Лабораторная работа №2

студента группы ИТ-222

Мокрищева Николая Павловича

Выполнение:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Защита: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Процессы в операционной системе UNIX**

Цель работы: ознакомление с процессами в операционной системой UNIX.

**Содержание работы**

Вариант №7

1. Текст программы, печатающей значения PID и PPID для текущего процесса. Копия экрана, подтверждающего правильность выполнения программы.

2. Текст программы 03-1.с создания нового процесса с одинаковой работой процессов ребенка и родителя. Копия экрана, подтверждающего правильность выполнения программы.

3. Текст программы создания нового процесса с различной работой процессов ребенка и родителя. Копия экрана, подтверждающего правильность выполнения программы.

4. Текст программы, распечатывающей аргументы командной строки и параметры среды. Копия экрана, подтверждающего правильность выполнения программы.

5. Тест программы 03-2.с, изменяющая пользовательский контекст процесса. Копия экрана, подтверждающего правильность выполнения программы.

6. Текст программы для изменения пользовательского контекста в порожденном процессе. Копия экрана, подтверждающего правильность выполнения программы.

7. Вывод.

**Ход работы**

Вариант №7

1. Чтобы вывести PPID и PID используется системный вызов getppid и getpid (Рисунок 1, 2).

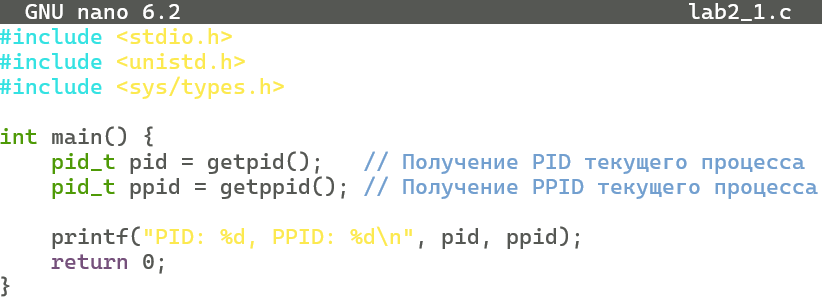


Рисунок 1. Текст программы, печатающее значение PPID и PID

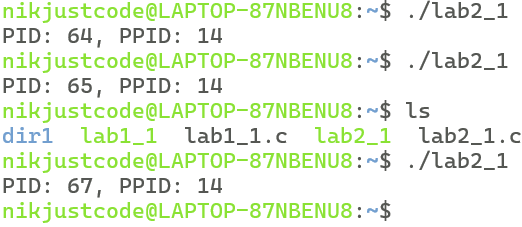


Рисунок 2. PPID и PID процессов

2. Чтобы создать новый процесс используется системный вызов fork, который единожды порождает новый процесс с тем же исходным кодом (Рисунок 3, 4).

