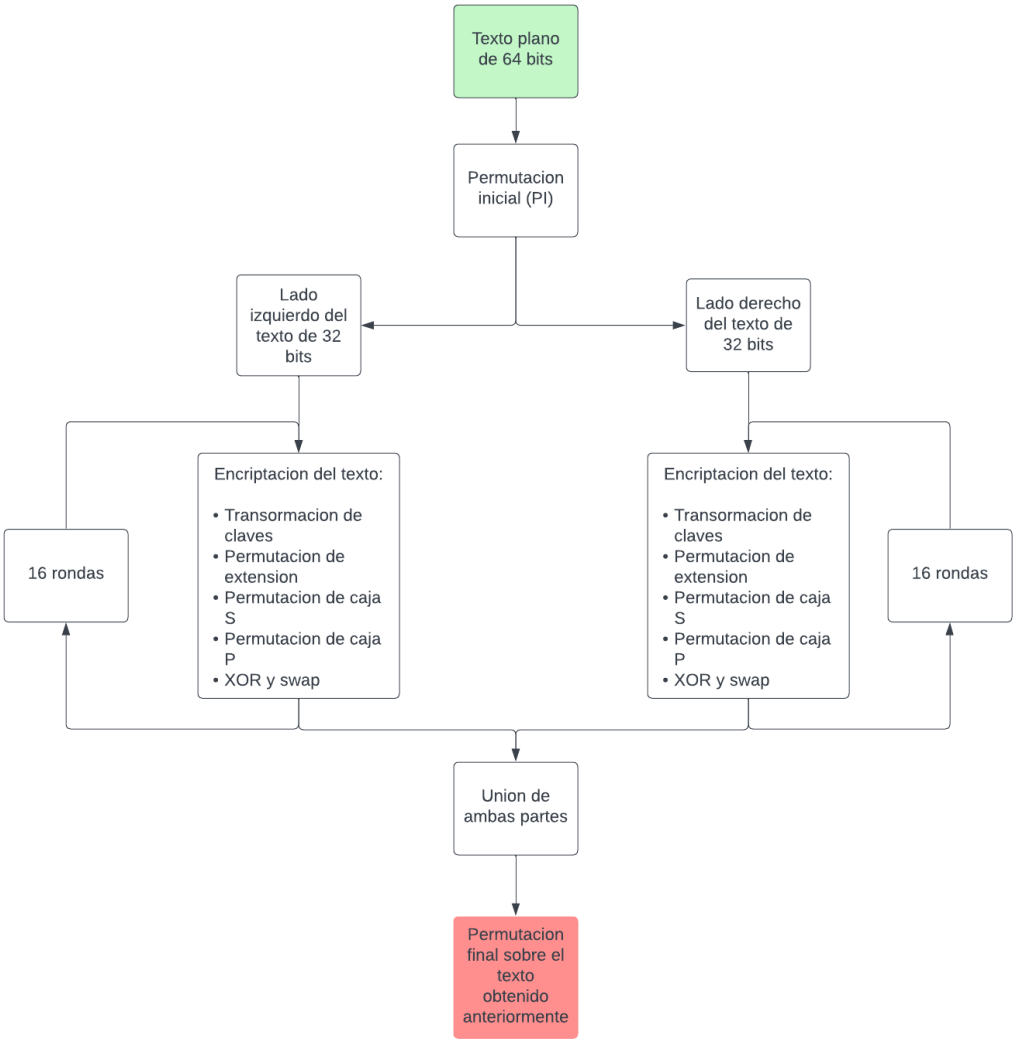
Proyecto #2

Parte A

- **Investigación sobre DES**
    - **Pasos para cifrar**
        - El mensaje se divide en bloques de 64 bits.
        - Se realiza una permutación inicial de los bloques de texto sin formato.
        - Los bloques permutados se dividen en dos mitades de 32 bits cada una: el texto en claro izquierdo (LPT) y el texto en claro derecho (RPT).
        - Tanto el LPT como el RPT se someten a 16 rondas de cifrado. Cada ronda de cifrado consta de cinco pasos:
            - Transformación de claves - La transformación de claves es un proceso en el que 16 subclaves diferentes de 48 bits cada una se derivan de la clave principal para cifrar el texto sin formato.
            - Permutación de expansión: un medio bloque de 32 bits se amplía a 48 bits mediante una permutación de expansión. Añade bits adyacentes de cada lado del bloque a los 32 bits del bloque para crear un bloque de 48 bits.
            - Permutación S-Box - Después de mezclar el bloque con la subclave, se divide en ocho partes de 6 bits. El proceso S-box utiliza una tabla de consulta para convertir las ocho partes de 6 bits en una salida de 4 bits cada una, lo que da como resultado una salida de 32 bits en total.
            - Permutación de caja P - La salida de 32 bits de la permutación de caja S se reordena según la permutación de caja P.
            - XOR y swap - XOR es una función matemática que compara dos conjuntos de bits que pueden ser 1s o 0s. Si los bits de ambos conjuntos coinciden, la salida de XOR es 0. Por el contrario, si no coinciden, la salida es 1.
        - Se combinan el LPT y el RPT.
        - La permutación final se realiza sobre el LPT y el RPT combinados, dando como resultado el texto cifrado final.
    - **Pasos para descifrar**
        - El texto cifrado se divide en bloques de 64 bits.
        - Se realiza una permutación inicial de los bloques de texto cifrado.

