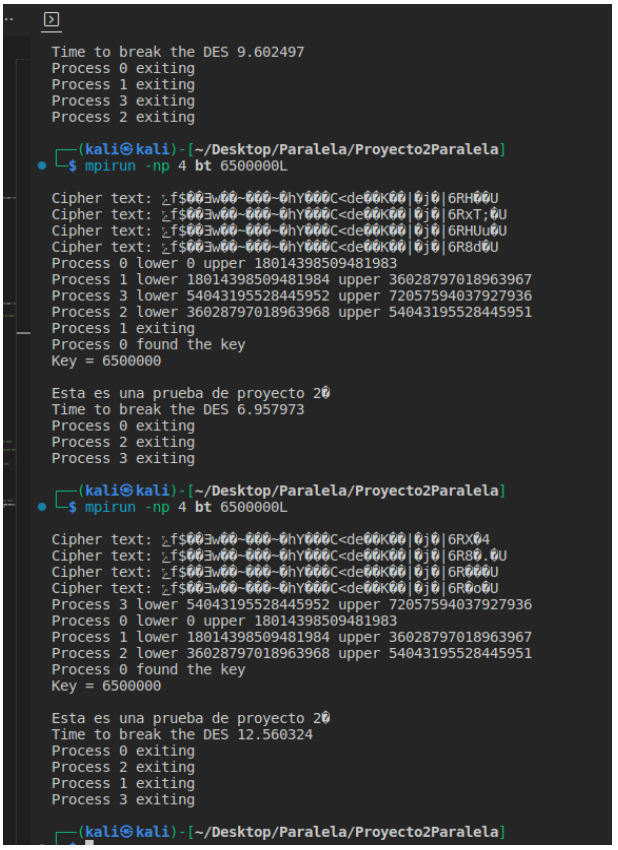

(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
$ mpirun -np 1 bt 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6R@@V
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000
Esta es una prueba de proyecto 2@
Time to break the DES 6.303389
Process 0 exiting
(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
$ mpirun -np 1 bt 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6RX*{@U
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000
Esta es una prueba de proyecto 2@
Time to break the DES 6.372005
Process 0 exiting
(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
$

Paralelo - n = 4

Iteración	Tiempo	Speedup	Eficiencia
1	6,153232	1,005627059	0,2514067648
2	6,507974	0,9508115122	0,237702878
3	6,705342	0,9228249059	0,2307062265
4	11,967519	0,5170542533	0,1292635633
5	7,026296	0,88067121	0,2201678025
6	9,602497	0,6444007845	0,1611001961
7	6,957973	0,8893188577	0,2223297144
8	12,560324	0,4926510335	0,1231627584
9	7,125939	0,8683566615	0,2170891654
10	7,647562	0,8091280071	0,2022820018
Promedio	8,23	0,80	0,20

```
┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 4 bt 6500000L

Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R�␣cU
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6j-�U
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R89��U
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R␣ ��U
Process 1 lower 18014398509481984 upper 36028797018963967
Process 3 lower 54043195528445952 upper 72057594037927936
Process 2 lower 36028797018963968 upper 54043195528445951
Process 0 lower 0 upper 18014398509481983
Process 0 found the key
Process 1 exiting
Key = 6500000

Esta es una prueba de proyecto 2�
Time to break the DES 6.153232
Process 0 exiting
Process 2 exiting
Process 3 exiting

┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 4 bt 6500000L

Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R8�2V
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R���KV
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R�:~U
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R
q�U
Process 2 lower 36028797018963968 upper 54043195528445951
Process 0 lower 0 upper 18014398509481983
Process 3 lower 54043195528445952 upper 72057594037927936
Process 1 lower 18014398509481984 upper 36028797018963967
Process 2 exiting
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2�
Time to break the DES 6.507974
Process 0 exiting
Process 1 exiting
Process 3 exiting

┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 4 bt 6500000L

Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R���CV
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6RX/�U
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R(%��U
```

```
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2�
Time to break the DES 6.705342
Process 0 exiting
Process 2 exiting
Process 1 exiting

┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 4 bt 6500000L

Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6Rx�j�U
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R����U
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R�Z�V
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6RZz�U
Process 0 lower 0 upper 18014398509481983
Process 1 lower 18014398509481984 upper 36028797018963967
Process 2 lower 36028797018963968 upper 54043195528445951
Process 3 lower 54043195528445952 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2�
Time to break the DES 11.967519
Process 0 exiting
Process 3 exiting
Process 2 exiting
Process 1 exiting

┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 4 bt 6500000L

Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6RxgHV
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R(e��U
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R�oV
Cipher text: ∑f$��∃w��~���~�hY���C<de��K��|�j�|6R���U
Process 2 lower 36028797018963968 upper 54043195528445951
Process 1 lower 18014398509481984 upper 36028797018963967
Process 3 lower 54043195528445952 upper 72057594037927936
Process 0 lower 0 upper 18014398509481983
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2�
Time to break the DES 7.026296
Process 0 exiting
Process 3 exiting
Process 1 exiting
Process 2 exiting
```



