



Escuela de Ingeniería en Computación
Ingeniería en Computación
IC6600 - Principios de Sistemas Operativos

Proyecto 2

Simulación de Algoritmos de Paginación

Gerald Calderón
gecalderon@estudiantec.cr
2023125197

Óscar Obando
osobando@estudiantec.cr
2023091684

Samuel Zúñiga
sazuniga@estudiantec.cr
2023029693

San José, Costa Rica
Octubre 2025

Contents

1	Introducción	2
1.1	Algoritmos de Paginación	2
1.1.1	Óptimo	2
1.1.2	FIFO y Second Chance	3
1.1.3	LRU y MRU	3
1.1.4	Random	3
1.2	Instrucciones	3
1.2.1	Cómo utilizar el programa	4
1.3	Conclusiones	6

Chapter 1

Introducción

Para este proyecto se implementó un compresor (y decompresor) de archivos de texto utilizando el algoritmo de Huffman. Este toma un directorio con archivos de texto y los comprime en un solo archivo en binario, al descomprimir, se restaura el directorio y los archivos dentro de este. Este sistema consiste en dos partes principales, el programa que comprime y el programa que descomprime. Ambos ofrecen tres modos de ejecución: serial, concurrente y paralelo.

1.1 Algoritmos de Paginación

1.1.1 Óptimo

El algoritmo óptimo de paginación es aquel que tiene la capacidad de ver el futuro, es decir, sabe cuales páginas serán utilizadas durante toda la ejecución de la máquina para poder decidir a cual mandaar a memoria virtual. Su funcionamiento consiste en buscar dentro de las páginas cargadas en memoria real aquella que nunca se va a volver a utilizar o la que se va a utilizar más tarde para reemplazarla por la que se requiera cargar en dicho momento.

Para su implementación al leer el archivo de instrucciones a utilizar se llama a la función “parse_for_optimal” (ver figura 1.1) que detecta cada aparición de “use” o “new” para registrar las páginas en una lista que contendrá el id de las páginas que son utilizadas en dichas instrucciones. Una vez se lee todo el archivo dicha lista de páginas es pasada al constructor de la MMU concreta que contiene el algoritmo óptimo como implementación de la función “paging” (ver figura 1.2).

```

void parse_for_optimal() {
    int i;
    int ptr_size, n_pages, pid, ptr, page_to_load;
    ops.pos = 0;
    while (ops.content[ops.pos] != '\0') {
        i = 0;
        while ((c = ops.content[ops.pos++]) != '(') {
            token[i++] = c;
        }
        token[i] = '\0';
        if (strcmp(token, "new") == 0) {
            pid = parse_num();
            ptr_size = parse_num();
            n_pages = ((ptr_size % PAGE_SIZE) == 0 ? ptr_size / PAGE_SIZE
                      : ptr_size / PAGE_SIZE + 1);

            std::vector<int> pages_for_ptr;
            for (int p = 0; p < n_pages; ++p) {
                pages.push_back(page_id);
                pages_for_ptr.push_back(page_id);
                page_id++;
            }
            ptr_to_pages[ptr_id] = pages_for_ptr;
            ptr_id++;
        } else if (strcmp(token, "use") == 0) {
            ptr = parse_num();
            for (int page : ptr_to_pages[ptr]) {
                pages.push_back(page);
            }
        }
        while ((c = ops.content[ops.pos++]) != '\n' && c != '\0');
    }
    ops.pos = 0;
}

```

Figure 1.1: Ilustración de la estructura del programa

1.1.2 FIFO y Second Chance

1.1.3 LRU y MRU

1.1.4 Random

1.2 Instrucciones

Cómo instalar el programa

1. Descargue el código fuente del programa. Puede hacerlo de las dos siguientes formas:
 - Dirigirse al repositorio de GitHub mediante su navegador a través del siguiente link: https://github.com/Andres2950/PS0_PagingSimulator.git
 - Instalarlo directamente con el comando
`wget https://github.com/Andres2950/PS0_PagingSimulator/archive/refs/heads/main.zip`
2. Descomprima el archivo .zip descargado utilizando el comando `unzip`.

```

int paging() {
    int to_remove_index = -1;
    int farthest_use = -1;
    for (int i = 0; i < MEMORY_SIZE; ++i) {
        int next_use = -1;

        // se busca cuando se usara mas reciente
        for (int j = current_page + 1; j < futureRequests.size(); ++j) {
            if (futureRequests[j] == memory[i]) {
                next_use = j;
                break;
            }
        }
        if (next_use == -1)
            return i;

        // si el siguiente uso de esta pagina es mas lejano que el mas lejano
        // actual se selecciona este para usarlo
        if (next_use > farthest_use) {
            farthest_use = next_use;
            to_remove_index = i;
        }
    }
    return to_remove_index;
}
};

```

Figure 1.2: Ilustración de la estructura del programa

3. Al extraer el archivo podrá observar la estructura de organización similar a la figura 1.3.
4. Ejecute el archivo `install.sh`, asegúrese de qué tenga permisos de ejecución, puede utilizar el comando `chmod +x install.sh` en caso de que no los posea y luego ejecute de la siguiente forma `./install.sh`. Este archivo se hará cargo de la instalación del compilador `g++` necesario para compilar el código fuente. Además hará la instalación del paquete `cmake` para la ejecución del archivo “CMakeLists.txt”, dicho archivo posee las instrucciones de compilación de `SDL` para resolver todas las dependencias.
5. Note que la instalación de dichos paquetes requiere permisos de usuarios root, al ejecutar el archivo `install.sh` este se volverá a ejecutar con dichos permisos, para esto solicitará la contraseña del usuario root para tener dichos permisos de ejecución (la contraseña es totalmente invisible para el programa) y así poder descargar los paquetes.
6. La compilación de los archivos se realiza sin permisos root.