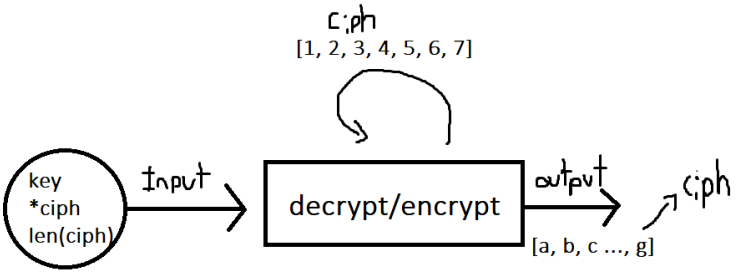
- Los bloques permutados se dividen en dos mitades de 32 bits cada una: el texto cifrado izquierdo (LCT) y el texto cifrado derecho (RCT).
- Tanto el LCT como el RCT se someten a 16 rondas de descifrado. Cada ronda de descifrado consta de cinco pasos:
    - Transformación de claves: Se utilizan las mismas subclaves de 48 bits generadas durante el proceso de cifrado.
    - Permutación de expansión: Similar al cifrado, se realiza una permutación de expansión en el RCT para obtener un bloque de 48 bits.
    - Permutación de caja S: Se aplica la permutación de caja S, pero en orden inverso, al bloque de 48 bits obtenido en el paso anterior.
    - Permutación de caja P: Se aplica la permutación de caja P al resultado de la permutación de caja S inversa.
    - XOR y swap: Se realiza una operación XOR y un intercambio entre el LCT y el RCT.
- Se combinan el LCT y el RCT.
- Se aplica la permutación final inversa sobre el LCT y el RCT combinados, obteniendo así el texto original descifrado.

- **Diagrama de flujo**



- **Diagrama de rutas**
  - **decrypt (key, *ciph, len)**
  - **encrypt (key, *ciph, len)**
  - **tryKey (key, *ciph, len)**
  - **memcpy**
  - **strstr**

	decrypt(key, *ciph, len)	**encrypt(key, *ciph, len)**
Explicación	Como parámetros, estas funciones reciben (key) la cual es la llave que se va a utilizar para desencriptar/encriptar, (ciph) que es un puntero a la dirección de memoria donde está almacenado el arreglo de caracteres y por último(len) que hace referencia a la longitud de la cadena de caracteres(ciph). El resultado obtenido de realizar la iteración, ya se encriptar/desencriptar se almacena en ciph.	

	 Dentro de estas rutinas además de recibir parámetros se declaran otras variables que permiten que la librería OpenSSL pueda realizar el cifrado/descifrado DES. DES_Key_schedule que almacena las claves que se utilizaran para realizar el cifrado/descifrado. DES_set_key_unchecked que funciona para establecer la clave en el schedule, recibiendo como parametros la llave y el schedule. Por ultimo, DES_ecb_encrypt lel cual se encarga de recibir como parámetros el arreglo de caracteres(ciph) en donde se van a almacenar(ciph en este caso) y la acción que se realizará, ya sea encriptar o desencriptar.	
	trykey(key, *ciph, len)	**memcpy**
Explicación	Recibe los mismos parámetros que las 2 funciones anteriores. Se encarga de descifrar el arreglo de caracteres con la llave y en caso esta logre descifrar el mensaje hara la comparación de que si se encuentra la cadena que hayamos definido(search) dentro de esta, si esto es correcto devolvera 1 sino 0.	Esta rutina se encarga nada más de tener un dato temporal, que en este caso sería el arreglo de caracteres ciph antes de descifrar para poder realizar las iteraciones necesarias hasta que esté descifrado.
	strstr	
Explicación	Esta rutina sirve para comprobar que una subcadena existe en una	

	cadena general, básicamente compara de que la variable de la subcadena (search) este dentro del mensaje descifrado general/principal.	

