Universidad de los Andesx
Ingeniería de Sistemas y Computación
ISIS 2203 Infraestructura Computacional
Semestre 2024-20

Caso de Estudio 2 – Memoria Virtual

Objetivos

- Entender la importancia de contar con una administración "apropiada" de la memoria: considerando infraestructura de soporte, número de procesos en ejecución, demanda del recurso por proceso y decisiones para adicionar y/o remover marcos de página asignados.
- Construir un prototipo a escala del sistema de administración de memoria virtual que permita simular y evaluar el comportamiento de un proceso de acuerdo con los recursos disponibles.

Problemática:

La memoria es un recurso limitado que debe administrarse con cuidado para garantizar el avance en la ejecución de los procesos creados. En este contexto surge el concepto de memoria virtual, el cual ofrece varias ventajas: independencia de direcciones físicas, posibilidad de compartir memoria y de correr programas más grandes que la memoria física que se les asigna. Queremos comprender un poco mejor cómo varía el comportamiento de un proceso de acuerdo con la memoria que el sistema le asigna.

Su tarea en este caso es escribir un programa en Java que simule el sistema de paginación usando el algoritmo de reemplazo "Paginas no usadas recientemente". Tanebaum explica este algoritmo en su libro Sistemas Operativos Modernos, capítulo 3 – sección 3.4.2 "El algoritmo de reemplazo de páginas: no usadas recientemente" (disponible en versión electrónica en la biblioteca).

Tareas:

1. Escribir un programa que simule la administración de memoria y calcule el número de fallas de página y el porcentaje de *hits* de datos en RAM (un *hit* es la búsqueda de una página que sí está en la RAM y un *miss* es la búsqueda de una página que genera una falla de página).
2. Además, calcule tiempo con *hits* y *misses* vs. tiempo si todas las referencias estuvieran en RAM vs. tiempo si todas las referencias condujeran a fallas de página.
3. Analice los resultados y escriba el informe correspondiente.

Contexto:

Para simplificar el problema del manejo de memoria, y así poder concentrarnos en entender los retos asociados, estudiaremos las referencias de páginas generadas por un solo proceso que recupera un mensaje escondido en una imagen. Este proceso es conocido como esteganografía; a continuación, se incluye la definición de la Real Academia Española (RAE): "Técnica criptográfica que consiste en ocultar mensajes en archivos digitales."

El formato BMP de 24 bits permite almacenar imágenes guardando cada uno de sus pixeles. Para ello, se representa cada pixel de la imagen utilizando tres bytes; cada byte guarda un valor que corresponde a un color: un byte para el rojo (R), uno para el verde (G) y uno para el azul (B). Al combinar estos tres valores, se forma el color final del pixel.

En este caso particular, el mensaje se esconderá distribuido bit por bit, en el bit menos significativo de cada byte de una imagen. Usar el bit menos significativo es conveniente porque de esta manera las diferencias de color, entre la imagen original y la imagen que esconde el mensaje, son imperceptibles.

Para recuperar el mensaje original es necesario recorrer la imagen, byte por byte, y recuperar el bit menos significativo de cada byte. La secuencia de bits recuperados debe interpretarse con base en la tabla ASCII; es decir, hay que agrupar 8 bits para identificar el código ASCII de una letra.

Además, considere que los primeros 16 bytes guardan la longitud del mensaje escondido, es decir, hay que recorrer esos 16 bytes para recuperar la longitud del mensaje escondido. El recorrido de recuperación del mensaje debe empezar en el byte 17 y debe recorrer tantos bytes como corresponda a la longitud del mensaje.

Al final del enunciado encontrará el código para el manejo de la imagen.

Tenga en cuenta:

- El proceso que estudiaremos solo trabaja con archivos de imágenes almacenadas en formato bmp, con definición de colores en formato RGB con una profundidad, *bit depth*, de 24 bits.
- Por desempeño, no es conveniente usar imágenes muy grandes. Imágenes de 500 x 300 pixeles son suficientes. ¿Cuál es la longitud máxima de mensaje que puede almacenar una imagen de este tamaño?
- Los datos de entrada, es decir imágenes, mensaje y los tamaños correspondientes, pueden cambiar entre ejecuciones.
- Para calcular los tiempos de interés usaremos los siguientes tiempos de acceso:
  - Tiempo de lectura para datos que están en la RAM: 25 ns
  - Tiempo de lectura para datos que están en SWAP (resolución de una falla de página): 10 ms