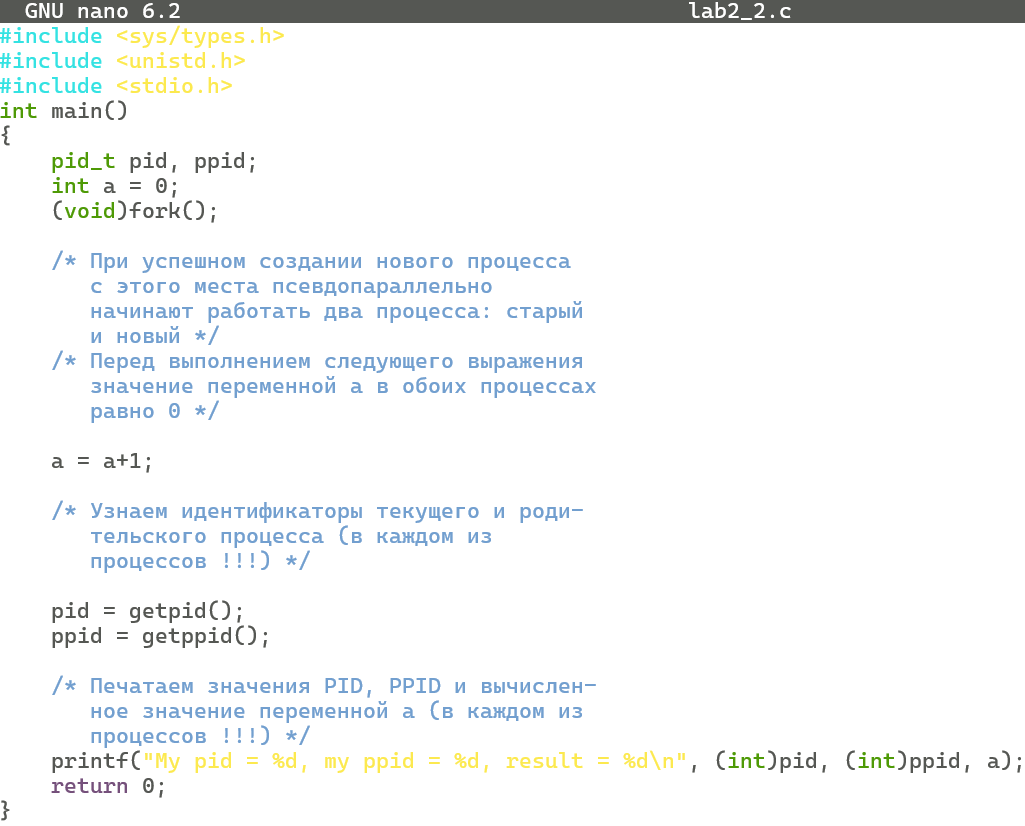


Рисунок 3. Текст программы 03-1.с

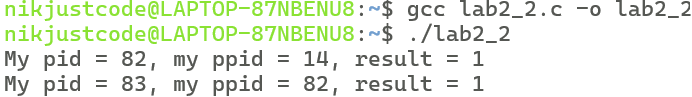


Рисунок 4. Результат работы программы с созданием нового процесса с одинаковой работой

3. Для того, чтобы сделать процессы с разной работой, нужно, чтобы в самом процессе проверялся pid. Если fork выдало pid > 0 => процесс является родительским и породило новый процесс, если же pid = 0, то процесс является ребёнком (Рисунок 5, 6).

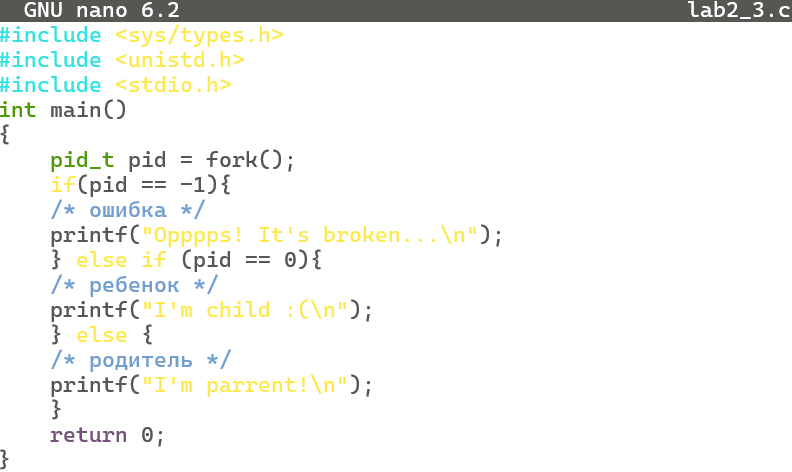


Рисунок 5. Текст программы создания нового процесса с различной работой

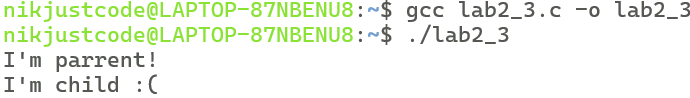


Рисунок 6. Результат работы программы создания нового процесса с различной работой

4. Функция main имеет 3 параметра, используя их можно использовать аргументы командной строки и так же работать с параметрами среды. К примеру, можно их вывести (Рисунок 7, 8, 9).