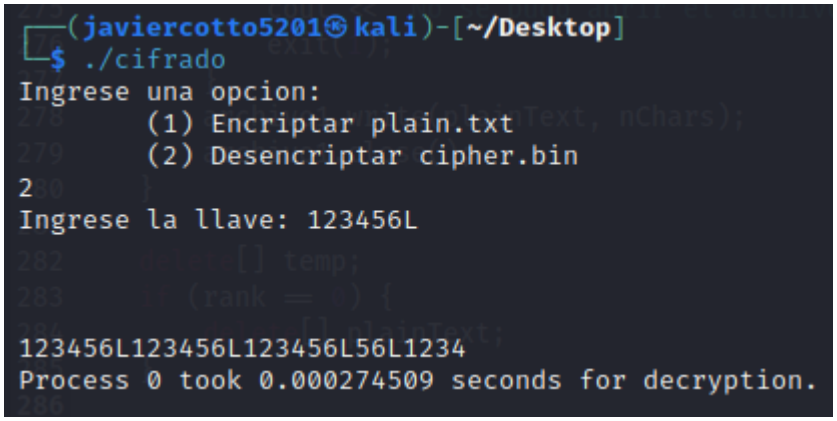
- **Explicación y flujo de comunicación de las primitivas de MPI**
  - **MPI_Irecv**
  - **MPI_Send**
  - **MPI_Wait**

	MPI_Irecv	**MPI_Send**
Explicación	Se utiliza para recibir datos de otros procesos en una comunicación punto a punto en el contexto de la biblioteca MPI (Message Passing Interface). A diferencia de la función MPI_Recv, MPI_Irecv es una operación no bloqueante. Esto significa que se inicia una operación de recepción, pero la llamada retorna inmediatamente sin esperar a que se complete la recepción de los datos.	Se utiliza para enviar datos desde un proceso emisor a un proceso receptor en una comunicación punto a punto en MPI. MPI_Send es una operación bloqueante, lo que significa que el proceso emisor se bloqueará hasta que los datos sean enviados y recibidos por el proceso receptor.
Flujo	1. Se especifica el buffer de recepción donde se almacenarán los datos recibidos. 2. Se especifica el tamaño y el tipo de los datos que se recibirán. 3. Se especifica el identificador del proceso remoto (el emisor) desde el cual se recibirán los datos. 4. Se especifica el identificador del comunicador en el que se realizará la operación de recepción.	1. Se especifica el buffer de datos que se enviará. 2. Se especifica el tamaño y el tipo de los datos que se enviarán. 3. Se especifica el identificador del proceso receptor al cual se enviarán los datos. 4. Se especifica el identificador del comunicador en el que se realizará la operación de envío.

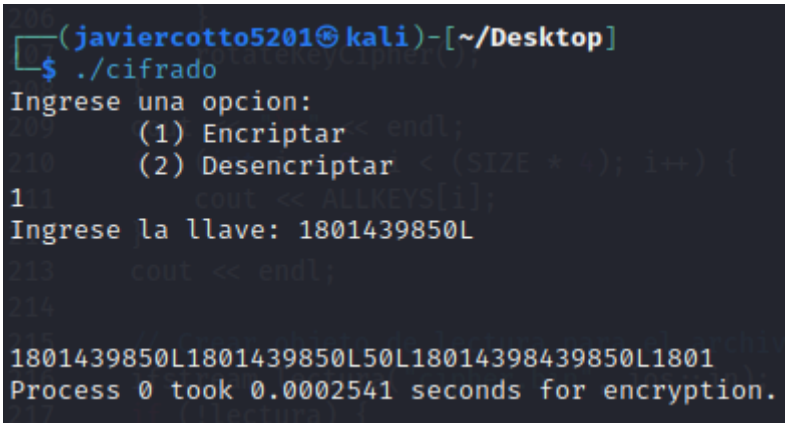
MPI_Wait
La primitiva MPI_Wait se utiliza para esperar hasta que se complete una operación de comunicación no bloqueante iniciada anteriormente, como MPI_Irecv. MPI_Wait bloqueará el proceso hasta que la operación de comunicación se haya completado.
1. Se especifica el identificador de la operación de comunicación no bloqueante que se espera que se complete. 2. El proceso se bloqueará hasta que se complete la operación de comunicación no bloqueante. Esto significa que se garantiza que los datos hayan sido recibidos o enviados correctamente antes de que el proceso continúe su ejecución.

