

Министерство образования и науки Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого

—
Институт кибербезопасности и защиты информации

ЛАБОРАТОРНАЯ РАБОТА №1

«Введение. Системный таймер.»
по дисциплине «Операционные системы»

Выполнил
студент гр. 4851003/10002

<подпись>

Галкин К. К.

Руководитель
К. Т. Н.

<подпись>

Крундышев В.М.

Санкт-Петербург
2022

1. ЦЕЛЬ РАБОТЫ

Цель работы – изучение системы управления процессами, а также механизма работы системного таймера в ОС Pintos, анализ его недостатков и модификация его алгоритма.

2. ХОД РАБОТЫ

Для понимания сути лабораторной работы и возможных недостатков нужно проанализировать файлы таймера(timer.c и timer.h) и файлы потока(thread.c и thread.h)

void timer_init (void)	Функция инициализации работы системного программируемого таймера, которая внутри себя производит конфигурацию работы каналов системного таймера. Описание этих каналов находится в файлах pit.h и pit.c
void timer_calibrate (void)	Находит максимальное значение переменной loops_per_tics, которая показывает количество действий, выполняемых перед прерыванием.
int64_t timer_ticks (void)	Возвращает точное число тиков после запуска ОС
int64_t timer_elapsed (int64_t then)	Возвращает значение тиков от какой-то точки отсчета then.
void timer_sleep (int64_t ticks)	Заставляет процесс остановить работу на заданное количество тиков ticks На вход получает число тиков.
void timer_msleep (int64_t milliseconds) void timer_usleep (int64_t microseconds) void timer_nsleep (int64_t nanoseconds)	Функции, которые усыпляют процесс на заданное количество миллисекунд/микросекунд/наносекунд
void timer_mdelay (int64_t milliseconds) void timer_udelay (int64_t microseconds) void timer_ndelay (int64_t nanoseconds)	Функции активного ожидания, вызывающая функцию real_time_delay(), принимающей на вход единицы измерения времени и само значение для сна. Используются при выключенных прерываниях с вызовом функции активного ожидания, что напрасно использует ресурсы CPU. Рекомендуется использовать функции timer_msleep, timer_usleep, timer_nsleep вместо них, если прерывания включены.
void timer_print_stats (void)	Выводит статистику системного таймера

Таблица 1. Описание функций заголовочного файла timer.h

Изначально системный таймер основан на механизме активного ожидания. Блок-схема алгоритма представлена ниже:

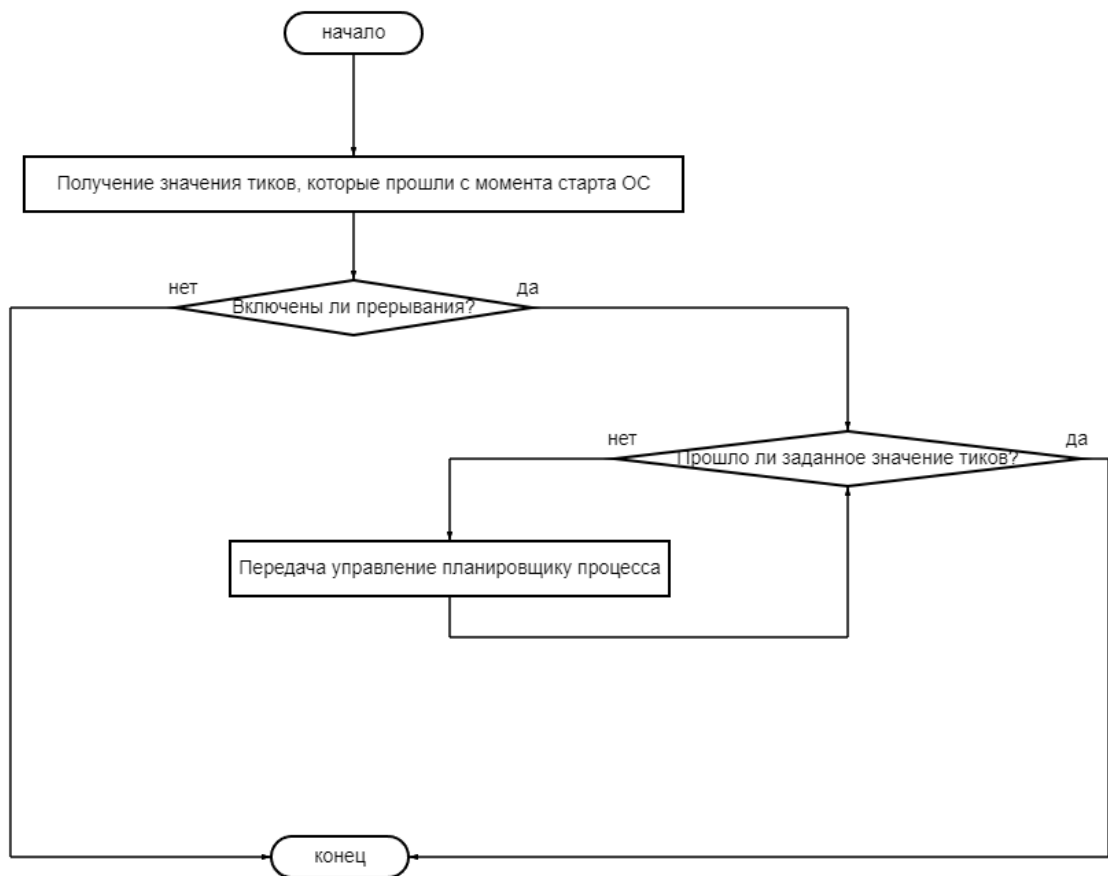


Рисунок 1. Блок-схема системного таймера

Как можно заметить и как было сказано, такой механизм работы использует ресурсы не особо оптимально. Поэтому эта функция подлежит переработке. Для этого выделим следующие этапы оптимизации:

- Создадим массив структур `array_elem`, состоящих из двух полей: указатель на поток, который должен уснуть и значение тиков, когда поток должен быть разбужен.
- Глобально объявим длину массива, чтобы было легче изменять в новых функциях
- Объявим и распишем новые функции взаимодействия с массивом:
 - `insert` – функция, которая будет добавлять новый поток в массив и сортировать массив по убыванию значения тиков в случае необходимости

- `sort_array` – функция, производящая сортировку пузырьком массива потоков. Конечно, можно было бы взять другой, более быстрый алгоритм сортировки, но в рамках лабораторной работы это не столько существенно.
- `pop` – функция, вызываемая после инкрементации текущего значения тиков в функции `timer_interrupt`. В ней проверяется, есть ли в конце массива такие потоки, которым пора просыпаться. Если есть, тогда вызывается функция `thread_unblock`, куда передается указатель на поток из элемента массива. Сами значения элемента массива зануляются, длина массива уменьшается. На данном этапе возникала проблема с использованием функции `free()` или `realloc()` на последний элемент массива, так как с их использованием была ошибка `Page-Fault`.
- Изменим функцию `timer_sleep()`, в которой, для начала, будет проверяться значение тиков, поданное на вход функции. Если оно отрицательное, то можно дальше ничего не делать и спокойно выйти из функции. Если это условие прошло, тогда подготавливается новый элемент массива `sleep_thread`, поток добавляется в массив через функцию `insert()`, блокируется через функцию `thread_block()`.
- Изменим функцию `timer_interrupt`: добавлен вызов функции `pop()`, принцип работы которой описан ранее.

Таким образом, новая блок-схема функции `timer_sleep()` выглядит следующим образом:

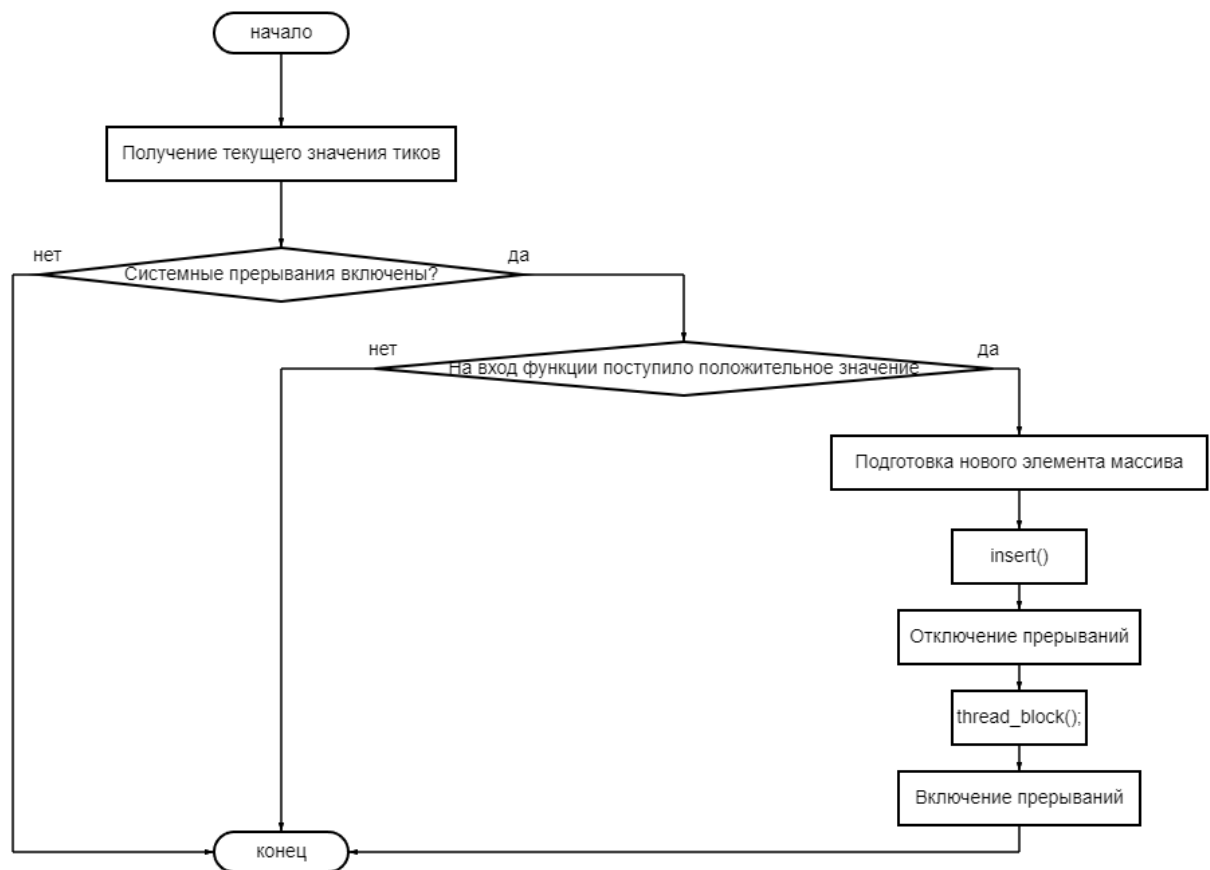


Рисунок 2. Модифицированная функция timer_sleep()

Изменения в коде написаны ниже:

- Timer.c:

```

void sort_array()
{
    struct array_elem tmp;
    for (int64_t i = 0; i < sleep_array_len - 1; i++)
    {
        for (int64_t j = i + 1; j < sleep_array_len; j++)
        {
            if (array[i].ticks < array[j].ticks)
            {
                tmp = array[j];
                array[j] = array[i];
                array[i] = tmp;
            }
        }
    }
}
  
```

```

    }
}
}
}

```

```

void insert(struct array_elem element)

```

```

{
    sleep_array_len++;
    switch (sleep_array_len)
    {
    case 1:
        array = (struct array_elem *)malloc(sizeof(struct array_elem) * sleep_array_len
);
        array[sleep_array_len - 1].thrd = element.thrd;
        array[sleep_array_len - 1].ticks = element.ticks;
        break;
    default:
        array = (struct array_elem *)realloc(array, sleep_array_len * sizeof(struct
array_elem));
        array[sleep_array_len - 1].thrd = element.thrd;
        array[sleep_array_len - 1].ticks = element.ticks;
        sort_array();
        return;
    }
}

```

```

void pop()