Paralelo - n = 2

Iteración	Tiempo	Speedup	Eficiencia
1	6,583696	0,9398758084	0,4699379042
2	6,683268	0,9258728814	0,4629364407
3	6,788797	0,911480576	0,455740288
4	6,608266	0,9363812837	0,4681906418
5	6,775098	0,9133235564	0,4566617782
6	6,57437	0,9412090588	0,4706045294
7	6,474925	0,9556645984	0,4778322992
8	6,589587	0,9390355723	0,4695177862
9	6,936355	0,8920905288	0,4460452644
10	6,584427	0,9397714638	0,4698857319
Promedio	6,66	0,93	0,46

```
┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 2 bt 6500000L

Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐÐnEV
Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐ_><V
Process 0 lower 0 upper 36028797018963967
Process 1 lower 36028797018963968 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2Ð
Time to break the DES 6.583696
Process 0 exiting
Process 1 exiting

┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 2 bt 6500000L

Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RHÐrU
Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RXÐÐÐU
Process 1 lower 36028797018963968 upper 72057594037927936
Process 0 lower 0 upper 36028797018963967
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2Ð
Time to break the DES 6.683268
Process 0 exiting
Process 1 exiting

┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 2 bt 6500000L

Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐÐV
Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐÐÐV
Process 0 lower 0 upper 36028797018963967
Process 1 lower 36028797018963968 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2Ð
Time to break the DES 6.788797
Process 0 exiting
Process 1 exiting

┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$
```

```
┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 2 bt 6500000L

Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐÐÐU
Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐ5ÐU
Process 0 lower 0 upper 36028797018963967
Process 1 lower 36028797018963968 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2Ð
Time to break the DES 6.608266
Process 0 exiting
Process 1 exiting

┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 2 bt 6500000L

Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RH;ÐÐU
Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐ'ÐV
Process 0 lower 0 upper 36028797018963967
Process 1 lower 36028797018963968 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2Ð
Time to break the DES 6.775098
Process 0 exiting
Process 1 exiting

┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 2 bt 6500000L

Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐÐCoU
Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐÐqÐU
Process 1 lower 36028797018963968 upper 72057594037927936
Process 0 lower 0 upper 36028797018963967
Process 0 found the key
Process 1 exiting
Key = 6500000

Esta es una prueba de proyecto 2Ð
Time to break the DES 6.574370
Process 0 exiting
```

```
┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 2 bt 6500000L

Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐÐÐ}U
Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6Rx ÐIÐU
Process 0 lower 0 upper 36028797018963967
Process 1 lower 36028797018963968 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2Ð
Time to break the DES 6.474925
Process 0 exiting
Process 1 exiting

┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 2 bt 6500000L

Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐÐkU
Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐ4V
Process 0 lower 0 upper 36028797018963967
Process 1 lower 36028797018963968 upper 72057594037927936
Process 1 exiting
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2Ð
Time to break the DES 6.589587
Process 0 exiting

┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 2 bt 6500000L

Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐD9ÐU
Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RÐoÐYU
Process 0 lower 0 upper 36028797018963967
Process 1 lower 36028797018963968 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2Ð
Time to break the DES 6.936355
Process 0 exiting
Process 1 exiting

┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$
```

