**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Операционные системы»**

**Тема: Взаимодействие родственных процессов. Управление процессами посредством сигналов. Многонитевое функционирование.**

| Студент гр. 1303 |  | Чубан Д.В. |
| --- | --- | --- |
| Преподаватель |  | Душутина Е.В. |

Санкт-Петербург

2023

## **Цель работы.**

* 1. **Задание.**

*Многонитевое функционирование*

18. Подготовьте программу, формирующую несколько нитей.

Нити для эксперимента могут быть практически идентичны.

Например, каждая нить в цикле: выводит на печать собственное имя и

инкрементирует переменную времени, после чего "засыпает"

(sleep(5); sleep(1); - для первой и второй нитей соответственно),

на экран (в файл) должно выводиться имя нити и количество

пятисекундных (для первой) и секундных (для второй) интервалов

функционирования каждой нити.

19. После запуска программы проанализируйте выполнение нитей,

распределение во времени. Используйте для этого вывод таблицы

процессов командой ps -axhf

Попробуйте удалить нить, зная ее идентификатор, командой kill.

Приведите и объясните результат.

20. Модифицируйте программу так, чтобы управление второй нитью

осуществлялось посредством сигнала SIGUSR1 из первой нити.

На пятой секунде работы приложения удалите вторую нить. Для этого

воспользуйтесь функцией

pthread\_kill(t2, SIGUSR); (t2 - дескриптор второй нити).

В остальном программу можно не изменять. Проанализируйте полученные

результаты.

21. Последняя модификация предполагает создание собственного

обработчика сигнала, содержащего уведомление о начале его работы и

возврат посредством функции pthread\_exit(NULL);

Сравните результаты, полученные после запуска этой модификации

программы с результатами предыдущей.

22. Перехватите сигнал «CTRL C» для процесса и потока однократно, а

также многократно с восстановлением исходного обработчика после

нескольких раз срабатывания. Проделайте аналогичную работу для

переназначения другой комбинации клавиш.

23. С помощью утилиты kill выведите список всех сигналов и дайте их

краткую характеристику на основе документации ОС. Для чего

предназначены сигналы с 32 по 64-й. Приведите пример их применения.

24. Проанализируйте процедуру планирования для процессов и потоков

одного процесса. 24.1. Обоснуйте результат экспериментально. 24.2.

Попробуйте процедуру планирования изменить. Подтвердите

экспериментально, если изменение возможно. 24.3. Задайте нитям разные

приоритеты программно и извне (объясните результат).

* 1. **Выполнение работы**

1. Напишем программу, формирующую несколько нитей. Каждая нить

Выводит на печать собственное имя и инкрементирует переменную counter, соответствующую своему счётчику, выводит на экран сколько в текущий момент времени прошло односекундных или пятисекундных интервалов и текущее время.



Рисунок 1 – структура программы, иллюстрирующей создание потоков

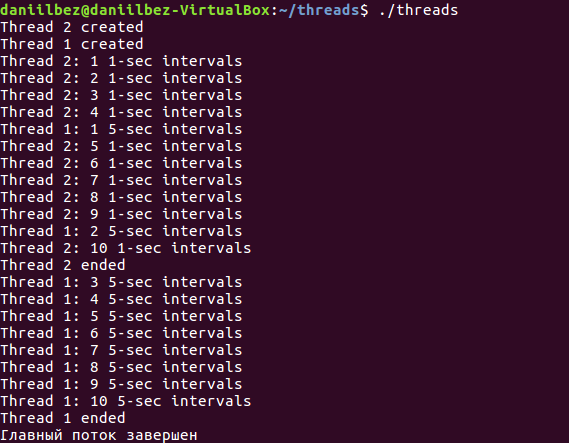


Рисунок 2 – результат выполнения представленной программы

2. Проанализируем выполнение нитей, распределение во времени. Для

этого используем вывод таблицы ps -axhf в разных местах программы (когда, например, будет работать первый и второй поток и когда будет работать только второй поток). Вывод программы направим в файл, и будем анализировать процессы с помощью htop.

