МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«МИРЭА – РОССИЙСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»
РТУ МИРЭА

Институт ИКБ

Специальность (направление): 09.03.02 (информационные системы и технологии)

Кафедра: КБ-3 «Разработка программных решений и системного программирования»

Дисциплина: “Безопасность операционных систем”

Практическая работа
на тему:
Управление памятью

Студент: _____ 24.09.2024 Крашенинников М.В.
подпись *Дата* *инициалы и фамилия*

Группа: БСБО-16-23 Шифр: 23Б0107

Преподаватель: _____ 24.09.2024 Иванова И.А.
подпись *дата* *инициалы и фамилия*

Москва 2024 г.

1 Задание:

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <fstream>
#include <cstdlib>
#include <ctime>

const int NUM_PHYSICAL_FRAMES = 10;
const int PAGE_SIZE = 4;

class Disk {
public:
    void swapIn(int pageID) {
        // Simulate loading a page from disk
        std::cout << "Swapping in page " << pageID << " from disk.\n";
    }

    void swapOut(int pageID) {
        // Simulate saving a page to disk
        std::cout << "Swapping out page " << pageID << " to disk.\n";
    }
};

class PageTable {
public:
    std::unordered_map<int, int> table; // Maps virtual page numbers to physical frames or -1 for disk
    Disk* disk;

    PageTable(Disk* d) : disk(d) {}

    int getFrame(int pageID) {
        if (table.find(pageID) == table.end()) {
            disk->swapIn(pageID);
            return -1; // page not in memory
        }
        return table[pageID];
    }

    void setFrame(int pageID, int frame) {
        table[pageID] = frame;
    }
```

```cpp
class PhysicalMemory {
public:
    std::vector<int> frames;

    PhysicalMemory() {
        frames.resize(NUM_PHYSICAL_FRAMES, -1); // Initialize empty frames
    }

    int allocateFrame(int pageID) {
        for (int i = 0; i < NUM_PHYSICAL_FRAMES; ++i) {
            if (frames[i] == -1) {
                frames[i] = pageID;
                return i;
            }
        }
        return -1; // Memory full
    }

    void freeFrame(int frameID) {
        frames[frameID] = -1;
    }
};

class Process {
public:
    int pid;
    PageTable pageTable;

    Process(int id, Disk* disk) : pid(id), pageTable(disk) {}
};

class MemoryManager {
public:
    Disk disk;
    PhysicalMemory memory;
    std::vector<Process> processes;

    MemoryManager(int numProcesses) {
        for (int i = 0; i < numProcesses; ++i) {
            processes.emplace_back(i, &disk);
        }
```

```
    void accessPage(int processID, int pageID) {
        Process& process = processes[processID];
        int frame = process.pageTable.getFrame(pageID);
        if (frame == -1) {
            frame = memory.allocateFrame(pageID);
            if (frame != -1) {
                process.pageTable.setFrame(pageID, frame);
                std::cout << "Allocated frame " << frame << " to page " << pageID << " of process " << processID << "\n";
            } else {
                std::cout << "Memory full, swap needed.\n";
            }
        } else {
            std::cout << "Page " << pageID << " of process " << processID << " is in frame " << frame << "\n";
        }
    }

    void simulate() {
        srand(time(0));
        for (int i = 0; i < 10; ++i) {
            int processID = rand() % processes.size();
            int pageID = rand() % 10;
            std::cout << "Accessing page " << pageID << " for process " << processID << "\n";
            accessPage(processID, pageID);
        }
    }
};

int main() {
    MemoryManager manager(2); // Simulate 2 processes
    manager.simulate();
    return 0;
}
```

Результат:

```
twox@Matvey:/mnt/c/Users/boltf/OneDrive/Рабочий стол/Безопасность ОС/практика 4$ ./a.out
Accessing page 4 for process 1
Swapping in page 4 from disk.
Allocated frame 0 to page 4 of process 1
Accessing page 5 for process 0
Swapping in page 5 from disk.
Allocated frame 1 to page 5 of process 0
Accessing page 8 for process 1
Swapping in page 8 from disk.
Allocated frame 2 to page 8 of process 1
Accessing page 8 for process 0
Swapping in page 8 from disk.
Allocated frame 3 to page 8 of process 0
Accessing page 5 for process 0
Page 5 of process 0 is in frame 1
Accessing page 3 for process 0
Swapping in page 3 from disk.
Allocated frame 4 to page 3 of process 0
Accessing page 1 for process 1
Swapping in page 1 from disk.
Allocated frame 5 to page 1 of process 1
Accessing page 0 for process 1
Swapping in page 0 from disk.
Allocated frame 6 to page 0 of process 1
Accessing page 9 for process 1
Swapping in page 9 from disk.
Allocated frame 7 to page 9 of process 1
Accessing page 7 for process 0
Swapping in page 7 from disk.
Allocated frame 8 to page 7 of process 0
```

Задание 2:

```cpp
#include <iostream>
#include <vector>

class PhysicalMemory {
public:
    PhysicalMemory(int totalPages) : totalPages(totalPages) {
        memory.resize(totalPages, 0);
    }

    int countFreePages() const {
        int freePages = 0;
        for (int page : memory) {
            if (page == 0) {
                freePages++;
            }
        }
        return freePages;
    }

    bool allocatePage(int pageIndex) {
        if (pageIndex >= 0 && pageIndex < totalPages && memory[pageIndex] == 0) {
            memory[pageIndex] = 1;
            return true;
        }
        return false;
    }

    bool freePage(int pageIndex) {
        if (pageIndex >= 0 && pageIndex < totalPages && memory[pageIndex] == 1) {
            memory[pageIndex] = 0;
            return true;
        }
        return false;
    }

    int getTotalPages() const {
        return totalPages;
    }
private:
    int totalPages;
    std::vector<int> memory; // (0 - свободна, 1 - занята)
};

int main() {
    int totalPages;
    std::cout << "Введите общее количество страниц физической памяти: ";
    std::cin >> totalPages;

    PhysicalMemory memory(totalPages);

    memory.allocatePage(0);
    memory.allocatePage(1);

    std::cout << "Общее количество страниц: " << memory.getTotalPages() << std::endl;
    std::cout << "Количество свободных страниц: " << memory.countFreePages() << std::endl;

    memory.freePage(0);
    std::cout << "После освобождения страницы 0, количество свободных страниц: "
              << memory.countFreePages() << std::endl;

    return 0;
}
```

Результат:

```
twox@Matvey:/mnt/c/Users/boltf/OneDrive/Рабочий стол/Безопасность ОС/практика 4$ ./a.out
Введите общее количество страниц физической памяти: 13
Общее количество страниц: 13
Количество свободных страниц: 11
После освобождения страницы 0, количество свободных страниц: 12
```

Задание 3:

```
C:\Users\boltf>wmic memorychip get Manufacturer,Capacity,PartNumber,Speed,DeviceLocator
Capacity    DeviceLocator  Manufacturer  PartNumber       Speed
8589934592  DIMM 0         Samsung       K3LKCKC0BM-MGCP  5500
8589934592  DIMM 0         Samsung       K3LKCKC0BM-MGCP  5500


C:\Users\boltf> wmic memphysical get MemoryDevices
MemoryDevices
2


C:\Users\boltf>wmic memorychip get MemoryType
MemoryType
0
0
```

Вопросы:

1. Методы управления памятью. Операционная система использует несколько способов работы с памятью, чтобы эффективно распределить ресурсы и избежать их исчерпания. Один из ключевых методов — виртуализация памяти, когда операционная система позволяет программам работать с большим объемом памяти, чем есть физически в компьютере. Для этого используется свопинг, разделение памяти на страницы и сегменты. Важно отметить, что система должна поддерживать баланс между использованием оперативной памяти и дискового пространства, чтобы обеспечить быструю работу программ.

2. Принцип работы механизма свопинга. Свопинг — это процесс, при котором данные, не использующиеся в данный момент, переносятся из оперативной памяти на жесткий диск (в swap-файл или раздел). Это позволяет освободить место для других процессов, которые нуждаются в памяти. Когда данные снова становятся необходимыми, они загружаются обратно в оперативную память. Свопинг позволяет операционной системе работать с большими объемами данных, чем есть в RAM, но за счет производительности — операции с диском всегда медленнее.

3. Средства Astra Linux и ОС Windows для управления виртуальной памятью. В Astra Linux управление виртуальной памятью базируется на стандартных механизмах Linux. ОС делит память на страницы, используя swap-раздел для временного хранения данных на диске, когда оперативная память заканчивается. В Windows процесс управления виртуальной памятью похож:

система использует файл подкачки для временной записи данных. В обоих случаях ОС самостоятельно решает, когда и какие данные перемещать в swap, но в Windows пользователь может настроить параметры виртуальной памяти через панель управления.

4.

- Capacity — это объем устройства, например, объём оперативной памяти или жёсткого диска.
- DeviceLocator — показывает, в каком месте системы находится устройство, например, в каком слоте на материнской плате.
- Manufacturer — производитель устройства, то есть компания, которая его выпустила.
- PartNumber — уникальный номер или идентификатор конкретного компонента, позволяющий отличить его от других.
- Speed — скорость устройства, например, частота работы процессора или скорость передачи данных для памяти.