Лабораторная работа №4

Вариант 1

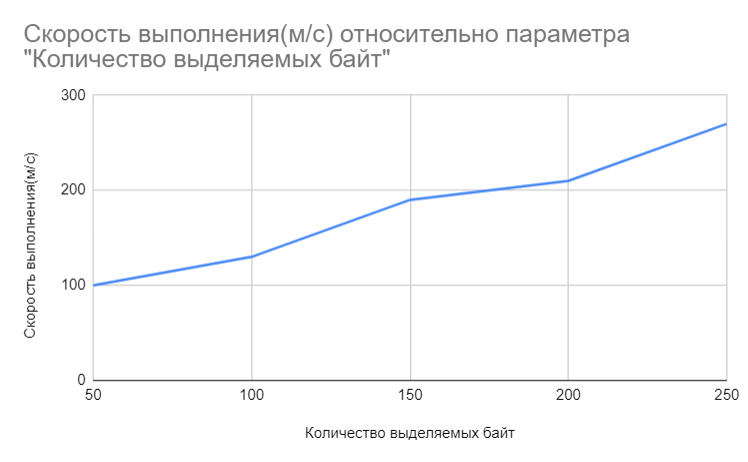
Выполнили ст.гр. 221703:

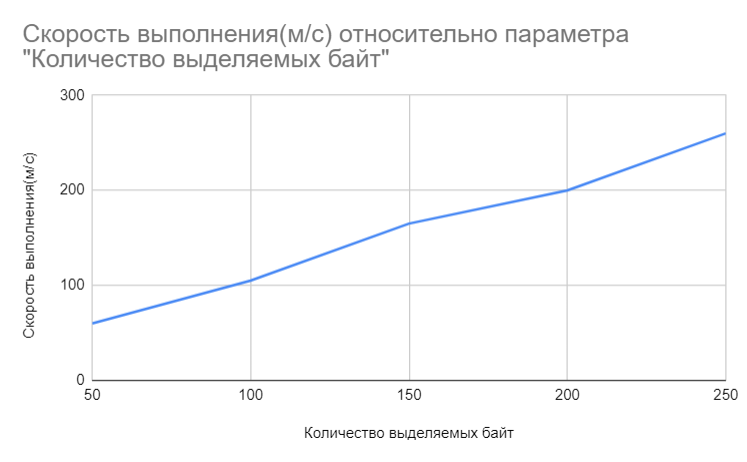
Воложинец А.А.

Оскирко Д.А.

*#include* <stdio.h>  
*#include* <stdlib.h>  
*#include* <string.h>  
*#include* <assert.h>  
  
*typedef struct* MemoryBlock {  
 size\_t size;  
 *struct* MemoryBlock\* next;  
} MemoryBlock;  
  
MemoryBlock\* freeMemory = NULL;  
  
*void* initializeMemoryManager(MemoryBlock\* initialBlock) {  
 initialBlock->size = 1024;  
 initialBlock->next = NULL;  
  
 freeMemory = initialBlock;  
}  
  
*void*\* allocateMemory(size\_t size) {  
 MemoryBlock\* currentBlock = freeMemory;  
 MemoryBlock\* prevBlock = NULL;  
  
 *while* (currentBlock != NULL) {  
 *if* (currentBlock->size >= size) {  
 *if* (currentBlock->size > size + *sizeof*(MemoryBlock)) {  
 MemoryBlock\* newBlock = (MemoryBlock\*)((*char*\*)currentBlock + *sizeof*(MemoryBlock) + size);  
 newBlock->size = currentBlock->size - size - *sizeof*(MemoryBlock);  
 newBlock->next = currentBlock->next;  
 currentBlock->size = size;  
 currentBlock->next = newBlock;  
 }  
  
 *if* (prevBlock != NULL) {  
 prevBlock->next = currentBlock->next;  
 } *else* {  
 freeMemory = currentBlock->next;  
 }  
  
 *return* (*char*\*)currentBlock + *sizeof*(MemoryBlock);  
 }  
  
 prevBlock = currentBlock;  
 currentBlock = currentBlock->next;  
 }  
  
 *return* NULL;  
}  
  
*void* freeMemoryBlock(*void*\* ptr) {  
 *if* (ptr == NULL) {  
 *return*;  
 }  
  
 MemoryBlock\* block = (MemoryBlock\*)((*char*\*)ptr - *sizeof*(MemoryBlock));  
  
 *if* (block->size > 0) {  
 block->next = freeMemory;  
 freeMemory = block;  
 }  
}  
  