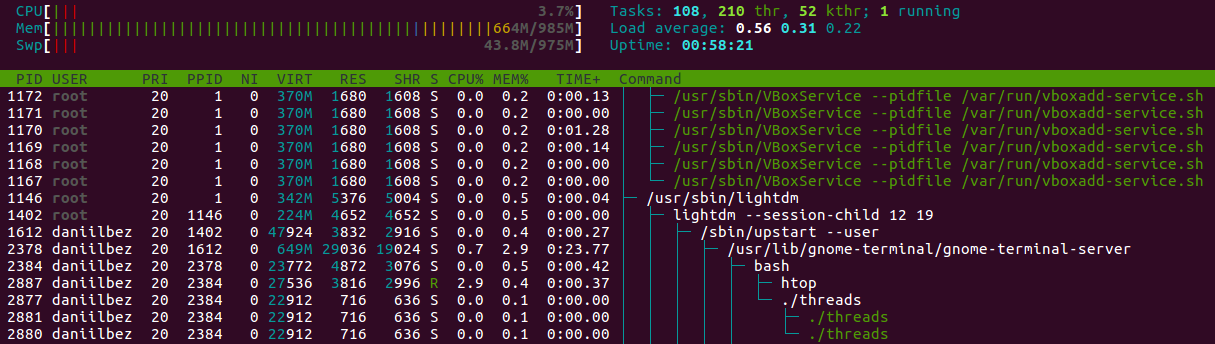


Рисунок 3 – потоки при запуске программы

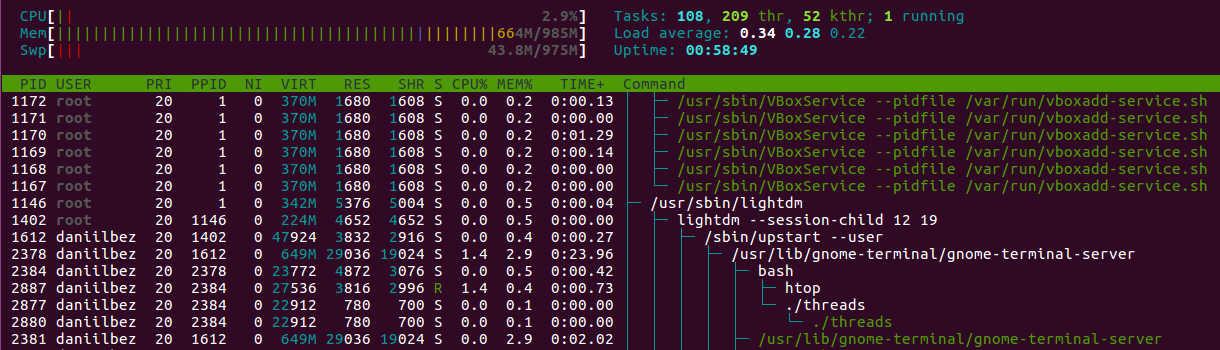


Рисунок 4 – потоки после завершения потока 2

Как видно, сначала работало два потока, далее один закончил свою работу и, наконец, второй закончил работу, и программа завершилась. Попробуем завершить один из потоков, послав ему *SIGTERM*.

