UNIVERSIDAD DEL BÍO-BÍO

DEPARTAMENTO DE SISTEMAS DE INFORMACIÓN

Sistemas Operativos

Laboratorio 10 – Sistemas de Almacenamiento Masivo

INTRODUCCIÓN

En esta parte se explora la estructura del almacenamiento masivo, centrándose en los sistemas de almacenamiento no volátil de una computadora. El almacenamiento secundario, compuesto por discos duros mecánicos (HDD) y dispositivos de memoria no volátil (NVM), es el principal sistema de almacenamiento en las computadoras modernas.

En la teoría se detalla la estructura física de los HDD y NVM, analiza los algoritmos de planificación para optimizar el rendimiento de las operaciones de E/S, y aborda el formateo de dispositivos.

OBJETIVOS

El objetivo de este laboratorio es comprender los diferentes algoritmos de planificación de E/S en sistemas de almacenamiento masivo, implementar y experimentar con algunos algoritmos de planificación (FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK). Analizar el rendimiento de cada algoritmo mediante ejercicios prácticos.

ANTECEDENTES: ALGORITMOS DE PLANIFICACION PARA OPTIMIZACIÓN DE RENDIMIENTO

En sistemas de almacenamiento masivo, especialmente aquellos que involucran discos duros (HDD) y dispositivos de memoria no volátil (NVM), existen varios algoritmos de planificación diseñados para optimizar el rendimiento de las operaciones de entrada/salida. Estos algoritmos se encargan de determinar el orden en que se procesan las solicitudes de E/S para maximizar la eficiencia y reducir el tiempo de espera. Algunos de los algoritmos más comunes son:

First-Come, First-Served

Este algoritmo atiende las solicitudes de E/S en el orden en que llegan, es simple y fácil de implementar, pero no es eficiente en términos de tiempo de espera promedio, especialmente si las solicitudes están dispersas por toda la unidad de almacenamiento. Para ilustrar cómo funciona este algoritmo, se considera una secuencia de solicitudes de E/S a las posiciones de disco 98, 183, 37, 122, 14, 124, 65, 67. Este algoritmo atendería las solicitudes en el orden: 98, 183, 37, 122, 14, 124, 65, 67.

SCAN (Elevator Algorithm)

La cabeza del disco se mueve en una dirección (por ejemplo, de la parte más interna a la más externa) atendiendo las solicitudes en el camino, y luego invierte la dirección al llegar al final. Este algoritmo proporciona un tiempo de respuesta más uniforme y evita la inanición, pero no es óptimo en términos de tiempo de espera promedio. Para ilustrar cómo funciona este algoritmo, se considera una secuencia de solicitudes de E/S a las posiciones de disco 98, 183, 37, 122, 14, 124, 65, 67. Este algoritmo atendería en una dirección hasta la última solicitud: 65, 67, 98, 122, 124, 183, luego regresa al principio y atiende 37, 14.

C-LOOK (Circular LOOK)
Es similar a C-SCAN, pero la cabeza del disco solo se mueve hasta la última solicitud en una dirección antes de volver al principio. Proporciona una mejora adicional en términos de tiempos de respuesta uniformes, pero la complejidad de implementación puede ser mayor. Para ilustrar cómo funciona este algoritmo, se considera una secuencia de solicitudes de E/S a las posiciones de disco 98, 183, 37, 122, 14, 124, 65, 67. Este algoritmo atendería en una dirección hasta la última solicitud: 65, 67, 98, 122, 124, 183, luego regresa al principio y atiende 14, 37.

ALGORITMOS DE PLANIFICACION PARA DISCOS SSD

Los algoritmos de planificación de disco que se han discutido anteriormente están diseñados principalmente para discos duros mecánicos (HDD), donde el movimiento del cabezal de lectura/escritura y la rotación de los platos son factores críticos en el rendimiento. Sin embargo, los discos de estado sólido (SSD) funcionan de manera diferente y, por lo tanto, requieren diferentes enfoques para la gestión de E/S.
Dado que los SSD no tienen tiempos de búsqueda y latencia de rotación, los algoritmos de planificación de disco para SSD se centran en optimizar otras características, como la distribución de la carga de trabajo y la gestión del desgaste.