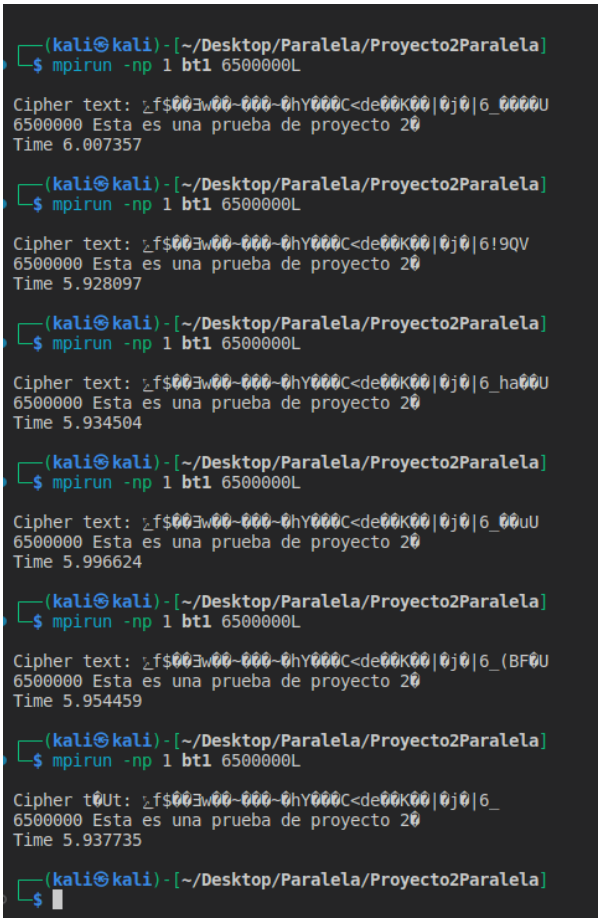
```
┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$ mpirun -np 2 bt 6500000L

Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RXÐtÐU
Cipher text: ¿f$ÐÐ∃wÐÐ~ÐÐÐ~ÐhYÐÐÐC<deÐÐKÐÐ|Ðj Ð|6RH&
V
Process 0 lower 0 upper 36028797018963967
Process 1 lower 36028797018963968 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2Ð
Time to break the DES 6.584427
Process 0 exiting
Process 1 exiting

┌──(kali㉿kali)-[~/Desktop/Paralela/Proyecto2Paralela]
└─$
```