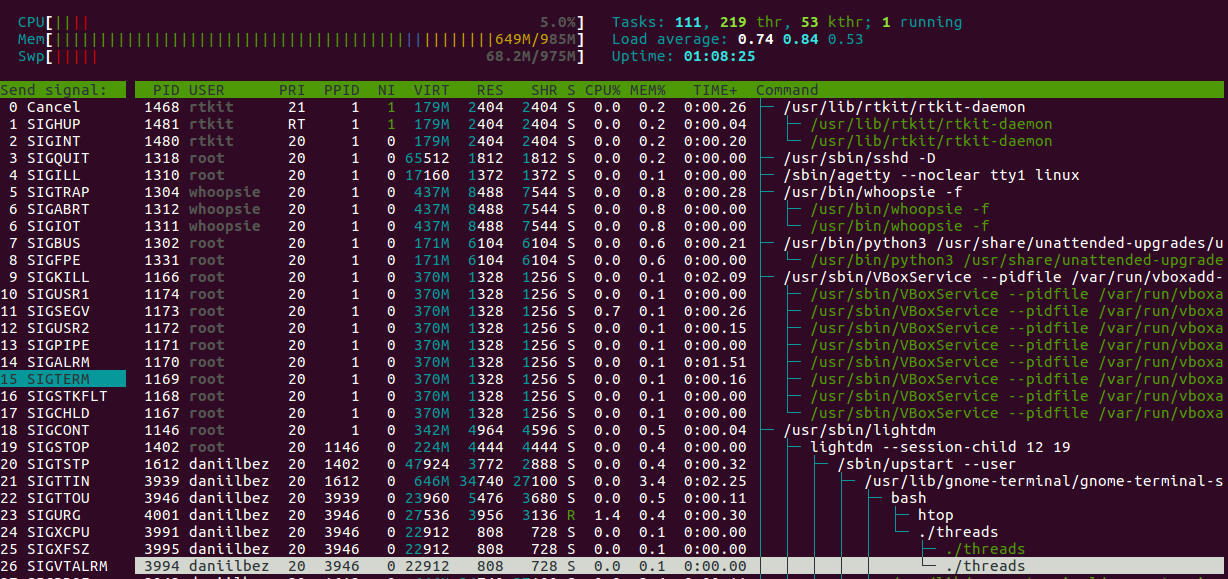


Рисунок 5 – kill второго потока

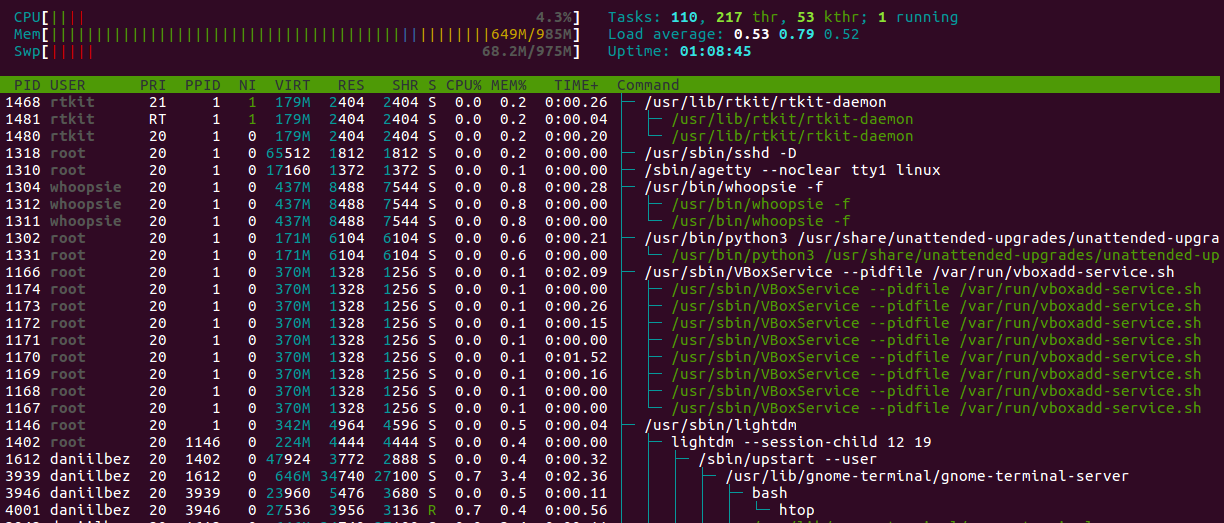


Рисунок 6 – результат отправления потока SIGTERM второму потоку

Как нетрудно заметить, после отправления данного сигнала в один из потоков -завершилась программа целиком.

3. Модифицирована программа так, чтобы управление второй нитью

осуществлялось посредством сигнала SIGUSR1 из первой нити. На пятой

секунде работы приложения удалена вторая нить. Для этого воспользовалась

функцией pthread\_kill(t2, SIGUSR);

Можно сделать вывод, что с помощью указанного способа удалить

только вторую нить не удалось, удалился процесс в целом.

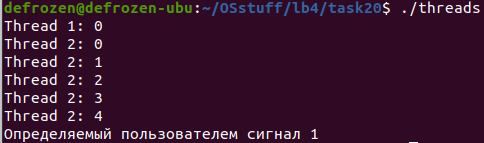


Рисунок 7 – результат выполнения эксперимента

Результат совпал с ожидаемым.

4. С помощью такого управления удалось корректно организовать

удаление одной нити: все остальные нити данного процесса процесса

сохранились и продолжили выполняться.

