

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

Отчет по лабораторной работе №1

«Обработка сигналов»
по дисциплине «Операционные системы»

Вариант № 6

Выполнил студент гр. 3-513П8-4

В.А. Кабин

«__»_____20__ г.

Проверил

к.т.н., доцент кафедры КСУП

В. П. Коцубинский

Томск 2025

Цель работы. Целью выполнения настоящей лабораторной работы является получение навыков программного управления процессами с помощью сигналов

Задание

Требуется разработать программу на языке C: ввод с клавиатуры содержимого текстового файла, имя которого задается в командной строке, используя стандартные функции.

Решение задания.

Программа должна:

- открыть существующий текстовый файл.
- создать два дочерних процесса.
- обеспечить поочередное чтение содержимого файла фиксированными порциями (по 10 символов) каждым из дочерних процессов.
- предварять каждый вывод номером процесса, который выполняет чтение.
- завершить работу либо при достижении конца файла, либо по истечении заданного интервала времени (10 секунд).

В соответствии с заданием, была разработана программа на языке Си (листинг 1).

Основные этапы работы программы:

Этап 1: Проверка аргументов командной строки

Программа проверяет, что пользователь указал имя файла в качестве аргумента командной строки.

Если аргумент не указан или указан неверно, программа выводит сообщение об использовании и завершается.

Этап 2: Открытие файла

Файл открывается в режиме только для чтения (O_RDONLY).

Если файл не может быть открыт (например, из-за отсутствия файла или недостатка прав доступа), программа завершается с соответствующим сообщением об ошибке.

Этап 3: Установка таймера и обработчика сигнала

Устанавливается обработчик для сигнала SIGALRM, который будет вызван по истечении заданного времени (10 секунд).

Таймер запускается с помощью функции alarm(TIMEOUT).

Этап 4: Создание дочерних процессов

Родительский процесс создает два дочерних процесса с помощью системного вызова fork().

Каждый дочерний процесс получает доступ к одному и тому же файловому дескриптору, наследуемому от родительского процесса.

Этап 5: Инициализация семафора

Создается семафор для координации работы между дочерними процессами.

Семафор используется для обеспечения чередования доступа к файлу: один процесс читает данные, пока другой ждет своей очереди.

Начальное значение семафора устанавливается равным 1, чтобы первый процесс мог начать работу без задержки.

Этап 6: Чтение данных из файла

Каждый дочерний процесс выполняет следующие действия:

а) Установка обработчика сигнала SIGTERM:

Дочерние процессы регистрируют обработчик для сигнала SIGTERM, чтобы корректно завершаться при получении этого сигнала.

б) Чтение данных из файла :

Процесс ожидает своей очереди с помощью операции P (уменьшение значения семафора).

Процесс читает фиксированную порцию данных (до 10 символов) из файла с помощью функции read().

Процесс выводит прочитанные данные на экран, предваряя их номером своего процесса.

Процесс разблокирует другой процесс с помощью операции V (увеличение значения семафора).

в) Проверка условий завершения :

Если достигнут конец файла (`bytes_read <= 0`) или установлен флаг тайм-аута (`timeout_occurred`), процесс завершает свою работу.

Этап 7: Управление тайм-аутом (родительский процесс)

а) Мониторинг состояния дочерних процессов :

Родительский процесс периодически проверяет статус дочерних процессов с помощью неблокирующего вызова `waitpid()` с флагом `WNOHANG`.

б) Обработка истечения тайм-аута :

Если тайм-аут истек (`timeout_occurred == 1`):

Родительский процесс отправляет сигнал SIGTERM дочерним процессам, чтобы они завершились корректно.

Если дочерние процессы не реагируют на SIGTERM, родительский процесс отправляет им сигнал SIGKILL для принудительного завершения.

в) Ожидание завершения дочерних процессов :

Родительский процесс продолжает проверять статус дочерних процессов до тех пор, пока все они не завершатся.