Wear Leveling (Nivelación de desgaste)
Es un algoritmo que distribuye las escrituras de manera uniforme entre todas las celdas de memoria flash para prolongar la vida útil del SSD. Aumenta la durabilidad del SSD, reduce la probabilidad de fallos prematuros en las celdas de memoria, pero puede introducir una sobrecarga adicional en la gestión del SSD.

ACTIVIDADES DEL LABORATORIO

1. Implementación y ejecución de Algoritmos de Planificación

Los siguientes códigos en lenguaje C, simulan la ejecución de algoritmos de planificación, se pide compilar y ejecutar utilizando la siguiente secuencia de solicitudes:

Secuencia de solicitudes: 98, 183, 37, 122, 14, 124, 65, 67
Posición inicial de la cabeza del disco: 53

FCFS

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// Función para calcular el movimiento total del cabezal y mostrar el orden de las
solicitudes
void   calcularMovimientoTotal(int   *solicitudes,   int   numSolicitudes,   int
posicionInicial) {
    int movimientoTotal = 0;
    int posicionActual = posicionInicial;

    printf("Orden de procesamiento de las solicitudes: \n");
    printf("%d -> ", posicionActual);

    for (int i = 0; i < numSolicitudes; i++) {
        movimientoTotal += abs(solicitudes[i] - posicionActual);
        posicionActual = solicitudes[i];
        if (i < numSolicitudes - 1) {
            printf("%d -> ", posicionActual);
```

```
        } else {
            printf("%d", posicionActual);
        }
    }

    printf("\nMovimiento total del cabezal: %d cilindros\n", movimientoTotal);
}

int main() {
    int posicionInicial, numSolicitudes;

    // Solicitar la posición inicial del cabezal
    printf("Ingrese la posición inicial del cabezal: ");
    scanf("%d", &posicionInicial);

    // Solicitar el número de solicitudes
    printf("Ingrese el número de solicitudes: ");
    scanf("%d", &numSolicitudes);

    int solicitudes[numSolicitudes];

    // Solicitar la secuencia de solicitudes de E/S
    printf("Ingrese la secuencia de solicitudes:\n");
    for (int i = 0; i < numSolicitudes; i++) {
        scanf("%d", &solicitudes[i]);
    }

    calcularMovimientoTotal(solicitudes, numSolicitudes, posicionInicial);

    return 0;
}
```

SCAN

```
#include <stdio.h>
#include <stdlib.h>

// Función para comparar enteros para qsort
int compare(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}

// Función para calcular el movimiento total del cabezal usando SCAN
void calcularMovimientoTotalSCAN(int *solicitudes, int numSolicitudes, int
posicionInicial, int direccion, int finDisco) {
    int movimientoTotal = 0;
    int posicionActual = posicionInicial;
    int i;

    // Ordenar las solicitudes
    qsort(solicitudes, numSolicitudes, sizeof(int), compare);

    // Encontrar el índice desde donde el cabezal debe comenzar a moverse
    int indiceInicio = 0;
    for (i = 0; i < numSolicitudes; i++) {
        if (solicitudes[i] >= posicionInicial) {
            indiceInicio = i;
            break;
        }
    }

    printf("Orden de procesamiento de las solicitudes: \n");
    printf("%d -> ", posicionActual);
```

```
    if (direccion == 1) {  // Movimiento ascendente primero
        // Procesar solicitudes en dirección ascendente
        for (i = indiceInicio; i < numSolicitudes; i++) {
            movimientoTotal += abs(solicitudes[i] - posicionActual);
            posicionActual = solicitudes[i];
            printf("%d -> ", posicionActual);
        }

        // Moverse al final del disco
        movimientoTotal += abs(finDisco - posicionActual);
        posicionActual = finDisco;

        // Procesar solicitudes en dirección descendente
        for (i = indiceInicio - 1; i >= 0; i--) {
            movimientoTotal += abs(solicitudes[i] - posicionActual);
            posicionActual = solicitudes[i];
            printf("%d -> ", posicionActual);
        }
    } else {  // Movimiento descendente primero
        // Procesar solicitudes en dirección descendente
        for (i = indiceInicio - 1; i >= 0; i--) {
            movimientoTotal += abs(solicitudes[i] - posicionActual);
            posicionActual = solicitudes[i];
            printf("%d -> ", posicionActual);
        }

