Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления (ИУ)»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии (ИУ7)»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Мониторинг состояния страниц, выделенных процессу»

Студент ИУ7-73Б (Группа) _____ (Подпись, дата) Р. Р. Хамзина (И. О. Фамилия)

Руководитель курсовой работы _____ (Подпись, дата) Н. Ю. Рязанова (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с заданием на курсовую работу необходимо разработать загружаемый модуль ядра для мониторинга состояния страниц, выделенных процессу.

Для решения поставленной задачи необходимо:

— проанализировать структуры и функции ядра, предоставляющие информацию о страницах, выделенных процессу;

— разработать алгоритмы и структуру программного обеспечения;

— реализовать программное обеспечение;

— провести анализ работы разработанного программного обеспечения.

1.2 Анализ управления памятью

Операционная система Linux является системой с поддержкой виртуальной памяти. Для каждого процесса создается адресное пространство, которое делится на блоки равного размера, называемые страницами. Размер страницы определяется системно. Единицей деления физической памяти является физическая страница или страничный кадр. При таком механизме организации памяти каждой виртуальной странице ставится в соответствие физический адрес.

Для отображения виртуальной памяти в физическую используются таблицы страницы. Виртуальный адрес разбивается на части: индекс в таблице страниц и смещение. Информация, хранящаяся в элементе таблицы страниц, соответствующем заданному индексу, указывает на адрес физической страницы памяти, в которой хранятся нужные данные. Схема преобразования виртуального адреса в физический адрес показана на рисунке 1.1.

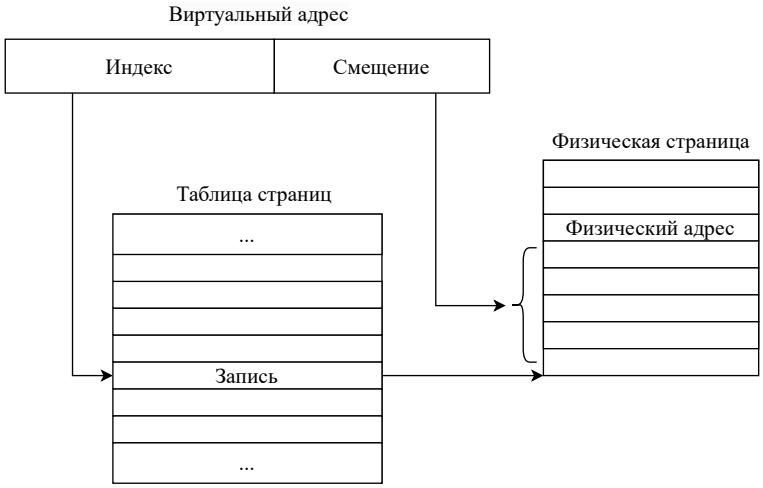

Рисунок 1.1 – Преобразование виртуального адреса в физический адрес

Для каждого процесса создается свой набор таблиц страниц. Адресное пространство процесса описывается структурой, называемой дескриптором памяти. В поле pgd дескриптора памяти хранится указатель на глобальный каталог страниц текущего процесса PGD. Элементы в таблице PGD содержат указатели на каталоги страниц среднего уровня PMD. Элементы таблиц PMD содержат указатели на таблицы PTE, записи которой указывают на страницы физической памяти. Связь таблиц страниц представлена на рисунке 1.2.