Этап 8: Завершение работы

а) Удаление семафора :

После завершения работы всех дочерних процессов, родительский процесс удаляет созданный семафор.

б) Вывод сообщения о завершении :

Родительский процесс выводит сообщение "Родительский процесс завершил работу."

Листинг 1. Программа на языке Си

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/sem.h>

#define PORTION_SIZE 10
#define TIMEOUT 10

// Глобальная переменная для сигнала тайм-аута
volatile sig_atomic_t timeout_occurred = 0;

// Обработчик сигнала SIGALRM (тайм-аут)
void handle_alarm(int signum) {
    timeout_occurred = 1; // Устанавливаем флаг тайм-аута
}

// Функция для инициализации семафора
int init_semaphore(key_t key, int value) {
    int sem_id = semget(key, 1, IPC_CREAT | 0666);
    if (sem_id == -1) {
        perror("Ошибка создания семафора");
        exit(EXIT_FAILURE);
    }
    union semun {
        int val;
    } arg;
    arg.val = value;
    if (semctl(sem_id, 0, SETVAL, arg) == -1) {
        perror("Ошибка инициализации семафора");
        exit(EXIT_FAILURE);
    }
    return sem_id;
}

// Функция для операций с семафором
void semaphore_op(int sem_id, int op) {
    struct sembuf sem_op;
```

```

sem_op.sem_num = 0;
sem_op.sem_op = op;
sem_op.sem_flg = 0;
if (semop(sem_id, &sem_op, 1) == -1) {
    perror("Ошибка операции с семафором");
    exit(EXIT_FAILURE);
}
}

// Обработчик сигнала SIGTERM для дочерних процессов
void handle_term(int signum) {
    printf("Процесс %d получил сигнал SIGTERM и завершается.\n", getpid());
    exit(EXIT_SUCCESS);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Использование: %s <имя_файла>\n", argv[0]);
        return EXIT_FAILURE;
    }

    const char *filename = argv[1];
    int fd = open(filename, O_RDONLY);
    if (fd == -1) {
        perror("Ошибка открытия файла");
        return EXIT_FAILURE;
    }

    // Установка обработчика сигнала SIGALRM
    signal(SIGALRM, handle_alarm);

    // Запуск таймера на 10 секунд
    alarm(10);

    pid_t pid1, pid2;

    // Создание ключа для семафора
    key_t key = ftok("/tmp", 'S');
    if (key == -1) {
        perror("Ошибка создания ключа для семафора");
        return EXIT_FAILURE;
    }

    // Инициализация семафора

```

```
int sem_id = init_semaphore(key, 1); // Семафор начинает с 1 (процесс 1 может начинать)
```

```
// Создание первого дочернего процесса
```

```
if ((pid1 = fork()) == 0) {
```

```
    // Первый дочерний процесс
```

```
    int process_number = 1;
```

```
    // Установка обработчика сигнала SIGTERM
```

```
    signal(SIGTERM, handle_term);
```

```
while (!timeout_occurred) {
```

```
    // Ожидание своей очереди
```

```
    semaphore_op(sem_id, -1); // P операция: уменьшаем семафор
```

```
    // Чтение порции данных из файла
```

```
    char buffer[PORTION_SIZE + 1];
```

```
    int bytes_read = read(fd, buffer, PORTION_SIZE);
```

```
    if (bytes_read <= 0 || timeout_occurred) {
```

```
        break; // Конец файла или тайм-аут
```

```
    }
```

```
    buffer[bytes_read] = '\0'; // Добавляем завершающий ноль
```

```
    printf("Процесс %d: %s\n", process_number, buffer);
```

```
    // Позволяем другому процессу работать
```

```
    semaphore_op(sem_id, 1); // V операция: увеличиваем семафор
```

```
    // Проверяем тайм-аут после вывода
```

```
    if (timeout_occurred) {
```

```
        break;
```

```
    }
```

```
}
```

```
close(fd); // Закрываем файловый дескриптор
```

```
exit(EXIT_SUCCESS);
```

```
}
```

```
// Создание второго дочернего процесса
```

```
if ((pid2 = fork()) == 0) {
```

```
    // Второй дочерний процесс
```

```
    int process_number = 2;
```

```
    // Установка обработчика сигнала SIGTERM
```

```

signal(SIGTERM, handle_term);

while (!timeout_occurred) {
    // Ожидание своей очереди
    semaphore_op(sem_id, -1); // P операция: уменьшаем семафор