        // Moverse al inicio del disco (asumimos que es 0)
        movimientoTotal += abs(0 - posicionActual);
        posicionActual = 0;

        // Procesar solicitudes en dirección ascendente
        for (i = indiceInicio; i < numSolicitudes; i++) {
            movimientoTotal += abs(solicitudes[i] - posicionActual);
            posicionActual = solicitudes[i];
            printf("%d -> ", posicionActual);
        }
    }

    printf("\nMovimiento total del cabezal: %d cilindros\n", movimientoTotal);
}

int main() {
    int posicionInicial, numSolicitudes, direccion, finDisco;

    // Solicitar la posición inicial del cabezal
    printf("Ingrese la posición inicial del cabezal: ");
    scanf("%d", &posicionInicial);

    // Solicitar la dirección inicial (1 para ascendente, 0 para descendente)
    printf("Ingrese la dirección inicial (1 para ascendente, 0 para descendente):
");
    scanf("%d", &direccion);

    // Solicitar el final del disco
    printf("Ingrese el final del disco: ");
    scanf("%d", &finDisco);

    // Solicitar el número de solicitudes
    printf("Ingrese el número de solicitudes: ");
    scanf("%d", &numSolicitudes);

    int solicitudes[numSolicitudes];

    // Solicitar la secuencia de solicitudes de E/S
    printf("Ingrese la secuencia de solicitudes:\n");
```

```
    for (int i = 0; i < numSolicitudes; i++) {
        scanf("%d", &solicitudes[i]);
    }

    calcularMovimientoTotalSCAN(solicitudes,   numSolicitudes,   posicionInicial,
direccion, finDisco);

    return 0;
}
```

C-LOOK

```
#include <stdio.h>
#include <stdlib.h>

// Función para comparar enteros para qsort
int compare(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}

// Función para calcular el movimiento total del cabezal usando C-LOOK
void  calcularMovimientoTotalCLOOK(int  *solicitudes,  int  numSolicitudes,  int
posicionInicial) {
    int movimientoTotal = 0;
    int posicionActual = posicionInicial;
    int i;

    // Ordenar las solicitudes
    qsort(solicitudes, numSolicitudes, sizeof(int), compare);

    // Encontrar el índice desde donde el cabezal debe comenzar a moverse
    int indiceInicio = 0;
    for (i = 0; i < numSolicitudes; i++) {
        if (solicitudes[i] >= posicionInicial) {
            indiceInicio = i;
            break;
        }
    }

    printf("Orden de procesamiento de las solicitudes: \n");
    printf("%d -> ", posicionActual);

    // Procesar solicitudes en dirección ascendente
    for (i = indiceInicio; i < numSolicitudes; i++) {
        movimientoTotal += abs(solicitudes[i] - posicionActual);
        posicionActual = solicitudes[i];
        printf("%d -> ", posicionActual);
    }

    // Volver a la primera solicitud no procesada
    movimientoTotal += abs(solicitudes[0] - posicionActual);
    posicionActual = solicitudes[0];

    // Procesar solicitudes en dirección ascendente desde la primera solicitud no
procesada
    for (i = 0; i < indiceInicio; i++) {
        movimientoTotal += abs(solicitudes[i] - posicionActual);
        posicionActual = solicitudes[i];
        printf("%d -> ", posicionActual);
    }

    printf("\nMovimiento total del cabezal: %d cilindros\n", movimientoTotal);
}
```

```
int main() {
    int posicionInicial, numSolicitudes;

    // Solicitar la posición inicial del cabezal
    printf("Ingrese la posición inicial del cabezal: ");
    scanf("%d", &posicionInicial);

    // Solicitar el número de solicitudes
    printf("Ingrese el número de solicitudes: ");
    scanf("%d", &numSolicitudes);

    int solicitudes[numSolicitudes];

