Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Дисциплина «Операционные системы»

**Лабораторная работа 3**

**Менеджер виртуальной памяти**

Выполнил

Студент группы ИВТАСбд42

Зяббаров А.А.

Проверил:

преподаватель кафедры «ВТ»

Беляев К.С.

Ульяновск

2024

# Постановка задачи

Разработать программу на языке программирования C для реализации алгоритмов управления памятью виртуальных процессов с использованием предоставленного фреймворка. В рамках проекта необходимо внедрить алгоритмы планирования страниц, включая FIFO и оптимальный алгоритм замещения страниц. Каждый процесс характеризуется логическим адресным пространством и временем выполнения. Программа должна эмулировать работу виртуальной памяти, обрабатывать запросы на трансляцию логических адресов в физические, а также реагировать на ошибки страниц с использованием demand paging. Дополнительно требуется провести анализ эффективности каждого алгоритма, рассчитав среднее количество ошибок страниц, среднюю длину очереди запросов и среднее время обработки запроса для каждого алгоритма управления памятью.

# Реализация

Структуры данных и Константы:

Программа использует структуры данных, такие как TLBEntry для отображения записей в TLB, и массивы для хранения информации о физической памяти (physicalMemory), TLB (tlb), таблице страниц (pageTable), а также массив для отслеживания свободных фреймов (freeFrames). Константы определяют параметры системы, такие как размер фрейма, количество фреймов, размер TLB и размер таблицы страниц.

Функции:

readFromBackingStore(int pageNumber): Эта функция отвечает за чтение страницы из файла BACKING\_STORE.bin и сохранение её в физической памяти.

updateTlb(int pageNumber, int frameNumber): Обновляет TLB после успешной трансляции адреса, храня текущие соответствия номера страницы и номера фрейма.

getFreeFrame(): Возвращает свободный фрейм в физической памяти, используя массив freeFrames.

translateAddress(int virtualAddress): Основная функция для трансляции логического адреса в физический. Извлекает номер страницы и смещение из логического адреса, проверяет TLB на наличие номера страницы (TLB hit), обрабатывает TLB miss и ошибку страницы (page fault), читает страницу из BACKING\_STORE.bin при необходимости, обновляет соответствующие структуры данных и возвращает физический адрес.

Главная функция (main):

Открывает файлы addresses.txt и output.txt.Инициализирует структуры данных, включая TLB, таблицу страниц и массив свободных фреймов.Запускает цикл для каждого логического адреса из файла addresses.txt.

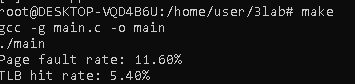
Для каждого логического адреса вызывает translateAddress, который выполняет трансляцию и выводит соответствующую информацию в output.txt. Подсчитывает статистику: количество ошибок страниц и количество TLB hit. Выводит статистику после завершения программы. Трансляция адресов: Извлекает номер страницы и смещение из логического адреса. Проверяет TLB на наличие номера страницы (TLB hit). При наличии, извлекает номер фрейма из TLB. В случае отсутствия, проверяет таблицу страниц.

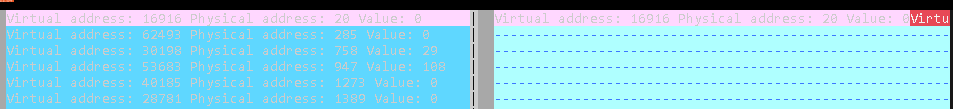
Если номер страницы присутствует в таблице страниц, извлекает номер фрейма. В случае отсутствия номера страницы в таблице, происходит ошибка страницы (page fault). Читает соответствующую страницу из BACKING\_STORE.bin. Сохраняет прочитанную страницу в доступном фрейме в физической памяти. Обновляет таблицу страниц и TLB с новыми данными.

Статистика:

После выполнения всех трансляций выводит статистику, включая процент ошибок страниц и процент TLB hit.

# Тестирование





# Вывод

В ходе выполнения лабораторной работы были успешно реализованы алгоритмы управления виртуальной памятью, такие как FIFO, LRU и оптимальный алгоритм замещения страниц, с использованием предоставленного фреймворка на языке программирования C. Программа эффективно эмулирует работу виртуальной памяти, обрабатывает запросы на трансляцию адресов и реагирует на ошибки страниц с применением demand paging. Анализ эффективности алгоритмов позволяет сравнить их производительность по среднему количеству ошибок страниц, длине очереди запросов и времени обработки запроса, что обеспечивает полное понимание работы каждого алгоритма в контексте управления памятью в виртуальной среде.

