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**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

Курсовая работа по операционным системам

Студент Ковель Александр Денисович

Группа ИУ7-76Б

Предмет ОС

Студент _____ **Ковель А. Д.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Рязанова Н. Ю.**
подпись, дата *фамилия, и.о.*

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы — разработка загружаемого модуль ядра, вычисляющий хеш очередей сообщений с использованием API ядра.

Для достижения поставленной цели, необходимо решить следующие задачи:

1) определить структуры, связанные с поставленной задачей;
2) проанализировать и выбрать алгоритм шифрования, встроенного в ядро;
3) разработать алгоритм шифрования и расшиврования очередей сообщений;
4) реализовать программное обеспечение;
5) провести исследование скорости работы очередей сообщений с шифрованием и без.

1 Аналитические раздел

Перед тем как приступать к описанию очереди работ, необходимо описать прерывания.

1.1 Прерывания

Прерывания делятся на:

- исключения (деление на ноль, переполнение стека), данные тип прерывания является синхронным;
- системные вызовы (программные) — вызываются с помощью команды из программы (int 21h), также являются синхронными;
- аппаратные прерывания (прерывания от системного таймера, клавиатуры), эти же прерывания — асинхронные.

Прерывания разделяют на две группы: быстрые и медленные.

Для сокращения времени обработки медленных прерываний, они делятся на 2 части:

1) «top half» — верхняя половина, запускается в результате получения процессором сигнала прерывания;
2) «bottom half» — нижняя половина, отложенные вызовы.

Существуют несколько способов реализации «bottom half» обработчиков:

1) softirq;
2) tasklet (тасклеты);
3) workqueue (очереди работы).

1.2 Обработчики аппаратных прерываний

Обработчик аппаратного прерывания призван минимизировать объем необходимых действий и обеспечить как можно более быструю завершаемость. В типичном сценарии, указанный обработчик прерывания осуществляет сохранение полученных данных от внешнего устройства в ядерном буфере. Однако, с целью полноценной обработки прерываний, обработчик аппаратного прерывания должен инициировать помещение отложенного действия в очередь для

его последующего выполнения.

Обработчики аппаратных прерываний представляют собой особые функции, которые вызываются операционной системой в ответ на возникновение прерывания от аппаратного устройства. Когда аппаратное в устройстве возникает прерывание (например, сигнализирует о завершении операции или возникновении ошибки), процессор прерывает текущее выполнение и передает управление на соответствующий обработчик прерывания.

Одной из основных задач обработчика аппаратного прерывания является сохранение состояния системы, осуществление необходимых операций для обработки прерывания и восстановление исходного состояния после завершения обработки. Обработчик может выполнять различные операции, такие как чтение данных из устройства, запись данных в память, обновление регистров и установка флагов. Кроме того, обработчик аппаратного прерывания может взаимодействовать с другими частями операционной системы, например, планировщиком задач, для оптимального распределения ресурсов и обработки прерываний в системе.

1.3 Очереди работ

Очереди работ представляют собой универсальный инструмент для отложенного выполнения операций, который позволяет функции обработчика блокироваться во время выполнения соответствующих действий. Очередь работ позволяет обработчику аппаратного прерывания добавлять необходимые операции для выполнения в очередь, вместо того чтобы выполнять их прямо внутри обработчика. Таким образом, функции обработчика могут блокироваться, ждать завершения определенных операций или условий, и продолжать работу только после их выполнения.