1.2.1 Cómo utilizar el programa

Una vez terminado el procedimiento de instalar el programa puede utilizar el comando `./build/app/app` en la carpeta del proyecto descargada para ejecutar el programa de simulación.

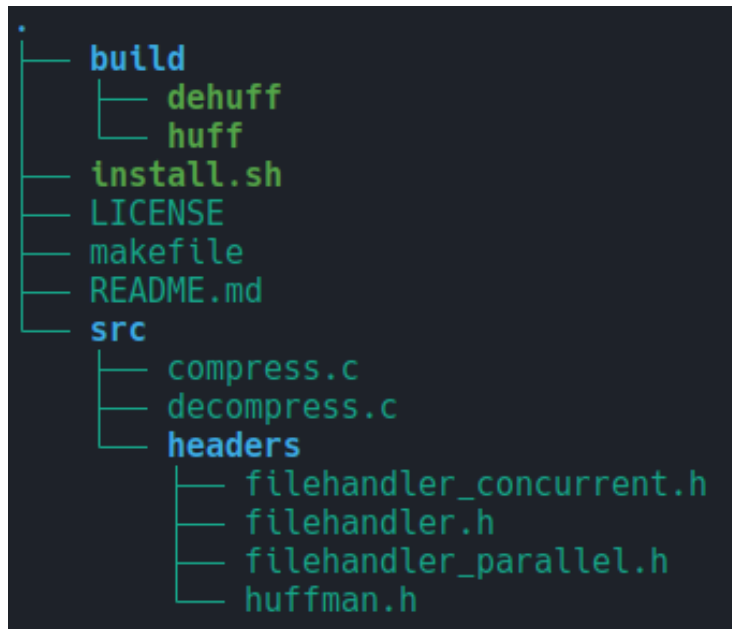


Figure 1.3: Ilustración de la estructura del programa

```
Usage: huff [options] src dst
Compress src directory into dst file.
Only handles plaintext files.

Options:
  Execution Mode:
    -s          serial
    -p[MEM_USE_FACTOR] parallel using fork
    -c          concurrent using pthread
  Benchmark:
    -b

MEM_USE_FACTOR is the coefficient used to determine the memory allocated while running the program.
MEM_USE_FACTOR is set to 2 per default. Files with unbalanced frequencies might need a higher coefficient.
```

Figure 1.4: Ilustración del menú de ayuda del comando huff -h

- **huff -h**
Provee una guía de los parámetros requeridos por el programa y su orden de entrada (Véase figura 1.4).
- **huff [Opción] src dst o dehuff [Opción] src dst**
src es el directorio objetivo a ser comprimido/descomprimido
dst es el archivo de destino
- **Opciones**
 - **-s** modo de ejecución serial.
 - **-p** modo de ejecución paralela usando fork.
 - **-c** modo de ejecución concurrente usando pthread

- **-b** modo de Benchmark para comparar los tiempos de ejecución entre los anteriores modos.

1.3 Conclusiones

Los resultados obtenidos muestran una clara mejora en los programas de compresión y de descompresión, a partir de esto se puede concluir que el algoritmo de Huffman implementado aprovecha mucho de las ventajas que proveen la concurrencia y paralelización. Esto se debe a que ambos programas manejan distintos archivos independientes entre ellos. Esta independencia permite que los archivos puedan ser manejados por separado, sea este procesamiento por medio de hilos o de procesos hijo. Por lo que se puede decir que cuando hay muchas subtarefas independientes por hacer en un programa, es bueno considerar alguno de los dos.

A pesar de lo dicho anteriormente, ambos en ambos modos de ejecución, ninguno se acercó al límite teórico que se propuso para cada uno. A partir de esto se puede concluir que el overhead que produce el uso de hilos o de subprocesos no es trivial. Al usar cualquiera de estos dos siempre es importante tomar en cuenta el overhead que estos causan para determinar si realmente vale la pena usar alguna de las técnicas descritas anteriormente.

Comparando los resultados que dieron las pruebas de concurrencia y las pruebas de paralelización, se puede concluir que la paralelización es ligeramente mejor que la concurrencia. Esto se puede deber a la naturaleza de la paralelización, gracias a que cada proceso puede correr en un procesador distinto. También se puede deber a la necesidad de la concurrencia de cambiar de contexto frecuentemente, lo que también toma recursos de la computadora.