

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Автоматизированных систем управления
(АСУ)

Операционные системы

Обработка сигналов
Лабораторная работа 4

Выполнила:
Студентка гр. 3-433П8-5

Медведева Юлия Евгеньевна

«6» октября 2024г.
(дата)

Проверил:

_____ (должность, ученая степень, звание)	
_____ (подпись)	/ И. О. Фамилия
« ____ »	_____ 20 ____ г. (дата)

Томск 2024

Оглавление

Введение..... 3

Основная часть 4

 1. Процесс-отец.....4

 2. Процесс 14

 3. Процесс 2.....5

 4. Процесс 3.....5

Заключение 9

Введение

Целью выполнения настоящей лабораторной работы является получение навыков программного управления процессами с помощью сигналов. В ходе работы будут рассмотрены и отработаны следующие аспекты:

1. Синхронизация процессов с помощью сигналов: понятие сигнала, типы сигналов, выдача сигнала процессом.
2. Терминальное управление процессами: управляющий терминал, сеанс, группа процессов, получение информации о процессах с помощью команды shell *ps -j*, применение команды shell *kill* для посылки сигнала процессу.
3. Обработка сигналов: варианты обработки сигналов, диспозиция сигналов.
4. Применение таймера для управления процессами: таймер, аларм, таймер интервала.

Основная часть

В данной лабораторной работе был разработан программный код, в котором процесс-отец создает три дочерних процесса. Каждый из дочерних процессов выполняет бесконечный цикл и особым образом реагирует на сигналы SIGINT и SIGQUIT. Рассмотрим детали реализации каждого процесса и их взаимодействие с сигналами.

1. Процесс-отец

Процесс-отец создается и порождает три дочерних процесса с помощью вызова *fork()*. После создания всех дочерних процессов процесс-отец завершает свою работу.

```
pid = fork();
if (pid == 0) {
    proc1(); // Дочерний процесс 1
    exit(0);
}
```

Данный фрагмент кода показывает, что процесс-отец вызывает функцию *fork()*, и, если возвращаемое значение равно нулю, выполняется дочерний процесс. Точно так же создаются процессы 2 и 3. После создания всех дочерних процессов процесс-отец завершает свою работу с помощью *exit(0)*.

2. Процесс 1

Процесс 1 обрабатывает сигнал SIGINT, выводя сообщение на экран с указанием текущей даты и времени. При этом процесс защищен от воздействия сигнала SIGQUIT: данный сигнал просто игнорируется, что позволяет процессу продолжать выполнение без остановки. Обработка сигнала SIGINT производится с использованием функции *handler_sigint_proc1()*:

```
void handler_sigint_proc1(int signum) {
    time_t now = time(NULL);
    char *date_time = ctime(&now);
    printf("Process 1: Was get SIGINT signal. Current time: %s", date_time);
}
```

```
}
```

Процесс находится в бесконечном цикле с использованием функции *pause()*, ожидая поступления сигналов.

3. Процесс 2

Процесс 2 также реагирует на сигнал SIGINT, но его поведение отличается от процесса 1: при получении сигнала он выводит сообщение и продолжает выполнение. Обработка сигнала выполняется через функцию *handler_sigint_proc2()*:

```
void handler_sigint_proc2(int signum) {  
    printf("Process 2: Was get SIGINT signal. In progress...\n");  
}
```

Процесс 2, как и процесс 1, выполняет бесконечный цикл с использованием *pause()*.

4. Процесс 3

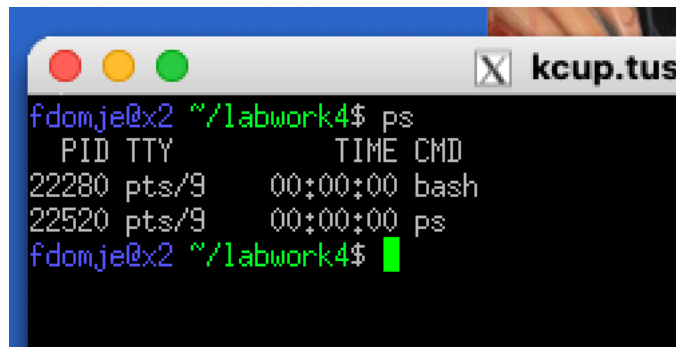
Процесс 3 защищен от воздействия сигнала SIGINT на уровне всего процесса. Блокировка осуществляется с помощью функции *sigprocmask()*, которая добавляет сигнал SIGINT в заблокированный набор:

```
sigset_t set;  
sigemptyset(&set);  
sigaddset(&set, SIGINT);  
sigprocmask(SIG_BLOCK, &set, NULL);
```

После блокировки сигнала процесс выполняет бесконечный цикл, ожидая другие сигналы. Поскольку SIGINT заблокирован, процесс 3 не реагирует на этот сигнал.