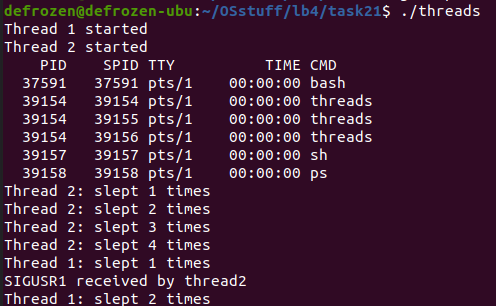


Рисунок 9 – результат выполнения модифицированной программы

5. Перехватим сигнал Ctrl C для процесса и потока однократно, а также

многократно с восстановлением исходного обработчика после нескольких раз срабатывания. Содержание программы и результат работы на рисунке 35.

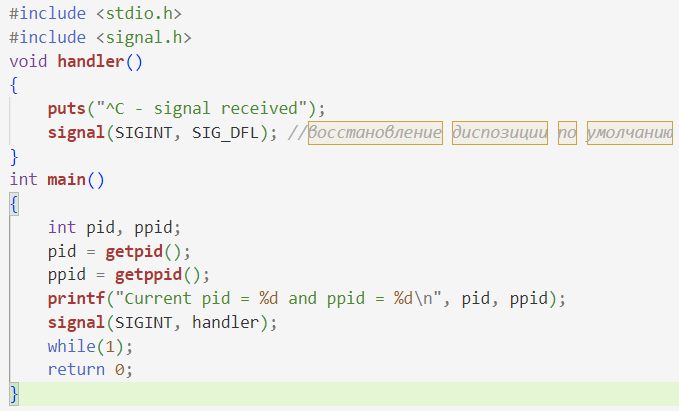


Рисунок 10 – программа, перехватывающая ^C единожды

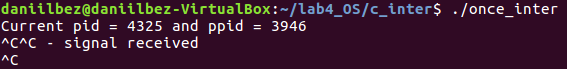


Рисунок 11 – результат работы и вызова 2 прерываний

Как видно из результата выполнения программы – после вызова прерывания в первый раз, сработал наш handler, который вывел сообщение и восстановил диспозицию по умолчанию, после срабатывания второго прерывания, обработчик уже не вызывался, поэтому программа прекратила свою работу.

Изменим нашу программу таким образом, что теперь срабатывание обработчика будет увеличивать значение статической переменной, после достижения ей значения 5, восстановим стандартную диспозицию.

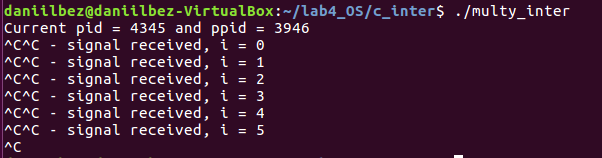


Рисунок 12 – результат работы измененной программы

Проделаем аналогичную работу для другой комбинации клавиш (Ctrl + Z). С помощью данной команды мы отправляем сигнал SIGTSTP, который «ставит на паузу» программу.

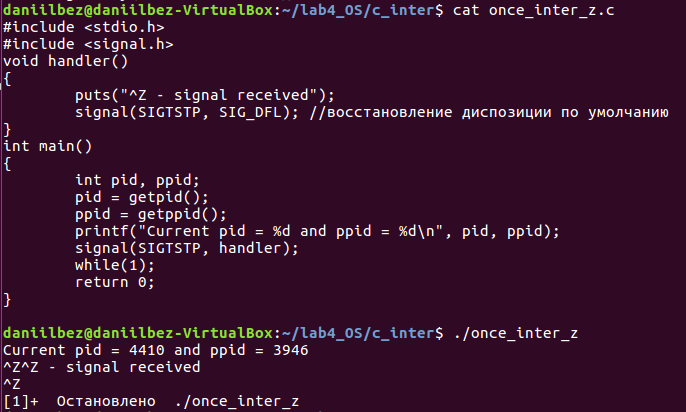


Рисунок 13 – текст программы с однократной обработкой сигнала и результат ее работы