El programa que debe escribir debe tener un menú alfanumérico con cuatro opciones:

- Opción 1: Generación de las referencias.
  - Esta opción recibe como parámetros (por consola) tamaño de página y nombre del archivo que guarda la imagen con el mensaje escondido.
  - Genera un archivo con la lista de referencias que el método de recuperación generaría durante su ejecución. Observe que solamente estudiaremos el comportamiento del método de recuperación, y en este método solamente las referencias a la matriz que almacena la imagen y el vector que almacena el mensaje. No consideraremos código, ni otras operaciones sobre la zona estática, heap o stack.
  - Supondremos que la matriz se almacena por filas (esto se conoce como row-major order), desde la página virtual 0 e inmediatamente a continuación se tiene el vector del mensaje.
  - Dependiendo de los tamaños de la matriz y el vector, podemos tener la matriz y el vector resultante en una sola página o la matriz y el vector distribuidos en varias páginas.
  - El archivo de referencias que se genere en esta opción debe incluir los siguientes datos: TP: tamaño de página, NF:#filas matriz imagen, NC: #columnas matriz imagen, NR:#referencias en el archivo y NP:# páginas virtuales del proceso (considerando solo lo necesario para almacenar la matriz imagen y el vector resultado). Después de estos datos deben estar las referencias generadas.

- Opción 2: Calcular datos buscados: número de fallas de página, porcentaje de hits, tiempos.
  - Esta opción recibe como parámetros (por consola): número de marcos de página y nombre del archivo de referencias.
  - Observe que este modo recibe como entrada el archivo generado por la opción anterior.
  - Calcula los datos resultantes.
  - En este caso el programa debe simular: (1) el comportamiento del proceso – es decir, cargar las referencias una por una y (2) el comportamiento del sistema de paginación, incluyendo identificar cuándo hay falla de página y tomar decisiones de reemplazo con base en el algoritmo de envejecimiento.
  - Además, el programa debe llevar el conteo de número de hits y fallas de página a medida que ocurren (solamente para lecturas y escrituras sobre la imagen y el mensaje - no consideraremos código, ni otras operaciones sobre la zona estática, heap o stack.)

- Es conveniente que desarrolle métodos adicionales para esconder un mensaje en una imagen y recuperar el mensaje posteriormente. Al final de este enunciado hay una guía que indica cómo esconder y recuperar un mensaje en una imagen. Es conveniente desarrollar estos métodos para que pueda hacer pruebas.
- Debe correr las opciones 1 y 2 en varios escenarios para calcular el número de fallas de página generados en cada escenario. El objetivo es entender cómo varía el resultado, al variar las condiciones de ejecución del método de recuperación (recuerde que usamos las opciones 1 y 2 para simular condiciones de ejecución).
- Los escenarios deben incluir: dos tamaños diferentes de imagen, cinco tamaños diferentes de mensaje escondido (100, 1000, 2000, 4000 y 8000) caracteres y 4 y 8 marcos de página. Observe que son 20 escenarios diferentes: 2 tamaños de imagen * 5 tamaños de mensaje * 2 cantidades de marcos.

Adicionalmente, en la segunda opción, el programa debe correr dos threads de forma concurrente:

- Un thread se encargará de ir actualizando el estado de la tabla de páginas y los marcos de página en RAM, de acuerdo con las referencias generadas por el proceso y el número de marcos de página asignados. Este thread debe correr cada milisegundo (en vez de pulsos de reloj usaremos milisegundos).
- El otro thread se encargará de ejecutar el algoritmo de actualización del bit R (con base en el esquema presentado por Tanenbaum). Este thread debe correr cada dos milisegundos (en vez de pulsos de reloj usaremos milisegundos).
- Para reducir la variación en los resultados no seleccione la página a reemplazar de forma aleatoria, busque en orden desde la primera entrada en su estructura de datos.
- Tenga en cuenta que los dos threads necesitarán compartir una o varias estructuras de datos por eso será necesario usar sincronización en algunos métodos (usted debe identificar cuáles).