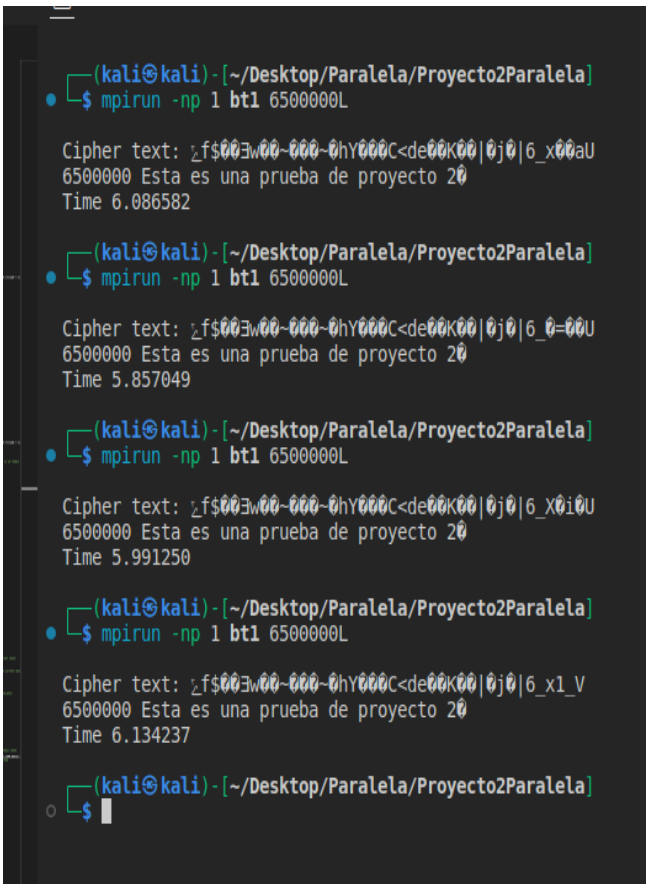
Adaptación 1 - Saltos - first_mpi.c

Llave: 6500000L

Secuencial

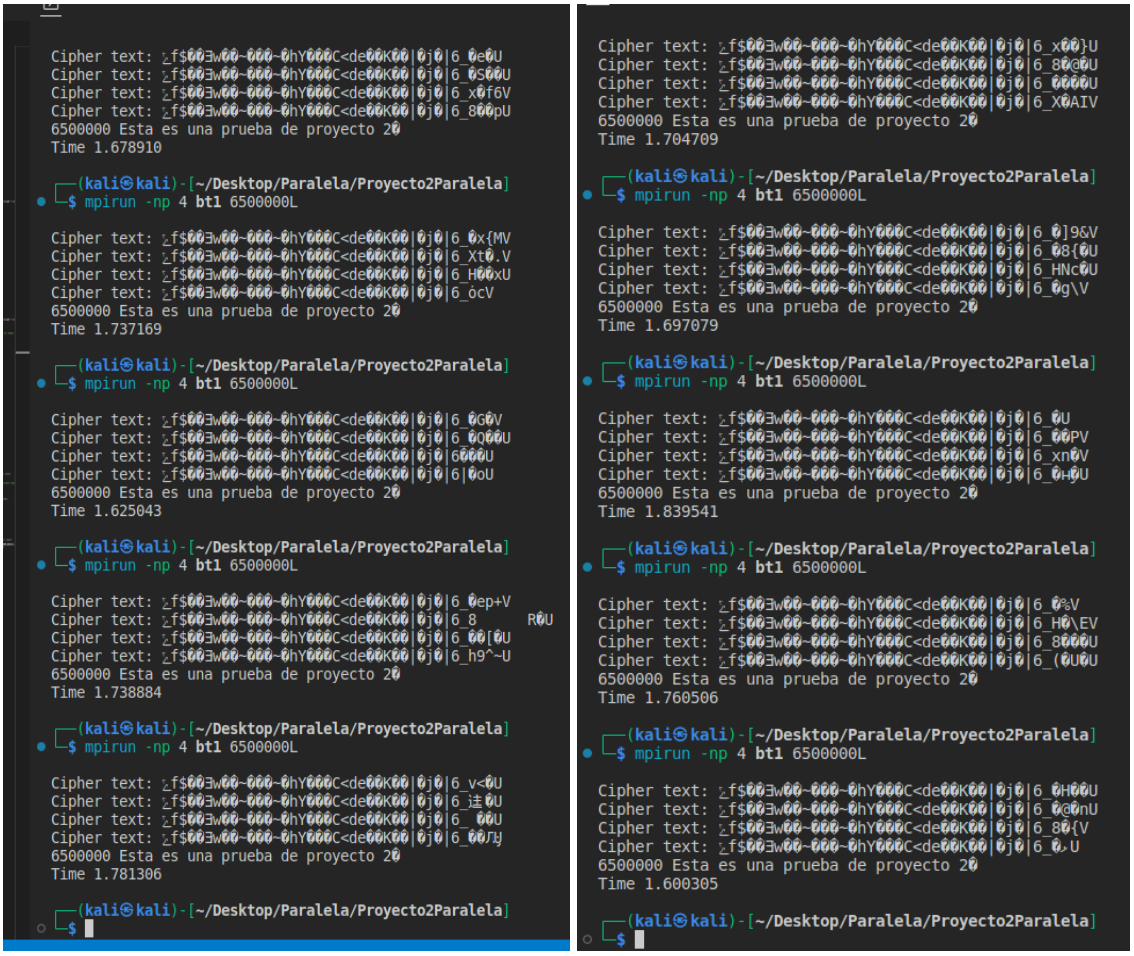
Iteración	Tiempo (s)
1	6,007357
2	5,9280897
3	5,934504
4	5,996624
5	5,954459
6	5,937735
7	6,086582
8	5,857049
9	5,99125
10	6,134237
Promedio	5,98