```

```

{
    while (sleep_array_len && array[sleep_array_len - 1].ticks == ticks)

```

```

{
    thread_unblock(array[sleep_array_len - 1].thrd);
    array[sleep_array_len - 1].thrd = NULL;
    array[sleep_array_len - 1].ticks = 0;
    sleep_array_len--;
}
}

void timer_sleep(int64_t ticks)
{
    int64_t start = timer_ticks();

    ASSERT(intr_get_level() == INTR_ON);
    if (ticks <= 0)
        return;

    struct array_elem sleep_thread;
    sleep_thread.thrd = thread_current();
    sleep_thread.ticks = start + ticks;
    enum intr_level old_level = intr_disable();
    insert(sleep_thread);
    thread_block();
    intr_set_level(old_level);
}

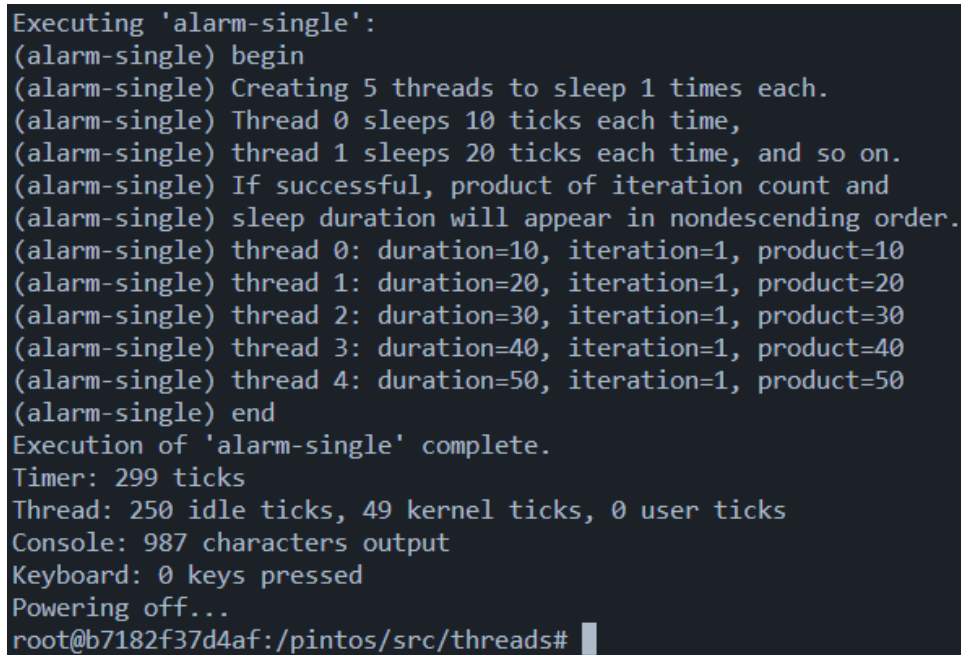
static void
timer_interrupt(struct intr_frame *args UNUSED)
{
    ticks++;
    pop(ticks);
    thread_tick();
}

```

- Timer.h

```
struct array_elem {
    struct thread *thrd;
    int64_t ticks;
};
struct array_elem *array;
int64_t sleep_array_len;
```

Результаты выполнения тестов представлены на скриншотах ниже:



```
Executing 'alarm-single':
(alarm-single) begin
(alarm-single) Creating 5 threads to sleep 1 times each.
(alarm-single) Thread 0 sleeps 10 ticks each time,
(alarm-single) thread 1 sleeps 20 ticks each time, and so on.
(alarm-single) If successful, product of iteration count and
(alarm-single) sleep duration will appear in nondescending order.
(alarm-single) thread 0: duration=10, iteration=1, product=10
(alarm-single) thread 1: duration=20, iteration=1, product=20
(alarm-single) thread 2: duration=30, iteration=1, product=30
(alarm-single) thread 3: duration=40, iteration=1, product=40
(alarm-single) thread 4: duration=50, iteration=1, product=50
(alarm-single) end
Execution of 'alarm-single' complete.
Timer: 299 ticks
Thread: 250 idle ticks, 49 kernel ticks, 0 user ticks
Console: 987 characters output
Keyboard: 0 keys pressed
Powering off...
root@b7182f37d4af:/pintos/src/threads#
```