Escriba un informe que incluya:

- Descripción del algoritmo usado para generar las referencias de página (modo uno)
- Descripción de las estructuras de datos usadas para simular el comportamiento del sistema de paginación y cómo usa dichas estructuras (cuándo se actualizan, con base en qué y en qué consiste la actualización).
- Esquema de sincronización usado. Justifique brevemente dónde es necesario usar sincronización y por qué.
- Una tabla con los datos recopilados (y porcentaje de *hits* y *misses* por cada escenario simulado).
- Una serie de gráficas que ilustren el comportamiento del sistema. Para eso muestre gráficas donde fije tamaño de página y grafique Tamaño de imagen vs. Marcos asignados vs. Porcentaje de hits. La gráfica al final del enunciado ilustra el tipo de gráfica que buscamos.
- Corra los escenarios y genere gráficas que muestren los datos recopilados para los diferentes escenarios.
- Además de los escenarios definidos, considere otras configuraciones que le permitan entender cómo afecta la memoria virtual el desempeño del programa.
- Incluya las gráficas de tiempo (hits, misses, total).
- Escriba su interpretación de los resultados: ¿corresponden a los resultados que esperaba, con respecto al número de marcos asignados? Explique su respuesta.
- ¿Si la localidad del problema manejado fuera diferente cómo variarían los resultados? Explique su respuesta. (considere una localidad mayor y una localidad menor).

Entrega:

- Cada grupo debe entregar un zip de un proyecto Java con los archivos Java con la implementación correspondiente. En el subdirectorio docs debe estar el informe en formato Word o pdf. **Al comienzo del informe, deben estar los nombres y carnés de los integrantes del grupo**. Si un integrante no aparece en el documento entregado, el grupo podrá informarlo posteriormente. Sin embargo, habrá una penalización: la calificación asignada será distribuida (dividida de forma equitativa) entre los integrantes del grupo.
- El trabajo se realiza en los grupos asignados para el caso 2. No debe haber consultas entre grupos.
- El grupo responde solidariamente por el contenido de todo el trabajo, y lo elabora conjuntamente (no es trabajo en grupo repartirse puntos o trabajos diferentes). Se puede solicitar una sustentación a cualquier miembro del grupo sobre cualquier parte del trabajo. Dicha sustentación será parte de la calificación de todos los miembros.
- El proyecto debe ser entregado en bloqueneon por uno solo de los integrantes del grupo.
- **La fecha límite de entrega es octubre 17, 2024 a las 23:50 p.m.**

Referencias:

- *Sistemas Operativos Modernos. Andrew Tanenbaum. Editorial Pearson. Edición 3, año 2009.*

Cronograma Propuesto (se propone completar las actividades a más tardar en las siguientes fechas) :

24 sept. :	Lectura del enunciado
25 sept. :	Acuerdo del contrato
27 sept.:	Arquitectura general de la solución (diagrama de clases)
8 oct.:	Estrategia de solución
14 oct. :	Implementación + pruebas
16 oct. :	Informe y entrega
17 oct. :	Realizar Coevaluación

ANEXOS - Información adicional:

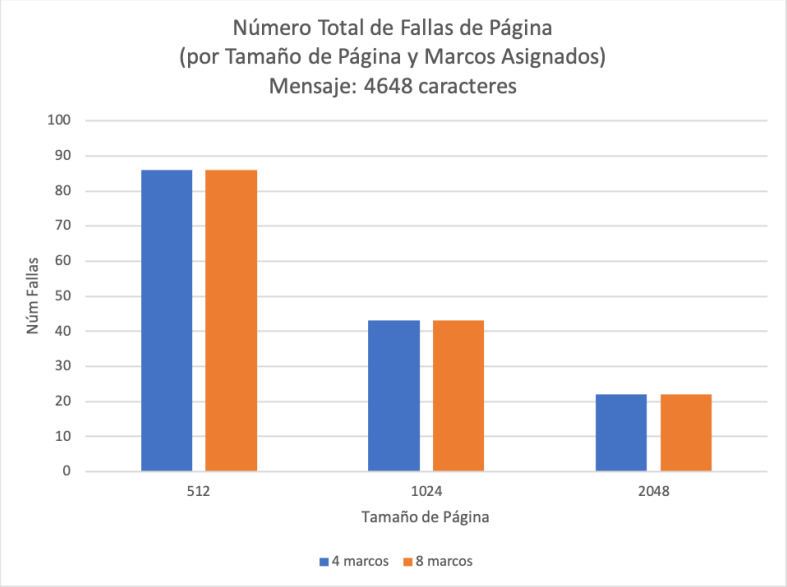
- **Anexo A.**

A continuación, se muestra el formato del archivo que debe generar el modo 1 como salida.