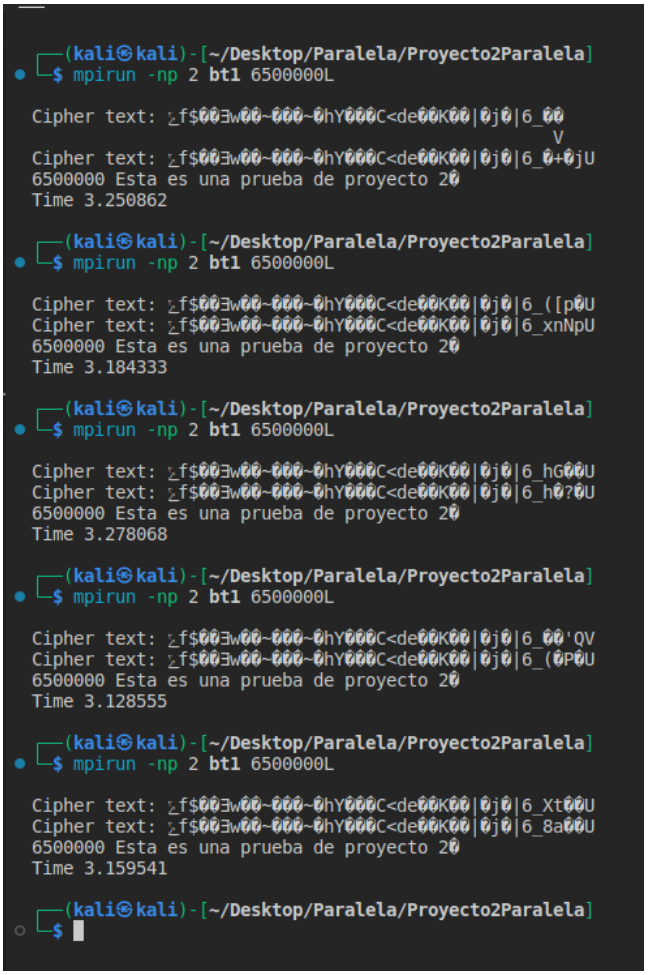
Paralelo - n = 4

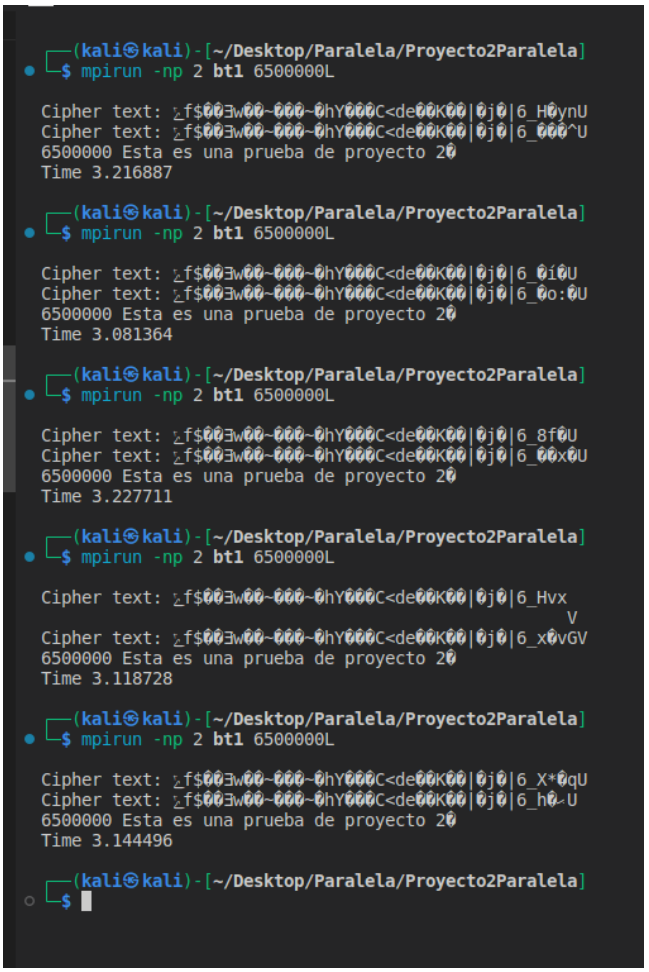
Iteración	Tiempo	Speedup	Eficiencia
1	1,67891	3,563495762	0,8908739405
2	1,737169	3,443987701	0,8609969252
3	1,625043	3,681618683	0,9204046708
4	1,738884	3,440591017	0,8601477542
5	1,781306	3,35865296	0,8396632401
6	1,7047090	3,525344825	0,8813362062
7	1,697079	3,525344825	0,8813362062
8	1,839541	3,252326896	0,8130817239
9	1,760506	3,398334723	0,8495836808
10	1,600305	3,738530261	0,9346325654
Promedio	1,72	3,49	0,87