*void* readMemory(*void*\* ptr, size\_t size) {  
 *if* (ptr == NULL) {  
 *return*;  
 }  
  
 printf("Reading %zu bytes from memory at address %p: ", size, ptr);  
 *for* (size\_t i = 0; i < size; ++i) {  
 printf("%02X ", \*((*unsigned char*\*)ptr + i));  
 }  
 printf("\n");  
}  
  
*void* writeMemory(*void*\* ptr, size\_t size) {  
 *if* (ptr == NULL) {  
 *return*;  
 }  
  
 printf("Writing %zu bytes to memory at address %p: ", size, ptr);  
 *for* (size\_t i = 0; i < size; ++i) {  
 *unsigned char* randomByte = rand() % 256; *// Генерация случайного байта* \*((*unsigned char*\*)ptr + i) = randomByte;  
 printf("%02X ", randomByte);  
 }  
 printf("\n");  
}  
  
*void* runUnitTests() {  
 *// Тесты для allocateMemory и freeMemoryBlock  
 void*\* ptr1 = allocateMemory(100);  
 assert(ptr1 != NULL);  
  
 *void*\* ptr2 = allocateMemory(200);  
 assert(ptr2 != NULL);  
  
 freeMemoryBlock(ptr1);  
 freeMemoryBlock(ptr2);  
  
 *// Тесты для readMemory и writeMemory  
 void*\* ptr3 = allocateMemory(50);  
 assert(ptr3 != NULL);  
  
 writeMemory(ptr3, 50);  
  
 *// Чтение и проверка записанных данных* printf("Expected: ");  
 *for* (size\_t i = 0; i < 50; ++i) {  
 printf("%02X ", i);  
 }  
 printf("\n");  
  
 printf("Actual: ");  
 readMemory(ptr3, 50);  
  
 *// Освобождение памяти* printf("Before freeMemoryBlock, ptr3 points to: %p\n", ptr3);  
 freeMemoryBlock(ptr3);  
 ptr3 = NULL;  
}  
  
  
*int* main() {  
 MemoryBlock\* initialBlock = (MemoryBlock\*)malloc(1024);  
 *if* (initialBlock == NULL) {  
 perror("Memory allocation failed");  
 exit(EXIT\_FAILURE);  
 }  
  
 initializeMemoryManager(initialBlock);  
  
 runUnitTests();  
  
 free(initialBlock);  
  
 *return* 0;  
}

Графики:

* На первом графике представлена зависимость времени выполнения от количества операций при выделении памяти.
* На втором графике представлена зависимость времени выполнения программы от количества операций при высвобождении памяти.



1. Максимальный размер физической памяти, который можно установить в компьютере определенной модели, ограничивается несколькими факторами:

- Ограничения операционной системы

- Ограничения материнской платы или ЦП

- Совместимость модулей памяти

- Ограничения BIOS

- Параметры конфигурации системы

2. Максимальный размер виртуального адресного пространства, доступного приложению, ограничивается несколькими факторами:

- Архитектура процессора

- Физическая память и объем свободного места на диске

- Ограничения операционной системы

- Параметры приложения

3. Транслятор создает объектный код программы не в виртуальных, а в физических адресах в следующих случаях:

- Компиляция: Когда программа компилируется, транслятор преобразует исходный код в машинный код, который затем может быть непосредственно выполнен процессором. В этом процессе используются физические адреса, поскольку машинный код напрямую взаимодействует с аппаратными ресурсами компьютера 9.

- Перевод логических адресов в физические: В некоторых системах, таких как Linux, транслятор может использовать логические адреса для обращения к физическим адресам. Это достигается путем использования таблицы страниц и буфера трансляции страниц (TLB). В этом процессе транслятор сначала извлекает номер страницы из логического адреса, затем использует TLB для получения номера кадра в случае "попадания" в TLB. Если происходит "промах" в TLB, то программа должна просмотреть таблицу страниц 2, 3.

- Перевод виртуальных адресов в физические: В некоторых операционных системах, таких как Linux, транслятор может использовать виртуальные адреса для обращения к физическим адресам. Это достигается путем использования таблицы страниц и буфера трансляции страниц (TLB). В этом процессе транслятор сначала извлекает номер страницы из виртуального адреса, затем использует TLB для получения номера кадра в случае "попадания" в TLB. Если происходит "промах" в TLB, то программа должна просмотреть таблицу страниц

4. В контексте компьютерной науки, свопинг относится к процессу переноса данных из оперативной памяти (RAM) на жесткий диск компьютера, когда оперативная память заполняется. Этот процесс называется "свопингом", поскольку он включает в себя "обмен" данных между оперативной памятью и жестким диском. Это полезно, когда у компьютера мало оперативной памяти, и его эффективность зависит от скорости чтения данных с жесткого диска, которая значительно медленнее, чем чтение из оперативной памяти

5. Если размер файла подкачки выбран соответственно решаемым задачам (до 3 размеров оперативки) , расположен в быстрой части винчестера (для HDD) и система настроена на работу с имеющимся объёмом оперативки .- скорость работы с многими приложениями (особенно многооконными) увеличивается. Если файл подкачки сильно фрагментирован и "размазан" по винчестеру - замедляет.