Datos archivo	Comentario
P=256 NF=256 NC=384 NR=86189 NP=1172 Imagen[0][0].R,0,0,R Imagen[0][0].G,0,1,R Imagen[0][0].B,0,2,R Imagen[0][1].R,0,3,R Imagen[0][1].G,0,4,R Imagen[0][1].B,0,5,R Imagen[0][2].R,0,6,R Imagen[0][2].G,0,7,R Imagen[0][2].B,0,8,R Imagen[0][3].R,0,9,R Imagen[0][3].G,0,10,R Imagen[0][3].B,0,11,R Imagen[0][4].R,0,12,R Imagen[0][4].G,0,13,R Imagen[0][4].B,0,14,R Imagen[0][5].R,0,15,R Mensaje[0],1152,0,W Imagen[0][5].G,0,16,R Mensaje[0],1152,0,W Imagen[0][5].B,0,17,R Mensaje[0],1152,0,W Imagen[0][6].R,0,18,R Mensaje[0],1152,0,W Imagen[0][6].G,0,19,R Mensaje[0],1152,0,W …	Tamaño de página (en bytes) NF y NC: Número de filas y columnas de la imagen NR: Número de referencias (en el archivo) NP: Número de páginas virtuales (las páginas necesarias para almacenar la matriz imagen y el vector resultante) Imagen: Matriz Imagen Mensaje: Vector resultante (es decir, donde se almacena el mensaje recuperado) Cada referencia tiene 4 campos (separados por ",”): • Campo 1: matriz o vector y posición correspondiente. Esta información no es estrictamente necesaria para calcular hits y fallas, pero la incluimos por claridad. • Campo 2: página virtual correspondiente • Campo 3: desplazamiento en la página virtual • Campo 4: bit de acción (R: lectura, W: escritura)

Anexo B.

A continuación, se presenta un ejemplo del tipo de gráfica que debe incluir en el informe, tenga en cuenta que esto es un ejemplo y el informe debe considerar más escenarios. También se presenta la tabla de datos, considere que, por el manejo de concurrencia, sus resultados pueden variar un poco con respecto a los valores presentados, pero el número de hits y misses debería estar en el rango +/- 5% (el conteo, no el porcentaje).



Páginas de 512, parrots_mod.bmp, mensaje: 4648 caracteres			
Marcos Asignados	Total referencias	Hits	Fallas
4	81956	81870	86
8	81956	81870	86
Páginas de 1024, parrots_mod.bmp, mensaje: 4648 caracteres			
	Total referencias	Hits	Fallas
Marcos Asignados	Total referencias	Hits	Fallas
4	81956	81913	43
8	81956	81913	43
Páginas de 2048, parrots_mod.bmp, mensaje: 4648 caracteres			
Marcos Asignados	Total referencias	Hits	Fallas
4	81956	81934	22
8	81956	81934	22

Anexo C.

El enunciado va acompañado de tres archivos:

- Un archivo de imagen original (formato BMP con profundidad de 24 bits): caso2-parrots.bmp
- El archivo bmp con el mensaje escondido: caso2-parrots_mod.bmp
- El archivo txt con el mensaje que se escondió en la imagen: caso2-mensaje_dollshousep1.txt

Usted debe usar archivos diferentes, estos se incluyen por claridad.

Anexo D.