Листинг 1 – Структура workqueue_struct

```
struct workqueue_struct {
  struct list_head  pwqs;   /* WR: all pwqs of this wq */
  struct list_head  list;   /* PR: list of all workqueues */
```

```
  struct mutex    mutex;    /* protects this wq */
  int     work_color; /* WQ: current work color */
  int     flush_color;  /* WQ: current flush color */
  atomic_t    nr_pwqs_to_flush; /* flush in progress */
  struct wq_flusher *first_flusher; /* WQ: first flusher */
  struct list_head  flusher_queue;  /* WQ: flush waiters */
  struct list_head  flusher_overflow; /* WQ: flush overflow list */
  struct list_head  maydays;  /* MD: pwqs requesting rescue */
  struct worker   *rescuer; /* MD: rescue worker */

  int     nr_drainers;  /* WQ: drain in progress */
  int     saved_max_active; /* WQ: saved pwq max_active */

  struct workqueue_attrs  *unbound_attrs; /* PW: only for unbound wqs */
  struct pool_workqueue *dfl_pwq; /* PW: only for unbound wqs */

#ifdef CONFIG_SYSFS
  struct wq_device  *wq_dev;  /* I: for sysfs interface */
#endif
#ifdef CONFIG_LOCKDEP
  char      *lock_name;
  struct lock_class_key key;
  struct lockdep_map  lockdep_map;
#endif
  char      name[WQ_NAME_LEN]; /* I: workqueue name */

  /*
   * Destruction of workqueue_struct is RCU protected to allow walking
   * the workqueues list without grabbing wq_pool_mutex.
   * This is used to dump all workqueues from sysrq.
   */
  struct rcu_head   rcu;

  /* hot fields used during command issue, aligned to cacheline */
  unsigned int    flags ____cacheline_aligned; /* WQ: WQ_* flags */
  struct pool_workqueue __percpu *cpu_pwqs; /* I: per-cpu pwqs */
  struct pool_workqueue __rcu *numa_pwq_tbl[]; /* PWR: unbound pwqs indexed by
     node */
};
```

Листинг 2 – Структура work_struct

```
struct work_struct {
  atomic_long_t data;
  struct list_head entry;
  work_func_t func;
#ifdef CONFIG_LOCKDEP
  struct lockdep_map lockdep_map;
#endif
};
```

Работа может инициализироваться двумя способами:

- статически — DECLARE_WORK(name, void func), где: name — имя структуры work_struct, func — функция, которая вызывается из workqueue — обработчик «bottom half»;
- динамически — INIT_WORK(struct work_struct *work, void func).

После того, как будет инициализирована структура для объекта work, следующим шагом будет помещение этой структуры в очередь работ. Этом можно сделать несколькими способами. Во-первых, можно добавить работу (объект work) в очередь работ с помощью функции queue_work, которая назначает работу текущему процессу. Во-вторых, можно с помощью функции queue_work_on указать процессор, на котором будет выполняться обработчик.

1.4 Шифрование

В ядре существуют специальные функции, отвечающие за регистрацию алгоритмов шифрования.

Листинг 3 – Регистрация алгоритмов шифрования.

```
/* include/linux/crypto.h */

int crypto_register_alg(struct crypto_alg *alg);
int crypto_register_algs(struct crypto_alg *algs, int count);

int crypto_unregister_alg(struct crypto_alg *alg);
int crypto_unregister_algs(struct crypto_alg *algs, int count);
```

Эти функции возвращают отрицательное значение в случае ошибки, и 0 — в случае успешного завершения, а регистрируемые алгоритмы описываются структурой crypto_alg.

Листинг 4 – Структура crypto_alg

```
/* include/linux/crypto.h */

struct crypto_alg {
    struct list_head cra_list;
    struct list_head cra_users;

    u32 cra_flags;
    unsigned int cra_blocksize;
    unsigned int cra_ctxsize;
    unsigned int cra_alignmask;

    int cra_priority;
    atomic_t cra_refcnt;

    char cra_name[CRYPTO_MAX_ALG_NAME];
    char cra_driver_name[CRYPTO_MAX_ALG_NAME];

    const struct crypto_type *cra_type;

    union {
        struct ablkcipher_alg ablkcipher;
        struct blkcipher_alg blkcipher;
        struct cipher_alg cipher;
        struct compress_alg compress;
    } cra_u;

    int (*cra_init)(struct crypto_tfm *tfm);
    void (*cra_exit)(struct crypto_tfm *tfm);
    void (*cra_destroy)(struct crypto_alg *alg);

    struct module *cra_module;
} CRYPTO_MINALIGN_ATTR;
```

- cra_flags: набор флагов, описывающих алгоритм.