Parte B

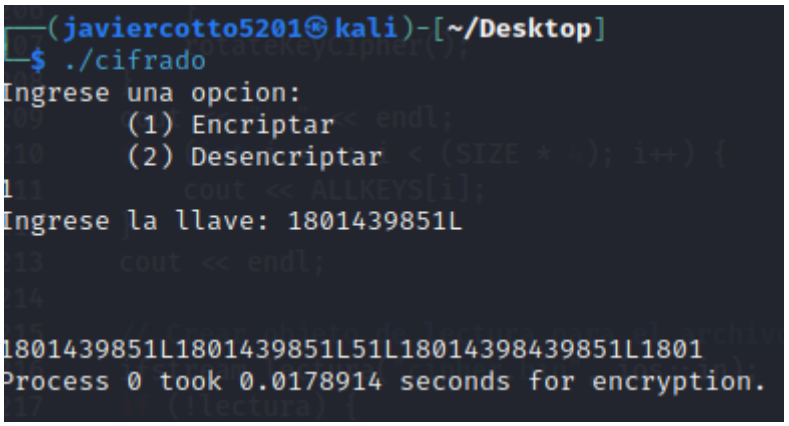
- **Cifrar un texto cargado desde un archivo**
- **4 pruebas *"Esta es una prueba de proyecto 2"***
  - **1 El caso de utilizar la llave 123456L**



  - **2**



- **3**



- **4**

El tiempo de ejecución es mayor debido a que con una llave más grande significa que esta es más difícil de encontrar por lo tanto más iteraciones y combinaciones para encontrar la llave.

- **Demostración**
  - $$E[tPar(n,k)] = \sum_{i}^{\Pi} x_i p_i = \frac{2^{55}}{n} + 1/2$$
  - Sea *x*, el tiempo esperado de ejecución del proceso *i*,
  - Sea *p*, la probabilidad de que el proceso *i* termine en x_i tiempo, dada la configuración paralela
  - Entonces el tiempo de ejecución del proceso *i* será $x_i p_i$
  - Entonces el tiempo total de ejecución será la suma del tiempo de cada uno de los procesos a ejecutar $\sum_{i}^{\Pi} x_i p_i$

d. Una llave fácil de encontrar, por ejemplo, con valor de (2^56) / 2 + 1 LLAVE 2200

Corrida	1 proceso	2 procesos	4 procesos
1	3.128	3.624	4.377
2	3.145	3.937	4.251
3	3.381	3.715	4.634
4	3.729	3.681	4.289
5	3.156	3.731	4.418

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

Corrida	1 proceso	2 procesos	4 procesos
1	6.177	7.602	3.623
2	6.167	7.362	3.571
3	6.067	7.402	3.661
4	6.229	8.078	3.594
5	6.827	7.539	3.595

Inciso 7

Para reducir el tiempo paralelo esperado en el problema planteado.

Opción 1: División y conquista

El enfoque de "división y conquista" implica dividir el problema en subproblemas más pequeños, resolverlos de manera independiente y combinar los resultados. Aquí está el pseudocódigo para esta opción:

Si n <= k, aplica el enfoque "naive" para resolver el problema directamente.
De lo contrario, divide el problema en m subproblemas más pequeños, donde m es un número adecuado de subproblemas.

Resuelve cada subproblema de manera recursiva utilizando el enfoque "división y conquista".
Combina los resultados de los subproblemas y devuelve el resultado final.
El valor esperado de tPar(n, k) para este enfoque sería el producto de los valores esperados de tPar para cada subproblema más el tiempo de combinación. El cálculo exacto del valor esperado dependerá de la naturaleza del problema en sí y cómo se realiza la división y combinación.

El speedup en este enfoque dependerá del número de subproblemas y la eficiencia de la combinación de resultados. Si la división y la combinación se realizan eficientemente, el speedup puede ser significativo en comparación con el enfoque "naive".

Opción 2: Algoritmo genético paralelo

El enfoque de algoritmos genéticos paralelos utiliza múltiples hilos o procesos para evolucionar una población de soluciones mediante operadores genéticos como mutación, recombinación y selección. Aquí hay un esquema básico para este enfoque:

Inicializa una población aleatoria de soluciones factibles.
Divide la población en m subpoblaciones.
Para cada generación:

a. En paralelo, aplica operadores genéticos (mutación, recombinación, selección) en cada subpoblación.
b. Combina las subpoblaciones en una nueva población global.

Repite el paso 3 hasta alcanzar un criterio de terminación.

El valor esperado de tPar(n, k) para este enfoque dependerá de la cantidad de generaciones necesarias para encontrar una solución óptima y la eficiencia de los operadores genéticos.

El speedup en este enfoque estará determinado por la cantidad de subpoblaciones y cómo se realizan las operaciones genéticas. Si las operaciones genéticas se pueden realizar eficientemente en paralelo, el speedup puede ser significativo en comparación con el enfoque "naive".

Para implementar estas opciones y realizar pruebas con llaves, necesitaría más información sobre el problema específico y cómo se relaciona con la ecuación proporcionada. Además, ten en cuenta que implementar algoritmos completos requeriría más detalles y ajustes específicos.