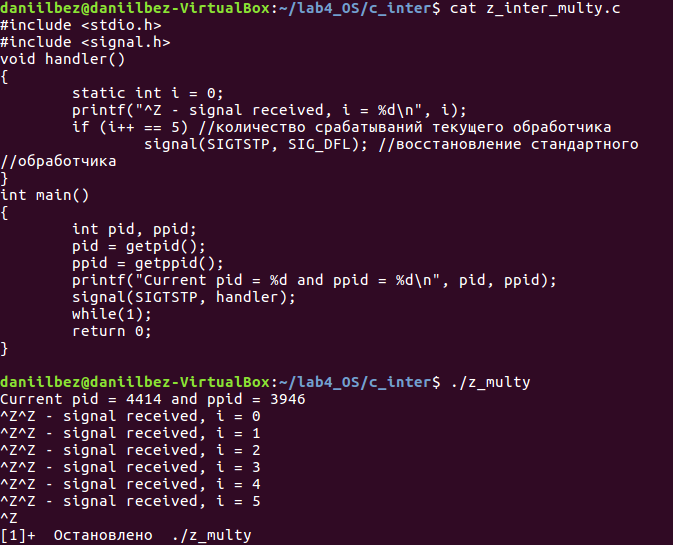


Рисунок 14 – текст программы со множественной обработкой сигнала и результат ее работы

6. С помощью утилиты kill выведем список сигналов и дадим краткую

характеристику.

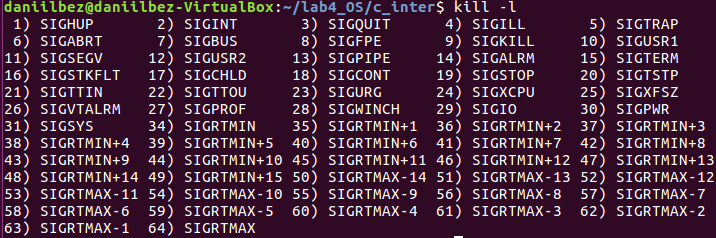


Рисунок 15 – список сигналов kill

* Процессы от 1 до 31 заданы стандартом *POSIX* и выполняют определенные функции. Например, завершение процесса (*SIGTERM*), неожиданное прерывание процесса (*SIGINT*), остановка процесса (*SIGSTOP*)
* Процессы 32 и 33 не отображаются – они используются POSIX-ядрами
* Сигналы с номерами от 34 и выше – переменные. *SIGRTMIN* и *SIGRTMAX* используются как минимальное и максимальное значение для этих сигналов (эти значения могут быть изменены). Их предназначение не обязательно определено, как говорилось раннее, они могут быть использованы при разработке. По умолчанию сигналы от 34 и выше завершают работу программы.

Попробуем послать данные сигналы фоновому процессу, посмотрим, что произойдет.

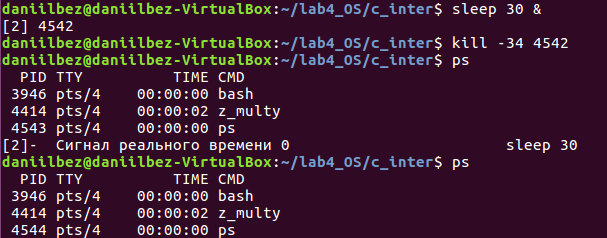
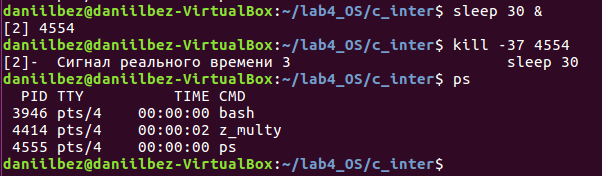
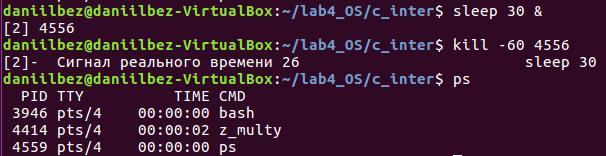


Рисунок 16 – результат использования SIGRTMIN

Нетрудно заметить, что после того, как мы послали данный сигнал, выполнение процесса прекратилось – подтверждает это вывод утилиты ps.

Приведем еще два примера – kill -37, kill -60

Результаты ожидаемы, и совпали с результатами первого эксперимента.

7. Проанализирована процедура планирования для процессов и

потоков одного процесса. Совершены попытки изменить процедуру планирования. Нитям заданы разные приоритеты.