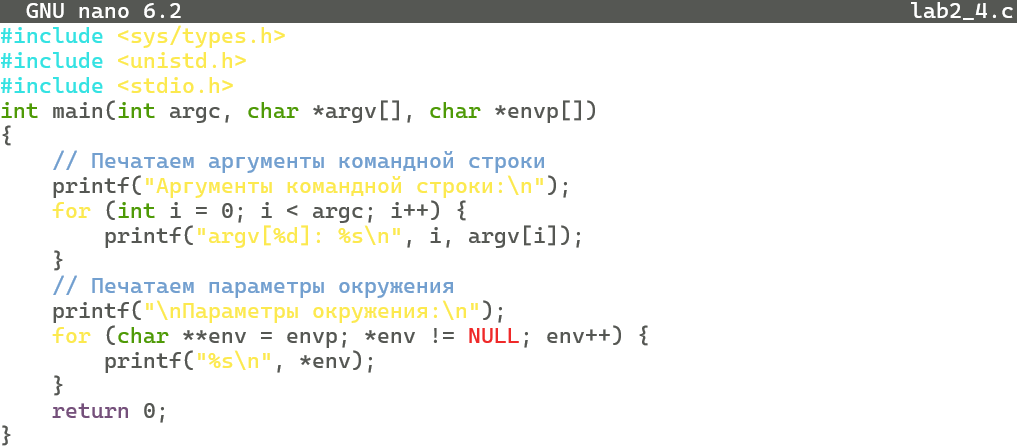


Рисунок 7. Текст программы, распечатывающей аргументы командной строки и параметры среды

