

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Операционные среды и системное программирование

ОТЧЁТ
к лабораторной работе №4
на тему

УПРАВЛЕНИЕ ПРОЦЕССАМИ И ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ

Выполнил: студент гр. 253503
Тимошевич К.С.

Проверил: ассистент кафедры
информатики Гриценко Н.Ю.

Минск 2025

СОДЕРЖАНИЕ

1 Постановка задачи.....	3
2 Описание работы программы.....	4
2.1 Обработка сигналов.....	4
2.2 Создание дочернего процесса.....	4
2.3 Периодическое выполнение действий.....	4
3 Ход выполнения программы.....	5
3.1 Примеры выполнения задания.....	5
Вывод.....	6
Список использованных источников.....	7
Приложение А (справочное) Исходный код.....	8

1 ПОСТАНОВКА ЗАДАЧИ

Целью данной лабораторной работы является изучение механизмов управления процессами и их взаимодействия в операционных системах семейства *Unix/Linux*. В рамках работы предстоит разработать программу, которая демонстрирует возможность создания самовосстанавливающегося процесса. Такой процесс должен быть способен корректно обрабатывать сигналы, которые обычно приводят к его завершению, и восстанавливать свою работу путем создания копии самого себя. Это позволяет избежать полного завершения процесса при получении критических сигналов, таких как *SIGINT* или *SIGTERM*, что может быть полезно в системах, где требуется высокая отказоустойчивость и непрерывность выполнения задач [1].

Программа должна быть реализована с использованием базовых механизмов работы с процессами, таких как *fork*, *exec* и обработка сигналов. В качестве демонстрации работоспособности процесса предлагается реализовать периодическое выполнение некоторого действия, например, увеличение счетчика и запись его значения в файл. Это позволит визуально наблюдать за работой процесса и убедиться в его способности восстанавливаться после получения сигналов.

Для реализации программы необходимо учитывать особенности работы с процессами в *Unix*-системах. В отличие от *Windows*, в *Unix* создание и управление процессами осуществляется более гибко и естественно, что позволяет легко реализовать механизм самовосстановления. Программа должна быть спроектирована таким образом, чтобы при получении сигнала завершения она создавала свою копию, которая продолжает выполнение с того же места, где был прерван оригинальный процесс. После успешного создания копии оригинальный процесс может завершиться, передав управление новой копии.

2 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

Данный раздел описывает работу программы, реализующей самовосстанавливающийся процесс. Программа демонстрирует механизм обработки сигналов, которые обычно приводят к завершению процесса, и создает свою копию для продолжения выполнения с прерванного места. Рассмотрены основные этапы работы программы, включая обработку сигналов, создание дочернего процесса и периодическое выполнение действий для демонстрации работоспособности.

2.1 Обработка сигналов

Программа начинает свою работу с установки обработчиков сигналов для *SIGTERM* и *SIGINT* с помощью функции *signal*. Эти сигналы обычно используются для завершения процессов, но в данной программе они перехватываются и обрабатываются специальным образом. При получении одного из этих сигналов вызывается функция *handle_signal*, которая отвечает за создание копии процесса.

Функция *handle_signal* выводит сообщение о получении сигнала и создает дочерний процесс с помощью системного вызова *fork* [2]. Если создание дочернего процесса завершается успешно, родительский процесс завершает свою работу, а дочерний процесс продолжает выполнение с того же места, где был прерван оригинальный процесс. Это позволяет избежать полного завершения программы при получении сигналов завершения.

2.2 Создание дочернего процесса

После получения сигнала программа создает дочерний процесс с помощью *fork*. Если вызов *fork* завершается успешно, в дочернем процессе переменная *keep_running* снова устанавливается в значение 1, что позволяет процессу продолжить выполнение. Родительский процесс завершает свою работу, выведя сообщение о завершении. Таким образом, программа реализует механизм самовосстановления, при котором процесс продолжает работать даже после получения сигналов завершения..