5. Демонстрация исполнения программы

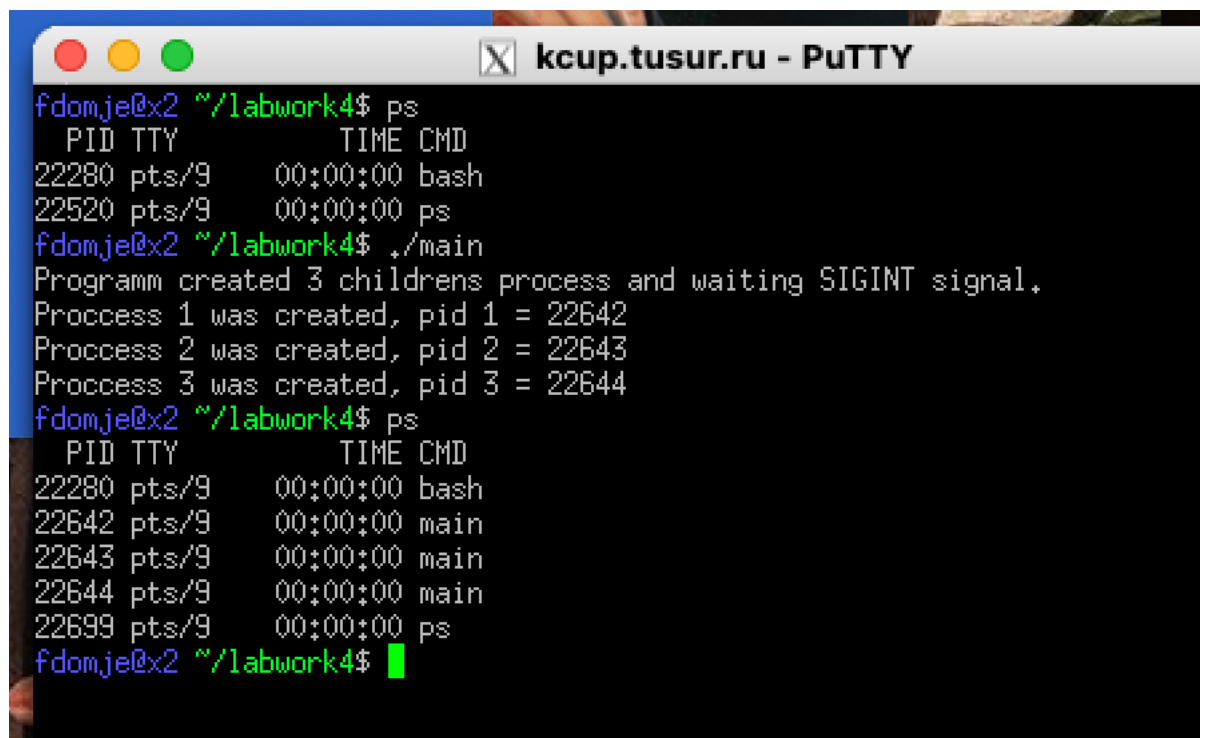
Перед запуском узнаем какие процессы запущены. В терминале выполним *ps*. Количество процессов и их PID демонстрирует рисунок 1.



```
fdomje@x2 ~/labwork4$ ps
  PID TTY          TIME CMD
 22280 pts/9        00:00:00 bash
 22520 pts/9        00:00:00 ps
fdomje@x2 ~/labwork4$
```