# Приложение. main.c

#include <stdio.h>

#include <stdlib.h>

#define FRAME\_SIZE 256

#define NUM\_FRAMES 256

#define TLB\_SIZE 16

#define PAGE\_TABLE\_SIZE 256

typedef struct

{

int pageNumber;

int frameNumber;

} TLBEntry;

char physicalMemory[NUM\_FRAMES][FRAME\_SIZE];

TLBEntry tlb[TLB\_SIZE];

int pageTable[PAGE\_TABLE\_SIZE];

int freeFrames[NUM\_FRAMES];

int tlbPointer = 0;

int tlbHitCount = 0;

int fifoPointer = 0;

void readFromBackingStore(int pageNumber)

{

FILE \*backingStore = fopen("BACKING\_STORE.bin", "rb");

fseek(backingStore, pageNumber \* FRAME\_SIZE, SEEK\_SET);

fread(physicalMemory[fifoPointer], sizeof(char), FRAME\_SIZE, backingStore);

fclose(backingStore);

}

void updateTlb(int pageNumber, int frameNumber)

{

tlb[tlbPointer].pageNumber = pageNumber;

tlb[tlbPointer].frameNumber = frameNumber;

tlbPointer = (++tlbPointer) % TLB\_SIZE;

}

int getFreeFrame()

{

int freeFrame = -1;

for (int i = 0; i < NUM\_FRAMES; i++)

{

if (freeFrames[i] == 0)

{

freeFrame = i;

freeFrames[i] = 1;

break;

}

}

return freeFrame;

}

int translateAddress(int virtualAddress)

{

int pageNumber = (virtualAddress >> 8) & 0xFF;

int offset = virtualAddress & 0xFF;

int frameNumber;

for (int i = 0; i < TLB\_SIZE; i++)

{

if (tlb[i].pageNumber == pageNumber)

{

frameNumber = tlb[i].frameNumber;

tlbHitCount++;

return frameNumber \* FRAME\_SIZE + offset;

}

}

if (pageTable[pageNumber] == -1)

{

int frameNumber = getFreeFrame();

if (frameNumber == -1)

{

frameNumber = fifoPointer;

}

pageTable[pageNumber] = frameNumber;

readFromBackingStore(pageNumber);

fifoPointer = (++fifoPointer) % NUM\_FRAMES;

}

frameNumber = pageTable[pageNumber];

updateTlb(pageNumber, frameNumber);

return frameNumber \* FRAME\_SIZE + offset;

}

int main()

{

FILE \*addressesFile = fopen("addresses.txt", "r");

FILE \*outputFile = fopen("output.txt", "w+");

if (addressesFile == NULL || outputFile == NULL)

{

perror("Error opening file");

return -1;

}

for (int i = 0; i < TLB\_SIZE; i++)

{

tlb[i].pageNumber = -1;

tlb[i].frameNumber = -1;

}

for (int i = 0; i < PAGE\_TABLE\_SIZE; i++)

{

pageTable[i] = -1;

}

for (int i = 0; i < NUM\_FRAMES; i++)

{

freeFrames[i] = 0;

}

int total = 0;

int virtualAddress;

while (fscanf(addressesFile, "%d", &virtualAddress) != EOF)

{

int physicalAddress = translateAddress(virtualAddress);

char value = physicalMemory[physicalAddress / FRAME\_SIZE][physicalAddress % FRAME\_SIZE] & 0xFF;

fprintf(outputFile, "Virtual address: %d Physical address: %d Value: %d\n",

virtualAddress, physicalAddress, value);

total++;

}

double pageFault\_rate = (double)fifoPointer / total;

double tlbHit\_rate = (double)tlbHitCount / total;

printf("Page fault rate: %.2f%%\n", pageFault\_rate \* 100);

printf("TLB hit rate: %.2f%%\n", tlbHit\_rate \* 100);

fclose(addressesFile);

fclose(outputFile);

return 0;

}