    // Solicitar la secuencia de solicitudes de E/S
    printf("Ingrese la secuencia de solicitudes:\n");
    for (int i = 0; i < numSolicitudes; i++) {
        scanf("%d", &solicitudes[i]);
    }

    calcularMovimientoTotalCLOOK(solicitudes, numSolicitudes, posicionInicial);

    return 0;
}
```

Calcular métricas (40pts).

Registrar el movimiento total del cabezal reportado por cada algoritmo, determinar el **Tiempo Promedio de Espera** (INVESTIGAR) utilizando la formula:

Tiempo promedio de espera = Tiempo de espera acumulado/número de solicitudes

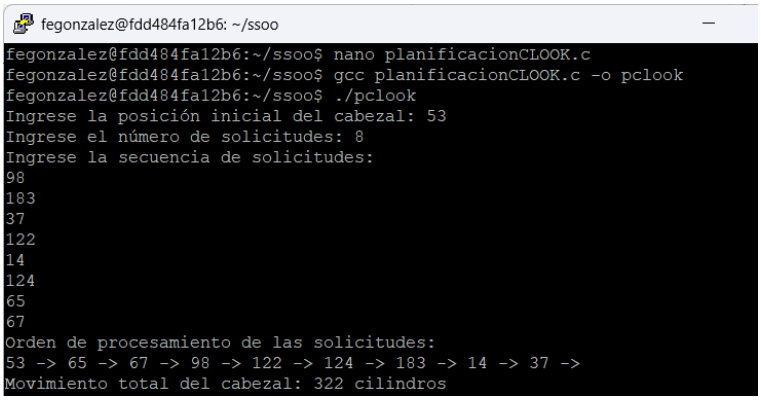
Donde el tiempo de espera acumulado se calcula como la suma del tiempo en que cada solicitud debe esperar antes de ser atendida (**completar tabla y mostrar desarrollo**).

Algoritmo	Movimiento Total del Cabezal	Tiempo Promedio de Espera
FCFS	640 cilindros	((|53–98| = **(45)** + (|98–183| = 85)) = **(130)** + (|183–37| = 146) = **(276)** + (|37–122 = 85) = **(361)** + (|122–14| = 108) = **(469)** + (|14–124| = 110) = **(579)** + (|124–65| = 59) = **(638)** + (|65–67| = 2) = **640) = *(3138 / 8) = 392,25***
SCAN	331 cilindros	((|53–65| = **(12)** + (|65–67| = 2)) = **(14)** + (|67–98| = 31) = **(45)** + (|98–122 = 24) = **(69)** + (|122–124| = 2) = **(71)** + (|124–183| = 59) = **(130)** + (|183–37| = 146) = **(276)** + (|37–14 = 23) = **299) = *(916 / 8) = 114,5***
C-LOOK	322 cilindros	((|53–65| = **(12)** + (|65–67| = 2)) = **(14)** + (|67–98| = 31) = **(45)** + (|98–122 = 24) = **(69)** + (|122–124| = 2) = **(71)** + (|124–183| = 59) = **(130)** + (|183–14| = 169) = **(299)** + (|14–37| = 23) = **322) = *(916 / 8) = 120,25***

Responde las siguientes preguntas (40pts):

- ¿Cuál algoritmo es más eficiente en términos de movimiento del cabezal?

Por lo mostrado, el algoritmo más eficiente es el de C-LOOK



- ¿Cuál algoritmo reduce el tiempo promedio de espera?

Por lo calculado, el algoritmo SCAN.

- ¿En qué escenarios sería preferible usar cada uno de los algoritmos?

FCFS para cuando hay pocas solicitudes pues es más simple.
SCAN y C-LOOK para cuando hay muchas solicitudes, aunque SCAN reduce movimientos innecesarios.

Entrega (10 pts)

Documente (screenshots) cada una de las acciones antes señaladas. Es de exclusiva responsabilidad del estudiante respetar el formato de entrega de informe de esta guía **(debajo de cada enunciado su screenshot en donde aparezca de forma clara las sentencias utilizadas)**. El formato de entrega debe ser en PDF, y el nombre del archivo debe contener su nombre y apellido (**Laboratorio_10_Nombre_Apellido**). Todas las actividades deben ser entregadas (subidas) a la plataforma digital en las fechas establecidas, por cada hora de atraso, se descontará 1 pto.