Paralelo - n = 2

Iteración	Tiempo	Speedup	Eficiencia
1	3,25086	1,840371062	0,9201855309
2	3,18433	1,878821815	0,9394109075
3	3,278068	1,825095962	0,912547981
4	3,128555	1,912316923	0,9561584613
5	3,159541	1,8935626	0,9467812999
6	3,216887	1,85980691	0,9299034548
7	3,081364	1,941604001	0,9708020003
8	3,227711	1,853570121	0,9267850607
9	3,118728	1,918342565	0,9591712823
10	3,144496	1,902622446	0,9513112228
Promedio	3,18	1,88	0,94





Adaptación 2 - second_mpi.c

Llave: 6500000L

Secuencial

Iteración	Tiempo (s)
1	5,488660
2	5,536377
3	5,569862
4	5,513971
5	5,505519
6	5,519402
7	5,541430
8	5,635812
9	5,513594
10	5,669109

Promedio	5,549374

```
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 1 ./bt4 6500000L
Cipher text: ¿f$��∃w��~���~�hY���C<de��K��|�j�|6
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2�
Time to break the DES 5.488660
Process 0 exiting
```

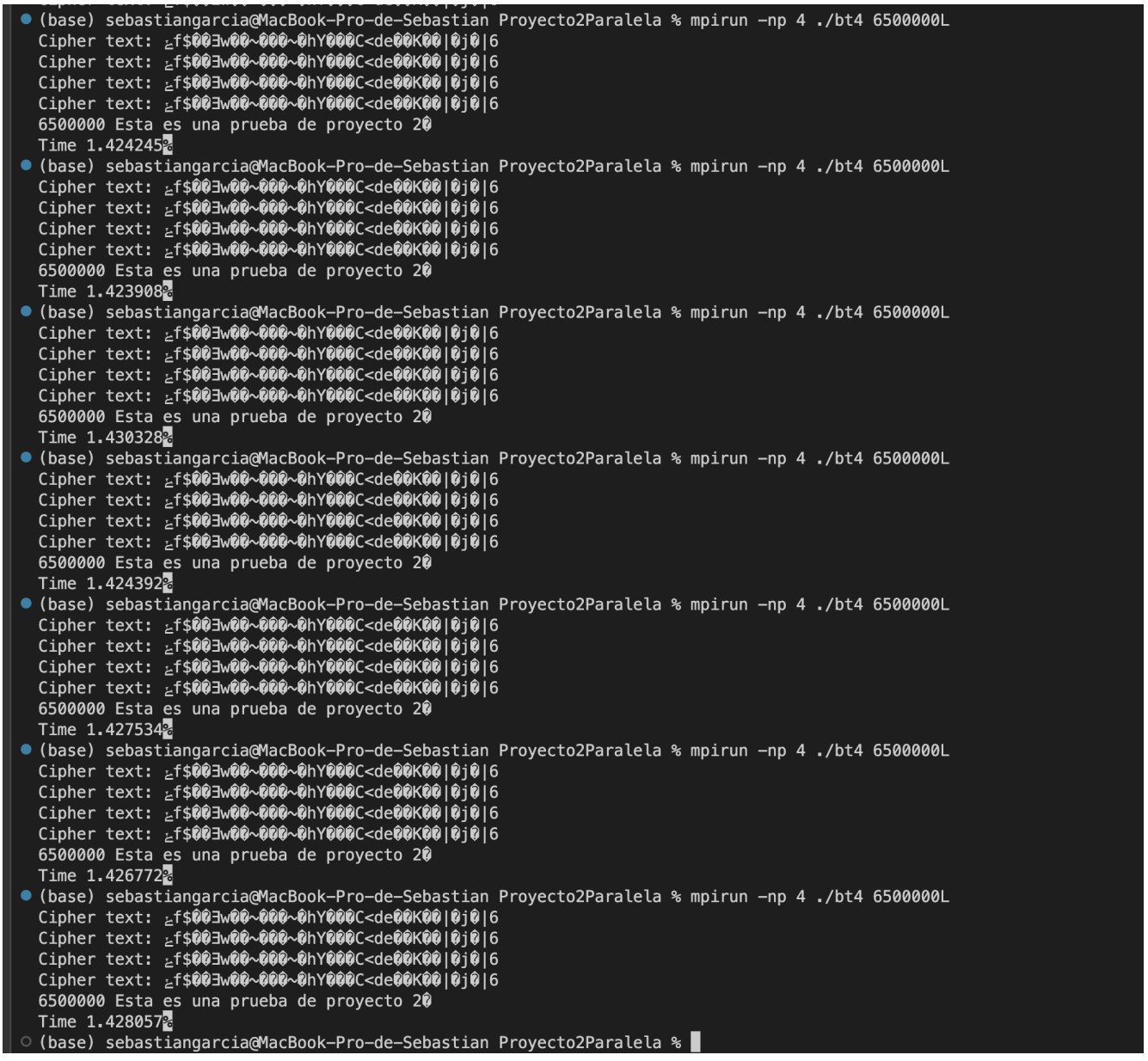
```
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 1 ./bt4 6500000L
Cipher text: ¿f$��∃w��~���~�hY���C<de��K��|�j�|6
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2�
Time to break the DES 5.536377
Process 0 exiting
```

```
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 1 ./bt4 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2@
Time to break the DES 5.569862
Process 0 exiting
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 1 ./bt4 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2@
Time to break the DES 5.513971
Process 0 exiting
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 1 ./bt4 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2@
Time to break the DES 5.505519
Process 0 exiting
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 1 ./bt4 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2@
Time to break the DES 5.519402
Process 0 exiting
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 1 ./bt4 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Process 0 lower 0 upper 72057594037927936
Process 0 found the key
Key = 6500000

Esta es una prueba de proyecto 2@
Time to break the DES 5.541430
Process 0 exiting
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela %
```