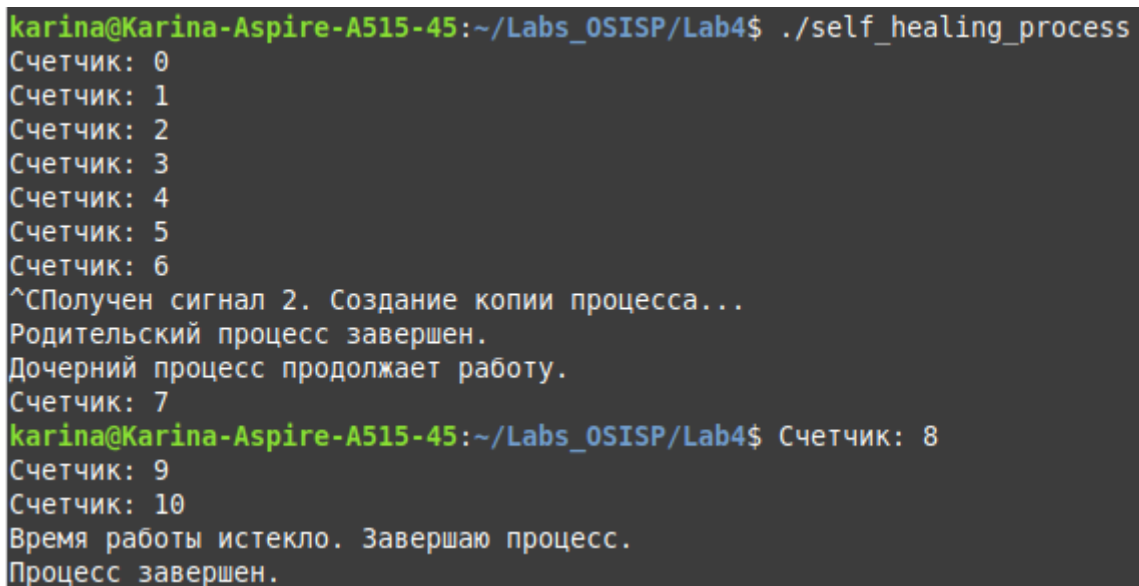
2.3 Периодическое выполнение действий

Для демонстрации работоспособности программы в основном цикле выполняется периодическое увеличение счетчика и вывод его значения на экран. Счетчик увеличивается каждую секунду с помощью функции *sleep*. Программа также отслеживает время своего выполнения с помощью функции *time* [3]. Если время работы программы превышает 10 секунд, она завершает свою работу, выводя соответствующее сообщение.

3 ХОД ВЫПОЛНЕНИЯ ПРОГРАММЫ

3.1 Примеры выполнения задания

На рисунке 3.1 продемонстрирована работа. Программа начинает свою работу, выводя на экран значение счетчика, которое увеличивается каждую секунду. На скриншоте видно, как счетчик последовательно увеличивается от 0 до 6. В этот момент пользователь отправляет сигнал *SIGINT* (например, нажатием *Ctrl+C*), что приводит к выводу сообщения "Получен сигнал 2. Создание копии процесса...". Программа перехватывает сигнал, создает дочерний процесс и завершает родительский процесс, о чем свидетельствует сообщение "Родительский процесс завершен.". Дочерний процесс продолжает выполнение с того же места, где был прерван родительский процесс, и выводит сообщение "Дочерний процесс продолжает работу.". Счетчик продолжает увеличиваться, начиная с 7, что демонстрирует корректное восстановление программы. После достижения счетчиком значения 10 программа завершает свою работу, выводя сообщение "Время работы истекло. Завершаю процесс." и "Процесс завершен.", что указывает на успешное завершение выполнения программы. Этот пример наглядно демонстрирует механизм самовосстановления программы и ее устойчивость к сигналам завершения.



```
karina@Karina-Aspire-A515-45:~/Labs_OSISP/Lab4$ ./self_healing_process
Счетчик: 0
Счетчик: 1
Счетчик: 2
Счетчик: 3
Счетчик: 4
Счетчик: 5
Счетчик: 6
^СПолучен сигнал 2. Создание копии процесса...
Родительский процесс завершен.
Дочерний процесс продолжает работу.
Счетчик: 7
karina@Karina-Aspire-A515-45:~/Labs_OSISP/Lab4$ Счетчик: 8
Счетчик: 9
Счетчик: 10
Время работы истекло. Завершаю процесс.
Процесс завершен.
```

