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»

ОТЧЕТ
К лабораторной работе № 4
на тему «Управление процессами и взаимодействие процессов»

Выполнил Н. В. Климкович

Проверил Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является изучение основных особенностей подсистемы управления процессами и средств взаимодействия процессов в Unix, а также практическое проектирование, реализация, и отладка программных комплексов из нескольких взаимодействующих процессов.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Процесс представляет собой выполнение программы на компьютере, обладающее собственной областью памяти, включая код, данные и системные ресурсы.[1]

Планировщик, как компонент операционной системы, занимается распределением процессорного времени между процессами.[2] Приоритеты процессов определяются для управления их выполнением в соответствии с значимостью и требованиями.

Межпроцессное взаимодействие (IPC) позволяет обмениваться данными и синхронизировать работу процессов. Каналы обеспечивают обмен данными, используя потоковые или дуплексные каналы. Семафоры используются для синхронизации и ограничения доступа к ресурсам.[3]

Сокеты обеспечивают сетевое взаимодействие между процессами на разных компьютерах. Сигналы являются сообщениями, которые процессы или ядро отправляют другим процессам для уведомления о событиях или запроса действий.

Сигналы в UNIX-подобных системах управляют процессами, а их протоколирование позволяет отслеживать информацию о событиях, таких как SIGHUP (перезагрузка), SIGTERM (завершение), SIGINT (прерывание) и других.[4] Для этого обычно используются системные вызовы, механизмы сигналов и межпроцессное взаимодействие. Протоколирование сигналов делает процесс-демон более информативным и контролируемым, повышая его эффективность в решении задач.

3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

В результате выполнения лабораторной работы было создано приложение, процесс, которого при получении сигнала, стандартно вызывающего завершение, создает свою копию процесса, который продолжает выполняться с прерванного места, и лишь после этого завершается, избегая таким образом безусловного «уничтожения» неперехватываемым сигналом. В качестве демонстрации живости процесса и его выполнения реализован счетчик, значение которого обновляется с заданной частотой. Результат работы приложения представлен на рисунке 3.1.

Рисунок 3.1 – Результат работы приложения

Таким образом, когда происходит экстренный выход из программы путем нажатия специальной комбинации, программа создает новый процесс и продолжает работать в нем, а изначальный завершает свою работу.

ВЫВОДЫ

В ходе выполнения данной лабораторной работы были изучены основные особенности подсистемы управления процессами и средств взаимодействия процессов в Unix. Практически было реализовано проектирование, разработка и отладка программных комплексов, состоящих из нескольких взаимодействующих процессов.

В ходе выполнения работы было разработано приложение, в котором при получении сигнала, вызывающего завершение, процесс создает свою копию и продолжает выполнение с прерванного места. Этот механизм позволяет избежать безусловного уничтожения процесса, обеспечивая его стабильность и возможность продолжения работы после получения сигнала.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Введение в Unix [Электронный ресурс]. – Режим доступа: https://unix.com. – Дата доступа: 01.02.2024.

[2] Makefile tutorial how to write [Электронный ресурс]. – Режим доступа: https://www.tutorialspoint.com/makefile. – Дата доступа: 02.02.2024.

[3] Функции C [Электронный ресурс]. – Режим доступа: https://metanit.com/cpp/tutorial/3.1.php. – Дата доступа: 05.02.2024.

[4] Классы C [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/cpp/cpp_classes.asp. – Дата доступа: 06.02.2024.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

Листинг 1 – Основной файл программы main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

int iteration = 0;

void signal_handler(int signo) {
    pid_t child_pid = fork();

    if (child_pid == -1) {
        perror("fork not created");
        exit(EXIT_FAILURE);
    } else if (child_pid > 0) {
        printf("\nParent: (PID %d), new child: (PID %d)\n", getpid(),
child_pid);
        exit(EXIT_SUCCESS);
    } else {
        printf("Child: (PID %d) continuing from iteration %d\n", getpid(),
iteration);
    }
}

int main() {
    if (signal(SIGINT, signal_handler) == SIG_ERR) {
        perror("signal is not tied to a method");
        return EXIT_FAILURE;
    }
    while (1) {
        if(iteration  == 10) {
        printf("exit");
            exit(EXIT_SUCCESS);
        }
        printf("Iteration %d\n", iteration++);

        sleep(1);
    }
    return 0;
}
```

Листинг 2 – Программная реализация makefile файла

```
CC = gcc
CFLAGS = -Wall

TARGET = lab4

all: $(TARGET)

$(TARGET): main.c
      $(CC) $(CFLAGS) -o $(TARGET) main.c

clean:
      rm -f $(TARGET)
```