Рисунок 3. Результат теста alarm-single


```

(alarm-multiple) thread 3: duration=40, iteration=2, product=80
(alarm-multiple) thread 2: duration=30, iteration=3, product=90
(alarm-multiple) thread 1: duration=20, iteration=5, product=100
(alarm-multiple) thread 4: duration=50, iteration=2, product=100
(alarm-multiple) thread 3: duration=40, iteration=3, product=120
(alarm-multiple) thread 1: duration=20, iteration=6, product=120
(alarm-multiple) thread 2: duration=30, iteration=4, product=120
(alarm-multiple) thread 1: duration=20, iteration=7, product=140
(alarm-multiple) thread 2: duration=30, iteration=5, product=150
(alarm-multiple) thread 4: duration=50, iteration=3, product=150
(alarm-multiple) thread 3: duration=40, iteration=4, product=160
(alarm-multiple) thread 2: duration=30, iteration=6, product=180
(alarm-multiple) thread 4: duration=50, iteration=4, product=200
(alarm-multiple) thread 3: duration=40, iteration=5, product=200
(alarm-multiple) thread 2: duration=30, iteration=7, product=210
(alarm-multiple) thread 3: duration=40, iteration=6, product=240
(alarm-multiple) thread 4: duration=50, iteration=5, product=250
(alarm-multiple) thread 3: duration=40, iteration=7, product=280
(alarm-multiple) thread 4: duration=50, iteration=6, product=300
(alarm-multiple) thread 4: duration=50, iteration=7, product=350
(alarm-multiple) end
Execution of 'alarm-multiple' complete.
Timer: 618 ticks
Thread: 550 idle ticks, 68 kernel ticks, 0 user ticks
Console: 2955 characters output
Keyboard: 0 keys pressed
Powering off...
root@b7182f37d4af:/pintos/src/threads#

```

Рисунок 4. alarm-multiple

```

(alarm-simultaneous) iteration 4, thread 0: woke up 10 ticks later
(alarm-simultaneous) iteration 4, thread 1: woke up 0 ticks later
(alarm-simultaneous) iteration 4, thread 2: woke up 0 ticks later
(alarm-simultaneous) end
Execution of 'alarm-simultaneous' complete.
Timer: 305 ticks
Thread: 249 idle ticks, 56 kernel ticks, 0 user ticks
Console: 1615 characters output
Keyboard: 0 keys pressed
Powering off...

```

Рисунок 5. alarm-simultaneous

```
Pintos booting with 3,968 KB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer... 314,163,200 loops/s.
Boot complete.
Executing 'alarm-zero':
(alarm-zero) begin
(alarm-zero) PASS
(alarm-zero) end
Execution of 'alarm-zero' complete.
Timer: 45 ticks
Thread: 0 idle ticks, 45 kernel ticks, 0 user ticks
Console: 385 characters output
Keyboard: 0 keys pressed
Powering off...
root@rk7102[374455:/pintos/ops/thread#
```

Рисунок 6. alarm-zero

```
Calibrating timer... 254,771,200 loops/s.
Boot complete.
Executing 'alarm-negative':
(alarm-negative) begin
(alarm-negative) PASS
(alarm-negative) end
Execution of 'alarm-negative' complete.
Timer: 47 ticks
Thread: 0 idle ticks, 47 kernel ticks, 0 user ticks
Console: 409 characters output
Keyboard: 0 keys pressed
Powering off...
```

Рисунок 7. alarm-negative

3. ВЫВОД

В ходе лабораторной работы был изучен механизм работы системного таймера в ОС Pintos. Проанализировав принцип его работы, стало понятно, что такой подход не является оптимальным и нуждается в переработке. Суть переработки заключается в создании хранилища процессов, которые заблокированы. Во время переработок возникали проблемы с освобождением памяти элемента массива спящих потоков, так что было принято решение просто обнулять элементы, которые вышли из массива. Такой подход не особо рациональный с точки зрения используемой памяти, но точно лучше, чем активное ожидание, реализованное с самого начала