Código para el manejo de la imagen:

```
public class Imagen {

byte[] header = new byte[54];
byte[][][]imagen;
int alto, ancho; // en pixeles
```

```
int padding;
String nombre;

/***
* Método para crear una matriz imagen a partir de un archivo.
* @param input: nombre del archivo. El formato debe ser BMP de 24 bits de bit depth
* @pos la matriz imagen tiene los valores correspondientes a la imagen
*              almacenada en el archivo.
* */
public Imagen (String input)  {

       nombre = new String(input);
       try {
              FileInputStream fis = new FileInputStream(nombre);
              fis.read(header);

              // Extraer el ancho y alto de la imagen desde la cabecera
              // Almacenados en little endian
              ancho = ((header[21] & 0xFF) << 24) | ((header[20] & 0xFF) << 16) |
                              ((header[19] & 0xFF) << 8) | (header[18] & 0xFF);
              alto = ((header[25] & 0xFF) << 24) | ((header[24] & 0xFF) << 16) |
                               ((header[23] & 0xFF) << 8) | (header[22] & 0xFF);

              System.out.println("Ancho: " + ancho + " px, Alto: " + alto + " px");
              imagen = new  byte[alto][ancho][3];

              int rowSizeSinPadding = ancho * 3;
              // El tamaño de la fila debe ser múltiplo de 4 bytes
              padding = (4 - (rowSizeSinPadding%4))%4;

              // Leer y modificar los datos de los píxeles
              // (en formato RGB, pero almacenados en orden BGR)
              byte[] pixel = new byte[3];
              for (int i = 0; i < alto; i++) {
                     for (int j = 0; j < ancho; j++) {
                            // Leer los 3 bytes del píxel (B, G, R)
                            fis.read(pixel);
                            // imagen[i][j] = new Color();
                            imagen[i][j][0] = pixel[0];
                            imagen[i][j][1] = pixel[1];
                            imagen[i][j][2] = pixel[2];
                     }
                     fis.skip(padding);
              }
              fis.close();
       } catch (IOException e) { e.printStackTrace(); }
 }

/**
* Método para esconder un valor en una matriz imagen.
* @param contador: contador de bytes escritos en la matriz
*
* @param valor: valor que se quiere esconder
* @param numbits: longitud (en bits) del valor
* @pre la matriz imagen debe haber sido inicializada con una imagen
* @pos los bits recibidos como parámetro (en valor) están escondido en la imagen.
*
*/
private void  escribirBits(int contador, int valor, int numbits) {
// la matriz tiene ancho pixel de ancho (num pixeles por fila)
// Cada pixel de la matriz corresponde a 3 Bytes.
// En cada pixel es posible esconder 3 bits (en cada byte de los componentes se esconde un bit)
         int bytesPorFila = ancho*3;// ancho de la imagen en bytes
         int mascara;
         for (int i=0; i < numbits ; i++) {
              // i: i-ésimo bit del valor que se va a esconder
              // 8*pos: por cada byte debemos esconder 8 bits
              // 8*pos + i indica el byte que se debe modificar

              int fila = (8*contador + i) / bytesPorFila;
              // Cada posición de la matriz agrupa 3 bytes (RGB).
              int col = ((8*contador + i) % bytesPorFila) / 3;
```

```
                int color = ((8*contador + i) % bytesPorFila) % 3;

                mascara = valor >> i;
                mascara = mascara & 1;
                imagen[fila][col][color] = (byte)((imagen[fila][col][color]  & 0xFE) | mascara  );
        }
    }

/**
* Método para esconder un mensaje en una matriz imagen.
* @param mensaje: Mensaje a esconder
* @param longitud: longitud del mensaje
* @pre la matriz imagen debe haber sido inicializada con una imagen
* @pre la longitud del mensaje en bits debe ser menor que el numero de pixels de la imagen * 3
* @pos la longitud del mensaje y el mensaje completo están escondidos en la imagen
*/
public  void esconder( char [] mensaje, int longitud) {
        int contador=0;
        byte elByte;
        escribirBits(contador,longitud,16) ;
        // La longitud del mensaje se esconderá en los primeros 16 bytes.
        // Eso es el equivalente a 2 caracteres (en necesidad de almacenamiento).
        // El primer byte del mensaje se almacena después de la longitud (a partir del byte 17)
        contador = 2;

        for (int i=0; i < longitud; i ++) {
                elByte =(byte) mensaje[i];
                escribirBits(contador, elByte, 8) ;
                contador++;
                if ( i % 1000  == 0)
                        System.out.println("Van " + i + " caracteres de " + longitud);
        }
}

/**
* Método para escribir una imagen a un archivo en formato BMP
* @param output: nombre del archivo donde se almacenará la imagen.
*                            Se espera que se invoque para almacenar la imagen modificada.
* @pre la matriz imagen debe haber sido inicializada con una imagen
* @pos se creó el archivo en formato bmp con la información de la matriz imagen
*/
public void  escribirImagen(String output) {
        byte pad = 0;
        try {
                FileOutputStream fos = new FileOutputStream(output);
                fos.write(header);
                byte[] pixel = new byte[3];

                for (int i = 0; i < alto; i++) {
                        for (int j  = 0; j < ancho; j++) {
                                // Leer los 3 bytes del píxel (B, G, R)
                                pixel[0] = imagen[i][j][0];
                                pixel[1] = imagen[i][j][1];
                                pixel[2] = imagen[i][j][2];
                                fos.write(pixel);
                        }
                        for (int k=0; k<padding; k++) fos.write(pad);
                }
                fos.close();
        } catch (IOException e) { e.printStackTrace(); }
}

/**
* Método para recuperar la longitud del mensaje escondido en una imagen.
*
* @pre la matriz imagen debe haber sido inicializada con una imagen.
*              la imagen debe esconder la longitud de un mensaje escondido,
*              y el mensaje.
* @return La longitud del mensaje que se esconde en la imagen.
*/
public int leerLongitud() {
        int longitud=0;
```

```
        // Usamos 16 bits para almacenar la longitud del mensaje.
        // Esos 16 bits se esconden en los primeros 16 bytes de la imagen.
        // Como cada posición de la matriz tiene 3 bytes y se esconde un bit en cada byte.
        // Debemos leer 5 pixeles completos y 1 byte del siguiente pixel.
        for (int i=0; i < 16 ; i++) {
                // ancho es el número de pixeles en una fila
                // ancho*3 es el número de bytes en una fila
                int col = (i % (ancho*3)) / 3;
                longitud = longitud | (imagen[0][col][((i % (ancho*3)) % 3)] & 1) << i ;

        }
        return longitud;
}

/**
* Método para recuperar un mensaje escondido en una imagen
* @param cadena: vector de char, con espacio ya asignado
* @param longitud: tamaño del mensaje escondido
*                      y tamaño del vector (espacio disponible para almacenar información)
* @pre la matriz imagen debe haber sido inicializada con una imagen.
*              la imagen debe esconder la longitud de un mensaje escondido,
*              y el mensaje.
* @pos cadena contiene el mensaje escondido en la imagen
*/

public void recuperar(char[] cadena, int longitud ) {
        //(imagen.ancho*3) ancho de cada fila de la matriz en bytes
        int bytesFila=ancho*3;
        for (int posCaracter = 0; posCaracter < longitud; posCaracter++) {
                cadena[posCaracter]=0;
                for (int i=0; i < 8 ; i++) {
                        // los dos primeros caracteres son la longitud. Hay que saltarlos
                        int numBytes = 16 + (posCaracter*8) + i;
                        int fila = numBytes/bytesFila;
                        int col = numBytes % (bytesFila) / 3;
                        cadena[posCaracter] =
          (char)( cadena[posCaracter] | (imagen[fila][col][( (numBytes % bytesFila) % 3 )] & 1) << i) ;

                }
        }
}

} // fin de la clase
```

