Лабораторная 4.

Задание:разработайте приложение, запускающее несколько программ. Определите идентификаторы соответствующих процессов. Установите родственные связи между ними.
Цель:получение навыков использования функций API создания процессов на платформе Linux

Для разработки приложения, которое запускает несколько программ и определяет идентификаторы соответствующих процессов в Linux, необходимо использовать системные вызовы fork() и exec(). Эти функции позволяют создавать дочерние процессы и запускать в них новые программы.

1. Для создания нового процесса используется системный вызов fork(). Этот вызов создает копию текущего процесса (родителя), и оба процесса (родитель и потомок) продолжают выполнение с точки, где был вызван fork().

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
  pid_t pid = fork();

  if (pid < 0) {
    perror("Ошибка fork");
    exit(EXIT_FAILURE);
  } else if (pid == 0) {
    printf("Дочерний процесс с PID: %d\n", getpid());
    exit(EXIT_SUCCESS);
  } else {
    printf(
        "Родительский процесс с PID: %d, создал дочерний процесс с PID:
%d\n",
        getpid(), pid);
  }

  return 0;
}
```

2. После создания дочернего процесса, можно использовать функции из семейства exec для запуска других программ, например, execl() или execvp().

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
  pid_t pid = fork();
```

```
  if (pid < 0) {
    perror("Ошибка fork");
    exit(EXIT_FAILURE);
  } else if (pid == 0) {
    execl("/bin/ls", "ls", "-l", NULL);
    perror("Ошибка exec");
    exit(EXIT_FAILURE);
  } else {
    printf(
        "Родительский процесс с PID: %d, создал дочерний процесс с PID:
%d\n",
        getpid(), pid);
    wait(NULL);
  }

  return 0;
}
```

3. Каждый созданный дочерний процесс имеет уникальный идентификатор - PID, который можно использовать для отслеживания их состояния. Родительский процесс может ожидать завершения своих дочерних процессов с помощью функции wait().

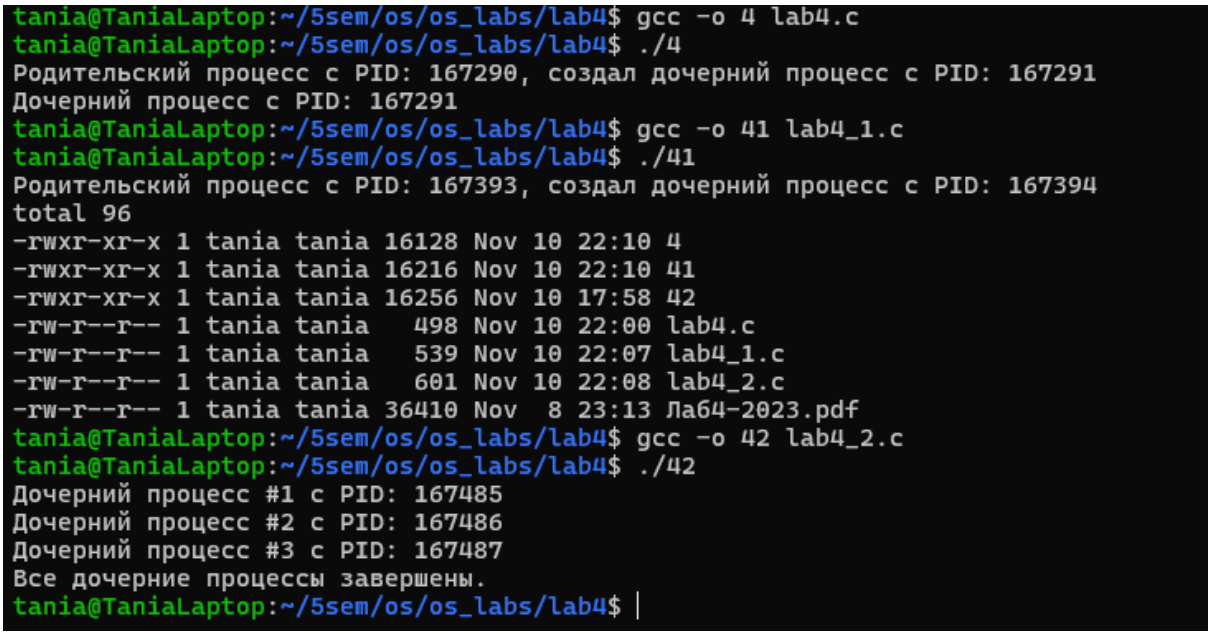
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
  for (int i = 0; i < 3; i++) {
    pid_t pid = fork();

    if (pid < 0) {
      perror("Ошибка fork");
      exit(EXIT_FAILURE);
    } else if (pid == 0) {
      printf("Дочерний процесс #%d с PID: %d\n", i + 1, getpid());
      execl("/bin/sleep", "sleep", "2", NULL);
      perror("Ошибка exec");
      exit(EXIT_FAILURE);
    }
  }
  for (int i = 0; i < 3; i++) {
    wait(NULL);
```

```
    }
    printf("Все дочерние процессы завершены.\n");
    return 0;
}
```

Результат работы программ:



Заключение:

Используя системные вызовы fork() и exec(), можно эффективно создавать и управлять несколькими процессами в Linux. Каждый созданный процесс имеет свой уникальный идентификатор, который позволяет отслеживать его состояние и устанавливать родственные связи между ними.

Используемые источники:

https://habr.com/ru/articles/447182/