Рисунок 3.1 – Результаты работы

ВЫВОД

В ходе выполнения лабораторной работы была разработана и реализована программа на языке C, демонстрирующая механизм самовосстанавливающегося процесса. Программа корректно обрабатывает сигналы, которые обычно приводят к завершению процесса (*SIGINT* и *SIGTERM*), и создает свою копию для продолжения выполнения с прерванного места. Это позволяет избежать полного завершения программы при получении критических сигналов, что особенно полезно в системах, требующих высокой отказоустойчивости и непрерывности выполнения задач.

В процессе работы были изучены и применены основные механизмы управления процессами в *Unix/Linux*, такие как создание процессов с помощью *fork*, обработка сигналов с использованием *signal*, а также управление временем выполнения программы с помощью функций *sleep* и *time*. Программа успешно демонстрирует возможность восстановления после получения сигналов завершения, что подтверждается корректным увеличением счетчика и выводом соответствующих сообщений на экран.

Для наглядности работы программы был реализован периодический вывод значения счетчика, который увеличивается каждую секунду. Это позволяет визуально наблюдать за работой процесса и убедиться в его способности восстанавливаться после получения сигналов. Программа также отслеживает время своего выполнения и завершает работу через 10 секунд, что упрощает тестирование и демонстрацию ее функциональности.

В результате выполнения лабораторной работы был получен практический опыт работы с процессами в *Unix*-системах, включая создание дочерних процессов, обработку сигналов и управление временем выполнения. Программа успешно демонстрирует механизм самовосстановления, что может быть полезно при разработке отказоустойчивых систем, требующих непрерывного выполнения задач даже в условиях внешних воздействий, таких как сигналы завершения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Понятие о сигналах [Электронный ресурс]. – Режим доступа: https://www.opennet.ru/docs/RUS/linux_parallel/node10.html. – Дата доступа: 01.03.2024.

[2] Создание процессов с помощью вызова fork [Электронный ресурс]. – Режим доступа: https://www.opennet.ru/docs/RUS/linux_parallel/node7.html. – Дата доступа: 01.03.2024.

[3] What does the sleep command do in Linux [Электронный ресурс]. – Режим доступа: <https://phoenixnap.com/kb/linux-sleep>. – Дата доступа: 01.03.2024.

ПРИЛОЖЕНИЕ А

(справочное)

Исходный код

```
# main.c

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <time.h>

volatile sig_atomic_t keep_running = 1;

void handle_signal(int sig) {
    if (sig == SIGTERM || sig == SIGINT) {
        printf("Получен сигнал %d. Создание копии процесса...\n", sig);

        pid_t pid = fork();
        if (pid < 0) {
            perror("Ошибка при создании копии процесса");
            exit(EXIT_FAILURE);
        } else if (pid == 0) {
            printf("Дочерний процесс продолжает работу.\n");
            keep_running = 1;
        } else {
            printf("Родительский процесс завершен.\n");
            exit(EXIT_SUCCESS);
        }
    }
}

int main() {
    signal(SIGTERM, handle_signal);
    signal(SIGINT, handle_signal);

    int counter = 0;
    time_t start_time = time(NULL);

    while (keep_running) {
        printf("Счетчик: %d\n", counter++);
        sleep(1);
        if (time(NULL) - start_time > 10) {
            printf("Время работы истекло. Завершаю процесс.\n");
            keep_running = 0;
        }
    }

    printf("Процесс завершен.\n");
    return 0;
}
```