- cra_blocksize: байтовый размер блока алгоритма. Все типы преобразований, кроме хэширования, возвращают ошибку при попытке обработать данные, размер которых меньше этого значения.
- cra_ctxsize: байтовый размер криптоконтекста. Ядро использует это значение при выделении памяти под контекст.
- cra_alignmask: маска выравнивания для входных и выходных данных. Буферы для входных и выходных данных алгоритма должны быть выровнены по этой маске.
- cra_priority: приоритет данной реализации алгоритма. Если в ядре зарегистрировано больше одного преобразования с одинаковым cra_name, то, при обращении по этому имени, будет возвращён алгоритм с наибольшим приоритетом.
- cra_name: название алгоритма. Ядро использует это поле для поиска реализаций.
- cra_driver_name: уникальное имя реализации алгоритма.
- cra_type: тип криптопреобразования.
- cra_u: реализация алгоритма.
- cra_init: функция инициализации экземпляра преобразования. Эта функция вызывается единожды, во время создания экземпляра (сразу после выделения памяти под криптоконтекст).
- cra_exit: деинициализация экземпляра преобразования.

Но crypt_alg является неполной структурой и сейчас используется усовершенствованный вариант struct skcipher_alg. Ее также необходимо зарегистрировать как и crypto_alg.

Листинг 5 – Структура skcipher_alg

```
    /* include/crypto/skcipher.h */

struct skcipher_alg {
    int (*setkey)(struct crypto_skcipher *tfm, const u8 *key,
```

```
                    unsigned int keylen);
    int (*encrypt)(struct skcipher_request *req);
    int (*decrypt)(struct skcipher_request *req);
    int (*init)(struct crypto_skcipher *tfm);
    void (*exit)(struct crypto_skcipher *tfm);

    unsigned int min_keysize;
    unsigned int max_keysize;
    unsigned int ivsize;
    unsigned int chunksize;
    unsigned int walksize;

    struct crypto_alg base;
};
```

В этой структуре используется crypto_alg, описывающий алгоритм. Из важных полей тут:

- chunksize: данное поле отвечает за размер блока шифрования (если этот блок не относится к поточным).
- walksize: равен значению chunksize, за исключением случаев, когда алгоритм может параллельно обрабатывать несколько блоков, тогда walksize может быть больше, чем chunksize, но обязательно должен быть кратен ему.

Также в в функциях encrypt и decrypt присутствует структура skcipher_request. Данная структура содержит данные, необходимые для выполнения операции симметричного шифрования.

В Crypto API есть еще некоторые особенности. Например, все алгоритмы шифрования данных произвольной длины работают со входными данными не через указатели на байтовые массивы, а через структуру scatterlist.

Листинг 6 – Структура skcipher_alg

```
    /* include/linux/scatterlist.h */

    struct scatterlist {
        /* ... */
```

```
        unsigned long   page_link;
        unsigned int    offset;
        unsigned int    length;
        /* ... */
    };
```

Экземпляр этой структуры можно проинициализировать указателем на некоторые данные. Например, при помощи вызова функции sg_init_one. В этой функции определяется страница памяти, с которой «начинается» buf (page_link), и определяется смещение указателя buf относительно адреса начала страницы (offset). Таким образом, криптографическая подсистема работает напрямую со страницами памяти.

Вывод

Были рассмотрены основополагающие материалы, которые в дальнейшем потребуются при реализации загружаемого модуля ядра. Данные материалы помогут при реализации обработчика прерываний и его шифрования.

2 Конструкторская часть

3 Технологическая часть

3.1 Средства реализации

Для реализации ПО был выбран язык C [1]. В данном языке есть все требующиеся инструменты для данной лабораторной работы. В качестве среды разработки была выбрана среда VS code [2].

4 Исследовательская часть

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были выполнены следующие задачи:

1)

Поставленная цель достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Язык программирования C [Электронный ресурс]. Режим доступа: https://www.youtube.com/watch?v=mj8zoGEdhhM. дата обращения: 12.11.2023.

2. Vscode [Электронный ресурс]. https://code.visualstudio.com/. дата обращения: 12.11.2023.