Рис.1 Процессы до исполнения кода

Сразу после вызова программы *main* отображается список созданных процессов. Для проверки данных вызовем снова *ps*. Это уже отражает рисунок 2.



```
fdomje@x2 ~/labwork4$ ps
  PID TTY          TIME CMD
 22280 pts/9        00:00:00 bash
 22520 pts/9        00:00:00 ps
fdomje@x2 ~/labwork4$ ./main
Programm created 3 childrens process and waiting SIGINT signal.
Proccess 1 was created, pid 1 = 22642
Proccess 2 was created, pid 2 = 22643
Proccess 3 was created, pid 3 = 22644
fdomje@x2 ~/labwork4$ ps
  PID TTY          TIME CMD
 22280 pts/9        00:00:00 bash
 22642 pts/9        00:00:00 main
 22643 pts/9        00:00:00 main
 22644 pts/9        00:00:00 main
 22699 pts/9        00:00:00 ps
fdomje@x2 ~/labwork4$
```

Рис.2 Результат работы кода

Тестируем первый процесс. Ожидаемый результат: при вводе команды *kill -SIGINT <PID>* появится сообщение с текущим временем. Фактический результат на рисунке 3.

```

fdomje@x2 ~/labwork4$ ps
  PID TTY          TIME CMD
 22280 pts/9        00:00:00 bash
 22642 pts/9        00:00:00 main
 22643 pts/9        00:00:00 main
 22644 pts/9        00:00:00 main
 22699 pts/9        00:00:00 ps
fdomje@x2 ~/labwork4$ kill -SIGINT 22642
fdomje@x2 ~/labwork4$ Process 1: was get SIGINT signal. Current time: Wed Oct 16
12:39:09 2024

```

Рис.3 Результат работы сигнала для первого процесса.

Рисунок 4 иллюстрирует наличие процесса после.

```

fdomje@x2 ~/labwork4$ kill -SIGINT 22642
fdomje@x2 ~/labwork4$ Process 1: was get SIGINT signal. Current time: Wed Oct 16
12:39:09 2024
^C
fdomje@x2 ~/labwork4$ ps
  PID TTY          TIME CMD
 22280 pts/9        00:00:00 bash
 22642 pts/9        00:00:00 main
 22643 pts/9        00:00:00 main
 22644 pts/9        00:00:00 main
 23065 pts/9        00:00:00 ps
fdomje@x2 ~/labwork4$

```

Рис.4 Процессы после первого тест кейса

Тестирование второго процесса. Ожидаемый результат: при вводе команды `kill -SIGINT <PID>` появится сообщение «In progress». И снова проверим, что процесс существует. Фактический результат на рисунке 5.

```

fdomje@x2 ~/labwork4$ kill -SIGINT 22643
Process 2: was get SIGINT signal. In progress...
fdomje@x2 ~/labwork4$ ps
  PID TTY          TIME CMD
 22280 pts/9        00:00:00 bash
 22642 pts/9        00:00:00 main
 22643 pts/9        00:00:00 main
 22644 pts/9        00:00:00 main
 23364 pts/9        00:00:00 ps
fdomje@x2 ~/labwork4$

```

Рис. 5 Результат второго тест кейса

Тестирование третьего процесса. Ожидаемый результат: процесс игнорирует вводе команды `kill -SIGINT <PID>`. Так же проверим, что процесс существует. Фактический результат на рисунке 6.

```
fdomje@x2 ~/labwork4$ kill -SIGINT 22644
fdomje@x2 ~/labwork4$ ps
  PID TTY          TIME CMD
 22280 pts/9        00:00:00 bash
 22642 pts/9        00:00:00 main
 22643 pts/9        00:00:00 main
 22644 pts/9        00:00:00 main
 23504 pts/9        00:00:00 ps
```

Рис. 6 Тестирование третьего процесса

Заключение

В данной лабораторной работе были рассмотрены и реализованы различные способы обработки сигналов в дочерних процессах, созданных процессом-отцом. Первый дочерний процесс реагировал на SIGINT, добавляя к сообщению текущее время, второй - продолжал выполнение после получения SIGINT, а третий был полностью защищен от данного сигнала.

Проведение данной лабораторной работы позволило приобрести практические навыки работы с сигналами в Unix-подобных операционных системах. Я научилась создавать процессы, управлять их обработкой сигналов, а также блокировать и игнорировать сигналы. Эти навыки необходимы для разработки более надежных и устойчивых приложений, которые корректно реагируют на внешние воздействия и сигналы.