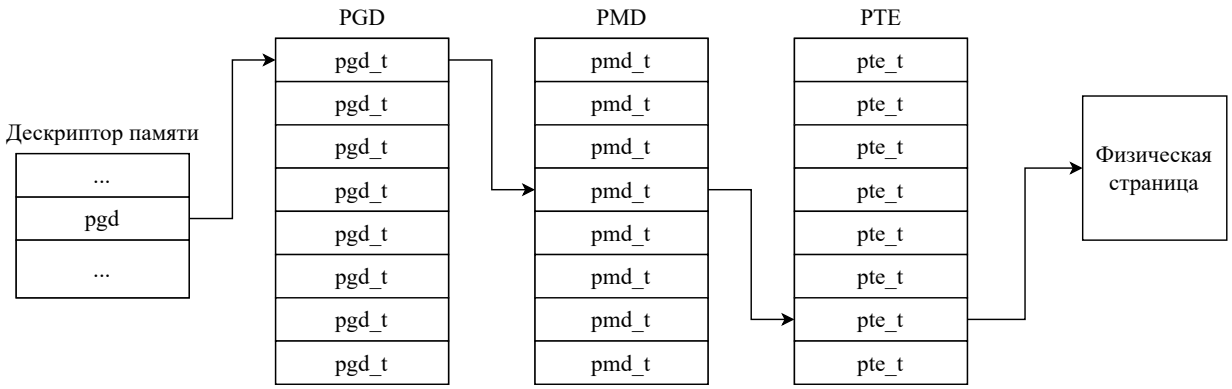

Рисунок 1.2 – Связь таблиц страниц

1.3 Анализ структур ядра, предоставляющих информацию о страницах, выделенных процессу

Для описания процесса в ядре используется структура task_struct, которая называется дескриптором процесса. Структура описана в файле <linux/sched.h>. Дескриптор процесса позволяет получить информацию о состоянии процесса, открытых файлах и другом. Некоторые поля этой структуры приведены в листинге 1.1.

Листинг 1.1 – Структура task_struct

```
struct task_struct {
    ...
    unsigned int            __state;
    ...
    unsigned int            flags;
    ...
    int             prio;
    int             static_prio;
    ...
    struct list_head        tasks;
    ...
    struct mm_struct        *mm;
    struct mm_struct        *active_mm;
    ...
    int             exit_code;
    ...
    pid_t               pid;
    pid_t               tgid;
    ...
    char                comm[TASK_COMM_LEN];
    ...
    struct fs_struct        *fs;
    struct files_struct     *files;
    ...
};
```

У объекта task_struct есть поле mm, содержащее указатель на структуру mm_struct — дескриптор памяти процесса. Эта структура описана в файле <linux/mm_types.h>. Дескриптор памяти процесса предоставляет всю информацию, относящуюся к адресному пространству процесса. Некоторые поля этой структуры показаны в листинге 1.2.

Листинг 1.2 – Структура mm_struct

```
struct mm_struct {
    struct vm_area_struct *mmap;
    struct rb_root mm_rb;
    ...
    unsigned long mmap_base;
    ...
    unsigned long task_size;
    unsigned long highest_vm_end;
    pgd_t * pgd;
    ...
    atomic_t mm_users;
    atomic_t mm_count;
    ...
    int map_count;
    spinlock_t page_table_lock;
    struct rw_semaphore mmap_lock;
    struct list_head mmlist;
    ...
    unsigned long total_vm;
    unsigned long locked_vm;
    ...
    unsigned long start_code, end_code, start_data, end_data;
    unsigned long start_brk, brk, start_stack;
    unsigned long arg_start, arg_end, env_start, env_end;
    ...
    struct mm_rss_stat rss_stat;
    ...
    unsigned long flags;
    ...
};
```

Поле mmap объекта mm_struct содержит указатель на структуру vm_area_struct, которая определена в файле <linux/mm_types.h>. Данная структура используется для описания одной непрерывной области памяти в данном адресном пространстве. Некоторые поля этой структуры представлены в листинге 1.3.

Листинг 1.3 – Структура vm_area_struct

```
struct vm_area_struct {
    unsigned long vm_start;
    unsigned long vm_end;
    struct vm_area_struct *vm_next, *vm_prev;
    ...
    struct mm_struct *vm_mm;
    ...
    unsigned long vm_flags;
```

```
    ...
    struct list_head anon_vma_chain;
    struct anon_vma *anon_vma;
    const struct vm_operations_struct *vm_ops;
    unsigned long vm_pgoff;
    struct file * vm_file;
    void * vm_private_data;
    ...
};
```

Каждый дескриптор памяти связан с уникальным диапазоном адресов в адресном пространстве процесса. В поле vm_start vm_area_struct хранится начальный адрес этого диапазона, а в поле vm_end — адрес первого байта, расположенного после описываемого диапазона.

В ядре каждая физическая страница памяти представляется в виде структуры page, которая определена в файле <linux/mm_types.h>. Некоторые поля этой структуры приведены в листинге 1.4.