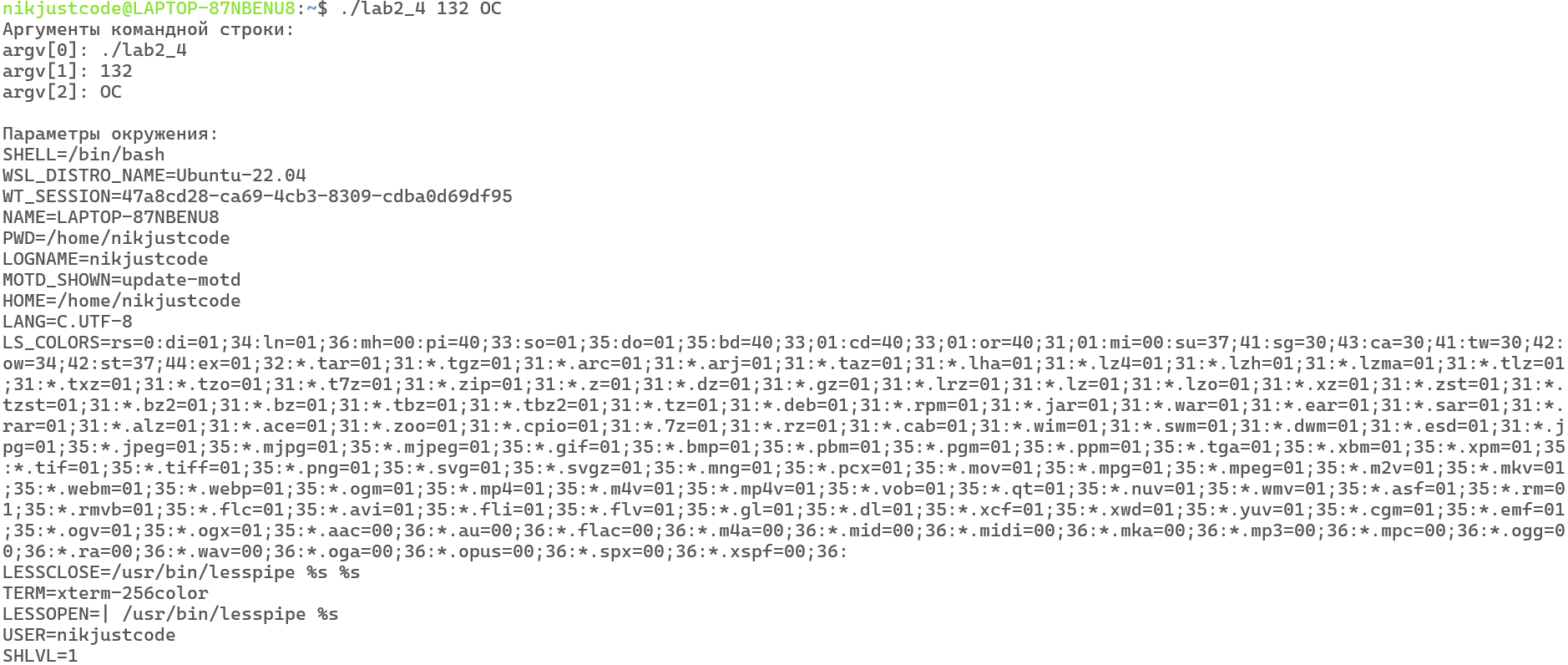


Рисунок 8. Начало вывода аргументов командной строки и параметров среды

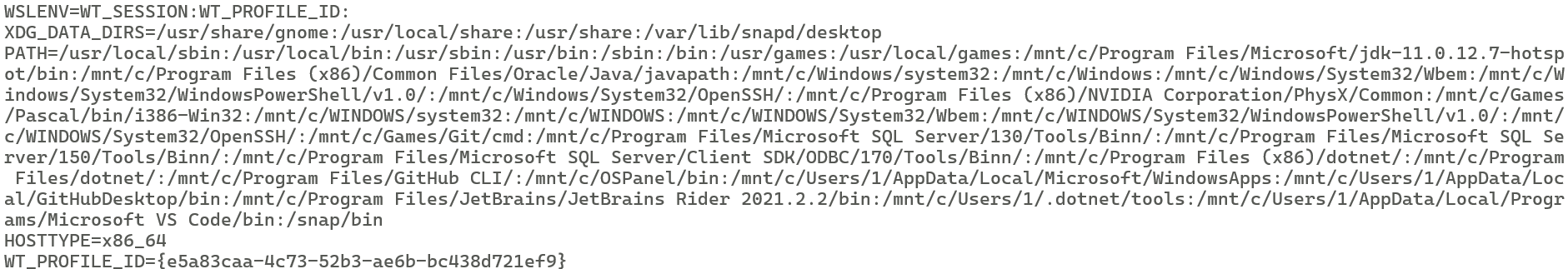


Рисунок 9. Конец вывода параметров среды

5. При помощи функции execle, execv и подобных можно загрузить новую программу в системный контекст текущего процесса, что позволить выполнить другую программу из другого процесса (Рисунок 10, 11).