    // Чтение порции данных из файла
    char buffer[PORTION_SIZE + 1];
    int bytes_read = read(fd, buffer, PORTION_SIZE);

    if (bytes_read <= 0 || timeout_occurred) {
        break; // Конец файла или тайм-аут
    }

    buffer[bytes_read] = '\0'; // Добавляем завершающий ноль
    printf("Процесс %d: %s\n", process_number, buffer);

    // Позволяем другому процессу работать
    semaphore_op(sem_id, 1); // V операция: увеличиваем семафор

    // Проверяем тайм-аут после вывода
    if (timeout_occurred) {
        break;
    }
}

close(fd); // Закрываем файловый дескриптор
exit(EXIT_SUCCESS);
}

// Процесс-родитель
close(fd); // Закрываем файловый дескриптор

// Ожидание завершения дочерних процессов
while (1) {
    if (timeout_occurred) {
        // Если тайм-аут истек, отправляем сигнал SIGTERM дочерним
        // процессам
        kill(pid1, SIGTERM);
        kill(pid2, SIGTERM);

        // Ждем некоторое время, чтобы дочерние процессы завершились
        sleep(1);

        // Проверяем, завершились ли дочерние процессы

```

```

    int status;
    if (waitpid(pid1, &status, WNOHANG) == 0) {
        // Если процесс 1 не завершился, отправляем SIGKILL
        kill(pid1, SIGKILL);
    }
    if (waitpid(pid2, &status, WNOHANG) == 0) {
        // Если процесс 2 не завершился, отправляем SIGKILL
        kill(pid2, SIGKILL);
    }

    break;
}

// Проверяем статус дочерних процессов
pid_t terminated_pid = waitpid(-1, NULL, WNOHANG);
if (terminated_pid == pid1 || terminated_pid == pid2) {
    continue;
}
if (terminated_pid == -1 && !timeout_occurred) {
    break;
}
}

// Удаление семафора
if (semctl(sem_id, 0, IPC_RMID) == -1) {
    perror("Ошибка удаления семафора");
}
printf("Родительский процесс завершил работу.\n");
return EXIT_SUCCESS;
}

```

Сохраним приведённый выше код в файле `processes.c`, затем скомпилируем его:

```
gcc processes.c -o processes
```

Создадим предварительно тестовый текстовый файл `input.txt`:
Hello, this is a test file for the program.

Посмотрим текущие процессы ОС командой `ps aux`:

И убеждаемся, что процесса «./processes input.txt» действительно больше нет в системе, что значит что он благополучно завершён.

Выводы

В ходе выполнения лабораторной работы были получены практические навыки работы с процессами, сигналами и семафорами в операционной системе UNIX. Полученные знания и навыки могут быть применены для разработки более сложных многопоточных и многопроцессорных приложений.

Для закрепления полученных знаний была создана программа на языке Си, которая создает два дочерних процесса для поочередного чтения данных из файла и завершает их работу по истечении заданного времени.

Программа создает два дочерних процесса, которые поочередно читают данные из файла. Использование семафора обеспечивает корректную синхронизацию дочерних процессов. Программа завершает работу по истечении 10 секунд или при достижении конца файла. В результате все процессы (родительский и дочерние) завершаются корректно, и семафор удаляется, что было проверено путём анализа системных процессов до запуска программы и после.