Paralelo - n = 4

Iteración	Tiempo	Speedup	Eficiencia
1	1,424245	3,896361651	0,9740904128
2	1,423908	3,897283813	0,9743209533
3	1,430328	3,879790929	0,9699477323
4	1,424392	3,895959539	0,9739898848
5	1,427534	3,887384539	0,9718461347
6	1,426772	3,889460685	0,9723651712
7	1,428057	3,885960855	0,9714902136

8	1,427815	3,886619485	0,9716548713
9	1,43308	3,872340414	0,9680851034
10	1,548655	3,583350456	0,8958376139
Promedio	1,439479	3,857451	0,964363



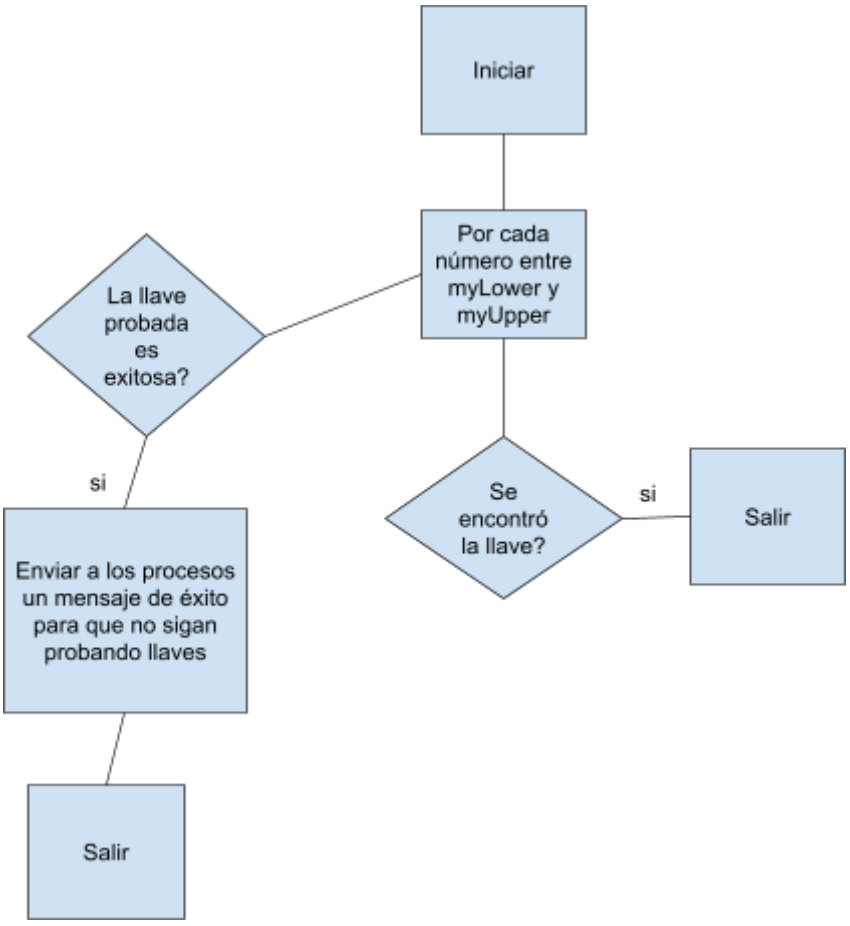
```
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 4 ./bt4 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
6500000 Esta es una prueba de proyecto 2@
Time 1.427815%
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 4 ./bt4 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
6500000 Esta es una prueba de proyecto 2@
Time 1.433080%
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 4 ./bt4 6500000L
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@∃w@@~@@@~@hY@@@C<de@@K@@|@j@|6
6500000 Esta es una prueba de proyecto 2@
Time 1.548655%
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela %
```