Листинг 1.4 – Структура page

```
struct page {
    unsigned long flags;
    struct list_head lru;
    struct address_space *mapping;
    pgoff_t index;
    unsigned long private;
    ...
    struct list_head slab_list;
    struct page *next;
    int pages;
    int pobjects;
    ...
    void *freelist;
    ...
    atomic_t _mapcount;
    unsigned int page_type;
    unsigned int active;
    int units;
    atomic_t _refcount;
    ...
    void *virtual;
    ...
};
```

Для страниц, которые можно сопоставить с пользовательским пространством, в поле _mapcount объекта page хранится количество записей в таблице

страниц, указывающих на данную страницу.

В поле _refcount объекта page хранится счетчик использования страницы, отражающий количество ссылок в системе на эту страницу. Только что выделенная страница имеет счетчик ссылок, равный 1. Когда счетчик ссылок достигает нуля, страница освобождается.

Для определения счетчика ссылок на страницу используется функция ядра page_ref_count, определенная в <linux/page_ref.h> (листинг 1.5):

Листинг 1.5 – Определение счетчика ссылок на страницу

```
#include <linux/atomic.h>
#include <linux/mm_types.h>
#include <linux/page-flags.h>

static inline int page_ref_count(const struct page *page)
{
    return atomic_read(&page->_refcount);
}
```

1.4 Использование записей таблицы страниц

Использование записей таблицы страниц позволяет получить физическую страницу, которая представляется структурой page. Для перемещения по таблицам страниц в файле <asm/pgtable.h> определены следующие функции ядра (листинг 1.6):

Листинг 1.6 – Перемещение по таблицам страниц

```
#define pte_offset_map(dir,addr)    pte_offset_kernel((dir),(addr))
#define pte_page(pte)           (mem_map+pte_pagenr(pte))
#define pte_pagenr(pte)         ((__pte_page(pte) - PAGE_OFFSET) >>
   PAGE_SHIFT)

pgd_t *pgd_offset(const struct mm_struct *mm, unsigned long address);
pmd_t *pmd_offset(pgd_t *dir, unsigned long address);
pte_t *pte_offset_kernel(pmd_t *dir, unsigned long address);
unsigned long __pte_page(pte_t pte);
```

Доступ к физической странице осуществляется следующим образом:

1. Системный вызов pgd_offset() принимает дескриптор памяти процесса и виртуальный адрес и возвращает элемент глобального каталога страниц PGD текущего процесса.

2. Системный вызов pmd_offset() запись глобального каталога страниц

PGD и виртуальный адрес и возвращает запись каталога страниц среднего уровня PMD.

3. Системный вызов pte_offset_map() принимает элемент каталога страниц среднего уровня PMD и виртуальный адрес и возвращает запись таблицы PTE.

4. Системный вызов pte_page() по полученной записи таблицы PTE возвращает объект page.

Функции ядра, представленные в листинге 1.7, используются для определения того, существуют ли соответствующие записи таблицы страниц. В случае отстутсвия записи в таблице возвращается 1.

Листинг 1.7 – Определения счетчика ссылок на страницу

```
1 int pgd_none(pgd_t pgd);
2 int pmd_none(pmd_t pmd);
3 int pte_none(pte_t pte);
```

Функции ядра, показанные в листинге 1.8, используются для проверки записей при передаче в качестве входных параметров функциям, которые могут изменить значение записей.

Листинг 1.8 – Определения счетчика ссылок на страницу

```
1 int pgd_bad(pgd_t pgd);
2 int pmd_bad(pmd_t pmd);
```

1.5 Загружаемый модуль ядра

Ядро операционной системы Linux является монолитным. Это означает, что оно выполняется в общем защищенном адресном пространстве. При реализации микроядра в привилегированном режиме работают только те службы, которым абсолютно необходим привилегированный режим, остальные работают в непривилегированном режиме.

При этом ядро Linux является динамически изменяемым. Это означает, что существует возможность вносить необходимую разработчику функциональность без компиляции ядра. Часть кода, которая может быть добавлена в ядро во время работы, называется модулем ядра. Поддержка модулей позволяет создать минимальное базовое ядро операционной системы, а все допол-

нительные возможности и драйверы скомпилировать в качестве загружаемых модулей, т.е. самостоятельных объектов.

Вывод

2 Конструкторский раздел

2.1 Диаграмма состояний IDEF0

TODO

2.2 Схемы алгоритмов

TODO

2.3 Структура программного обеспечения

TODO

3 Технологический раздел

3.1 Выбор языка и среды программирования

TODO

3.2 Реализация загружаемого модуля ядра

TODO

4 Исследовательский раздел

ЗАКЛЮЧЕНИЕ