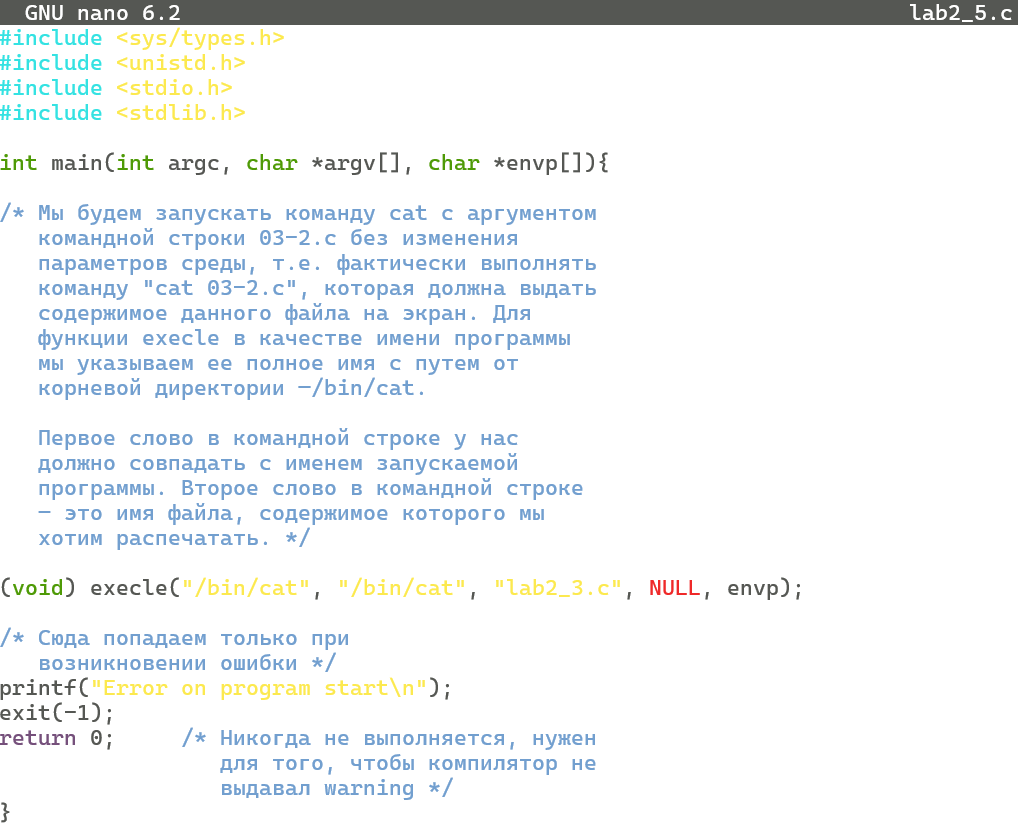


Рисунок 10. Тест программы 03-2.с

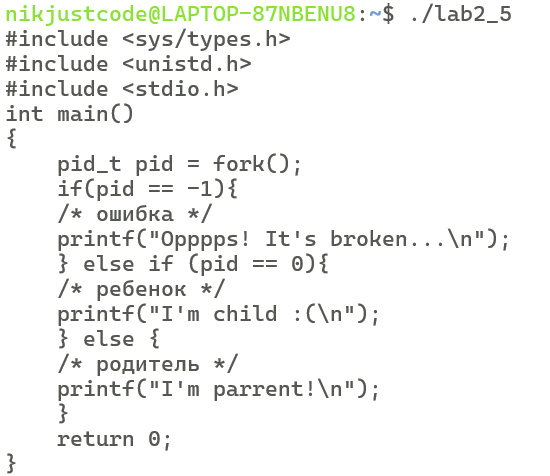


Рисунок 11. Вывод программы 03-02.с

6. Используя функции execle, execv и т.д. в связке с проверкой pid из задания 3 можно делать программы, который меняют пользовательский контекст в порождённых программах, чтобы запустить другие программы из процесса (Рисунок 12, 13).



Рисунок 12. Текст программы для изменения пользовательского контекста в порожденном процессе

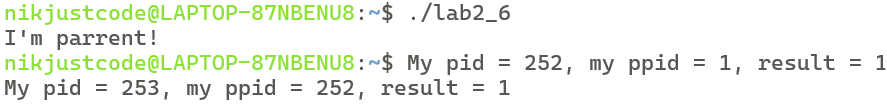


Рисунок 13. Результат работы программы для изменения пользовательского контекста в порожденном процессе

7. Вывод: в ходе проделанной работы была изучена работа процессов в операционной системе UNIX. В операционной системе UNIX процессы играют ключевую роль, и их структура включает два контекста: **пользовательский** и **системный**. Пользовательский контекст включает код и данные процесса, такие как константы, инициализируемые и неинициализируемые переменные, стек и динамическую память. Системный контекст (ядра) включает такие элементы, как идентификаторы процесса (PID), пользователя (UID), группы (GID), а также стек ядра и другие системные данные.

Каждый процесс в UNIX имеет уникальный идентификатор (PID). Процессы связаны через иерархию родительских и дочерних процессов. Если родительский процесс завершает свою работу до дочернего, процесс-родитель "усыновляется" процессом с идентификатором 1 — **init**.

Процессы в UNIX могут находиться в различных состояниях (готовность, выполнение, ожидание и завершение), при этом важно различие между режимами работы процесса в **пользовательском** и **режиме ядра**. Важную роль играют системные вызовы **getpid()** и **getppid()**, которые возвращают идентификаторы текущего и родительского процессов соответственно.

Создание новых процессов происходит через вызов **fork()**, при этом новый процесс — это почти полная копия родительского, с измененными PID и PPID. После вызова **fork()** оба процесса продолжают выполнение, и их различное поведение можно организовать через проверку возвращаемого значения. Процесс завершается с использованием функции **exit()**, которая позволяет корректно завершить процесс и передать системе код завершения.

В языке C функция **main()** может принимать аргументы командной строки и параметры окружения. Аргументы командной строки позволяют программе изменять свое поведение при каждом запуске, а параметры окружения обеспечивают долгосрочные настройки, такие как переменная **TERM**, которая определяет тип терминала.

Таким образом были изучены ключевые механизмы работы процессов в UNIX, системные вызовы для их создания и завершения, а также способы взаимодействия процессов с командной строкой и окружением.