Paralelo - n = 2

Iteración	Tiempo	Speedup	Eficiencia
1	2,806685	1,977198581	0,9885992906
2	2,799888	1,981998423	0,9909992114
3	2,802624	1,98006354	0,9900317702
4	2,803081	1,979740721	0,9898703605
5	2,805355	1,978135958	0,9890679789
6	2,802989	1,9798057	0,9899028501
7	2,800935	1,981257544	0,9906287722
8	2,807447	1,976661928	0,988330964
9	2,802935	1,979843842	0,9899219211
10	2,833737	1,958323444	0,9791617218
Promedio	2,81	1,98	0,99

```
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 2 ./bt4 6500000L
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
6500000 Esta es una prueba de proyecto 2@
Time 2.806685%
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 2 ./bt4 6500000L
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
6500000 Esta es una prueba de proyecto 2@
Time 2.799888%
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 2 ./bt4 6500000L
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
6500000 Esta es una prueba de proyecto 2@
Time 2.802624%
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 2 ./bt4 6500000L
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
6500000 Esta es una prueba de proyecto 2@
Time 2.803081%
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 2 ./bt4 6500000L
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
6500000 Esta es una prueba de proyecto 2@
Time 2.805355%
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 2 ./bt4 6500000L
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
6500000 Esta es una prueba de proyecto 2@
Time 2.802989%
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 2 ./bt4 6500000L
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
6500000 Esta es una prueba de proyecto 2@
Time 2.800935%
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 2 ./bt4 6500000L
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
6500000 Esta es una prueba de proyecto 2@
Time 2.807447%
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 2 ./bt4 6500000L
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
6500000 Esta es una prueba de proyecto 2@
Time 2.802935%
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela % mpirun -np 2 ./bt4 6500000L
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
Cipher text: ¿f$@@Ǝw@@~@@@~@hY@@@C<de@@K@@|@j@|6
6500000 Esta es una prueba de proyecto 2@
Time 2.833737%
(base) sebastiangarcia@MacBook-Pro-de-Sebastian Proyecto2Paralela %
```

ANEXO 3 - Diagrama de flujo



LITERATURA CITADA

Tutorialspoint. (s.f). C library function - memcpy(). Extraído de: https://www.tutorialspoint.com/c_standard_library/c_function_memcpy.htm

Tutorialspoint. (s.f). C library function - strstr(). Extraído de: https://www.tutorialspoint.com/c_standard_library/c_function_strstr.htm

Simplilearn . (2022). What Is DES (Data Encryption Standard)? DES Algorithm and Operation. Extraído de: https://www.simplilearn.com/what-is-des-article

KeepCoding. (2023). ¿Qué es el algoritmo DES?. Extraído de: https://keepcoding.io/blog/que-es-el-algoritmo-des/

Geek For Geeks. (2023). Data encryption standard (DES). Extraído de: https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/

Neo. (s.f.) DES Extraido de: https://neo.lcc.uma.es/evirtual/cdd/tutorial/presentacion/des.html

Keep Coding. (2023) ¿Que es el algoritmo DES? Extraido de: https://keepcoding.io/blog/que-es-el-algoritmo-des/#Que_es_el_algoritmo_DES