El siguiente ejemplo ilustra la lectura de una imagen y cómo esconder el mensaje:

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
try {
    System.out.println("Nombre del archivo con la imagen a procesar: ");
    ruta  = br.readLine();
    imagen = new Imagen(ruta);

    System.out.println("Nombre del archivo con el mensaje a esconder: ");
    String ruta2 = br.readLine();

    int longitud = leerArchivoTexto(ruta2);
    imagen.esconder(mensaje, longitud);
    imagen.escribirImagen("salida" + ruta);
    // Ud debería poder abrir el bitmap de salida en un editor de imágenes y no debe percibir
    // ningún cambio en la imagen, pese a tener modificaciones por el mensaje que esconde
    br.close();
  } catch (Exception e) { e.printStackTrace(); }
```

El siguiente ejemplo ilustra cómo recuperar un mensaje escondido:

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
try {
    System.out.println("Nombre del archivo con el mensaje escondido: ");
    ruta = br.readLine();
```

```
    System.out.println("Nombre del archivo para almacenar el mensaje recuperado : ");
    String salida = br.readLine();
    imagen = new Imagen(ruta);

    int longitud = imagen.leerLongitud();
    mensaje = new char[longitud];
    imagen.recuperar(mensaje, longitud);
} catch (Exception e) { e.printStackTrace(); }
```