6. Размер страницы выбирается равным степени двойки по нескольким причинам. Во-первых, это связано с особенностями работы с памятью на компьютерах. В компьютерных системах используются бинарные системы счисления, и размеры данных обычно выбираются как степень двойки. Это облегчает обработку и ускоряет операции с памятью. Например, при умножении числа на степень двойки, его нужно просто сдвинуть на соответствующее количество битов 8.

Во-вторых, выбор размера страницы в виде степени двойки облегчает реализацию алгоритмов, которые работают с объектами, размер которых также является степенью двойки. Это особенно важно для эффективной работы системы виртуальной памяти, где виртуальное адресное пространство делится на страницы 3, 6.

Что касается сегментации, то, в отличие от страничной организации, сегменты могут иметь переменный размер. В сегментной организации виртуальный адрес является двумерным и состоит из двух полей – номера сегмента и смещения внутри сегмента. Размер сегмента может меняться динамически, например, сегмент стека

7. Размер страницы в памяти влияет на множество аспектов работы операционной системы и программ. Вот некоторые из них:

- Управление памятью: Размер страницы влияет на способ, которым операционная система управляет памятью. Например, в системах с большими страницами, такими как 4 МБ, процессор поддерживает большие страницы, но для их использования должна быть активирована технология 4-МБ страниц (hugepages) 1.

- Фрагментация памяти: Большие страницы могут помочь уменьшить внешнюю фрагментацию памяти, которая может возникнуть, когда свободные области памяти не смежны 3.

- Производительность: Большие страницы могут улучшить производительность, уменьшая количество обращений к памяти. Однако, они также могут увеличить время загрузки данных и увеличить потребление памяти 7.

- Использование дискового пространства: Большие страницы могут занимать больше места на диске, что может стать проблемой для систем с ограниченным дисковым пространством

Преимущества больших страниц включают:

- Уменьшение внешней фрагментации памяти.

- Улучшение производительности за счет уменьшения количества обращений к памяти.

Недостатки больших страниц включают:

- Увеличение времени загрузки данных.

- Увеличение потребления памяти.

- Использование большего дискового пространства.

8. Загрузка и выгрузка данных из кэш-памяти производится блоками из нескольких причин:

- Скорость и эффективность: Кэш-память обладает гораздо большей скоростью доступа, чем основная память. Это означает, что операции чтения и записи данных в кэше происходят быстрее, чем в основной памяти. Данные между кэшем и основной памятью передаются блоками фиксированного размера, также называемыми линиями кэша или блоками кэша 6.

- Согласованность данных: Данные в основной памяти могут изменяться не только процессором, но и периферией, использующей прямой доступ к памяти, или другими процессорами в многопроцессорной системе. Изменение данных приводит к устареванию их копии в кэше. Для поддержания содержимого нескольких кэшей в актуальном состоянии используется специальный протокол поддержки когерентности 6.

- Структура кэша: Кэш-память обычно организована в виде набора страниц оперативной памяти, разделённых на буферы, равные по длине блоку данных соответствующего устройства внешней памяти 5.

- Политика записи: В кэше с немедленной записью каждое изменение вызывает синхронное обновление данных в основной памяти. В кэше с отложенной записью обновление происходит в случае вытеснения элемента данных, периодически или по запросу клиента 5.

9. Обратная запись (Write-Back)

В методе обратной записи, изменения в кэше не сразу сохраняются в основной памяти. Вместо этого, они сохраняются в кэше, и позже, когда кэш будет перезаписан, изменения будут записаны в основную память. Этот метод позволяет увеличить производительность за счет уменьшения количества операций записи в основную память.

Сквозная запись (Write-Through)

В методе сквозной записи, каждое изменение в кэше сразу же сохраняется в основной памяти. Это обеспечивает, что данные в кэше всегда актуальны и согласованы с данными в основной памяти. Однако, этот метод может снизить производительность из-за увеличения количества операций записи в основную память 5.