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**

Студент В. П. Бычко

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Постановка задачи 3](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631498)

[2 Краткие теоретические сведения 4](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631499)

[3 Результаты выполнения лабораторной работы 6](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631500)

[Выводы 8](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631501)

[Приложение А (обязательное) Листинг кода](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631503) 9

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является изучение основных особенностей подсистемы управления процессами и средств взаимодействия процессов в Unix. Практическое проектирование, реализация и отладка программных комплексов из нескольких взаимодействующих процессов. Разработать и реализовать программу, процесс которой при получении сигнала, стандартно вызывающего завершение, создает свою копию, которая продолжает выполняться с прерванного места, и лишь после этого завершается.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Процесс (вычислительный процесс, process) – ассоциируется с выполняющейся программой. Идентификация – целочисленный идентификатор Process ID (PID), уникальный в пределах системы. Генерация идентификаторов – счетчик. По достижении верхнего предела «заворачивается» к началу и выбирает освободившиеся значения. В Unix-системах с поддержкой концепции вычислительных потоков (thread), потоки являются «облегченной версией» процессов с разделяемым адресным пространством, а место «настоящих» процессов занимают группы потоков.

Сигнал в операционных системах семейства Unix — асинхронное уведомление процесса о каком-либо событии, один из основных способов взаимодействия между процессами. Когда сигнал послан процессу, операционная система прерывает выполнение процесса, при этом, если процесс установил собственный обработчик сигнала, операционная система запускает этот обработчик, передав ему информацию о сигнале, если процесс не установил обработчик, то выполняется обработчик по умолчанию.

Количество поддерживаемых сигналов менялось по мере развития Unix-систем, наиболее часто используемый набор – около 30 (битовая маска рассчитана на 32 сигнала), но набор сигналов и номера некоторых из них могут быть различны в зависимости от семейства и версии Unix.

Доставка сигналов – асинхронная: процесс-получатель не обрабатывает сигналы немедленно после их доставки, но и не выбирает произвольно момент приема и обработки, поэтому его состояние (стадия выполнения) заранее не известно. Ядро доставляет сигнал и инициирует его обработку только в моменты – возврат из режима ядра в режим задачи после выполнения системного вызова или обработки исключения, перед переходом в состояние ожидания («сна»), если это позволяет приоритет, после выхода из состояния ожидания («сна»), если это позволяет приоритет. Таким образом, программа, которая не обращается к системным вызовам и не провоцирует исключения, не будет видеть отправленные ей сигналы (не считая «внешних» по отношению к ней исключений).

Сигнал INT посылается программе при нажатии клавиш Ctrl+C.

Предусмотрено три пути обработки поступивших сигналов – обработка по умолчанию (обработчик по умолчанию), в соответствии с номером сигналов (см. выше), игнорирование, отсутствие реакции на сигнал независимо от его номера, но сигнал считается обработанным, пользовательский обработчик (функция в программе). Текущее назначение сигналов – обработка по умолчанию, игнорирование или пользовательский обработчик – диспозиция. Сигналы SIGKILL и SIGSTOP могут быть обработаны только «по умолчанию», их диспозицию изменить нельзя.

Для порождения процессов в ОС Linux можно использовать вызов fork(). Он создаёт новый процесс посредством копирования вызывающего процесса. Новый процесс считается дочерним процессом. Вызывающий процесс считается родительским процессом.

Дочерний процесс является точной копией родительского процесса за исключением следующих моментов – потомок имеет свой уникальный идентификатор процесса, и этот PID (идентификатор процесса) не совпадает ни с одним существующим идентификатором группы процессов, счётчики использования ресурсов и времени ЦП у потомка сброшены, набор ожидающих сигналов потомка изначально пуст.

При успешном завершении родителю возвращается PID процесса-потомка, а процессу-потомку возвращается 0. При ошибке родительскому процессу возвращается -1, процесс-потомок не создаётся, а значение errno устанавливается в соответствующее значение.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы был реализован самовосстанавливающийся процесс. На рисунке 3.1 представлен запуск программы, где вывод счетчика в файл выполняет главный процесс.

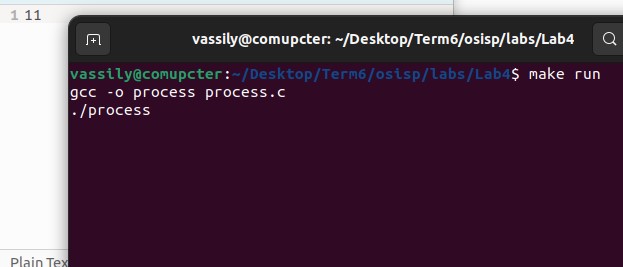


Рисунок 3.1 – Работа главного процесса

На рисунке 3.2 представлена работа программы, когда главного процессу подается сигнал о завершении работы. В данном случае создается копия процесса и продолжает свою работу с того места, где остановился главный процесс.

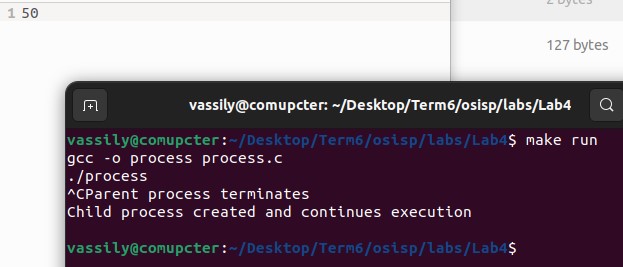


Рисунок 3.2 – Обработка сигнала программой

На рисунке 3.3 представлен результат работы программы через 58 секунд от старта. В файл записан счетчик секунд от начала программы. Сначала счетчик записывала основная программа, потом программа копия

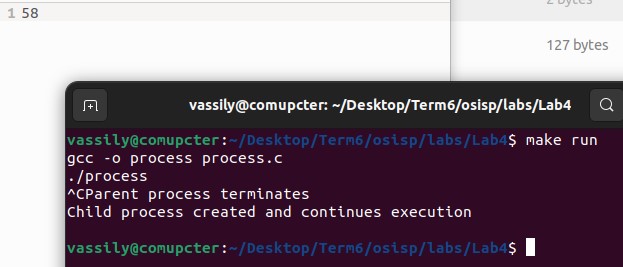


Рисунок 3.3 – Результат программы

## ВЫВОДЫ

В ходе выполнения данной лабораторной были изучены основных особенностей подсистемы управления процессами и средств взаимодействия процессов в Unix. Разработана и реализована программа, процесс которой при получении сигнала, стандартно вызывающего завершение, создает свою копию, которая продолжает выполняться с прерванного места, и лишь после этого завершается.

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**process.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

static int count = 0;

void signal\_handler(int signum)

{

pid\_t pid = fork();

if (pid == 0)

{

printf("Child process created and continues execution\n");

while(1) {

count++;

FILE \*fp = fopen("count.txt", "w");

if (fp != NULL)

{

fprintf(fp, "%d", count);

fclose(fp);

}

sleep(1);

if (count == 200)

exit(0);

}

}

else

{

printf("Parent process terminates\n");

exit(0);

}

}

int main()

{

signal(SIGINT, signal\_handler);

while (1)

{

count++;

FILE \*fp = fopen("count.txt", "w");

if (fp != NULL)

{

fprintf(fp, "%d", count);

fclose(fp);

}

if (count == 200)

exit(0);

sleep(1);

}

return 0;

}