Лабораторная работа №3: Многозадачное программирование в Linux

Цель:

Изучить создание процессов с помощью fork(), их синхронизацию с wait()/waitpid(), завершение процессов и обработку сигналов.

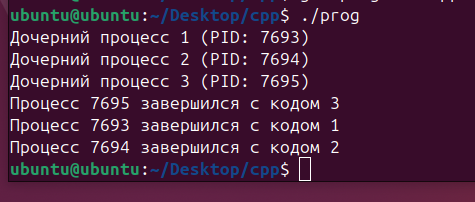
Задание:

1. Создание процессов:

Напишите программу, которая создает несколько дочерних процессов. Каждый дочерний процесс должен выполнять свою задачу (например, выводить уникальное сообщение). Родительский процесс должен дождаться завершения всех дочерних процессов и вывести их коды завершения.

1. Обработка сигналов:

Модифицируйте программу так, чтобы родительский процесс перехватывал сигнал SIGCHLD и асинхронно обрабатывал завершение дочерних процессов.



#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <signal.h>

void sigchld\_handler(int sig) {

int status;

pid\_t pid = waitpid(-1, &status, WNOHANG);

if (pid > 0) {

printf("Процесс %d завершился (асинхронно)\n", pid);

}

}

int main() {

pid\_t pid;

int status;

for (int i = 0; i < 3; i++) {

pid = fork();

if (pid == 0) {

// Дочерний процесс

printf("Дочерний процесс %d (PID: %d)\n", i+1, getpid());

sleep(1);

exit(i+1); // Завершаем с кодом i+1

}

}

// Родительский процесс ожидает завершения всех потомков

while ((pid = wait(&status)) != -1) {

if (WIFEXITED(status)) {

printf("Процесс %d завершился с кодом %d\n", pid, WEXITSTATUS(status));

}

}

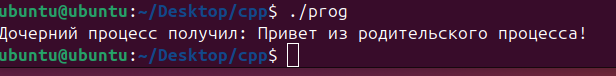
return 0;

}

**Лабораторная работа №4: Каналы передачи данных**

1. Неименованные каналы (pipe):

Напишите программу, где родительский процесс создает канал, порождает дочерний процесс, и они обмениваются сообщениями через канал.



#include <stdio.h>

#include <unistd.h>

#include <string.h>

int main() {

int fd[2];

char buf[100];

pipe(fd); // Создаем канал

if (fork() == 0) {

// Дочерний процесс читает из канала

close(fd[1]); // Закрываем запись

read(fd[0], buf, sizeof(buf));

printf("Дочерний процесс получил: %s\n", buf);

close(fd[0]);

} else {

// Родительский процесс пишет в канал

close(fd[0]); // Закрываем чтение

strcpy(buf, "Привет из родительского процесса!");

write(fd[1], buf, strlen(buf)+1);

close(fd[1]);

}

return 0;

}