Лабораторная работа #10

Основы многозадачности в ОС Linux.
Процессы в Linux

[pid, ppid, init, fork, exec, wait, nice, system]

by Viktor Ivanchenko

ЛАБОРАТОРНАЯ РАБОТА #10
Основы многозадачности в ОС Linux. Процессы в Linux

Цель работы

Изучить фундаментальные концепции многозадачности в ОС, а также научиться порождать и управлять многозадачными приложениями в ОС Linux.

Требования

1) На базе полученных знаний разработать модульное консольное приложение на C/C++ согласно варианту задании A и B с использованием архитектурного шаблона проектирования MVC.
2) АПРЕЩАЕТСЯ в программе использовать под любым предлогом ГЛОБАЛЬНЫЕ переменные!
3) Для повышения производительности программы и закрепления навыков работы с памятью везде, где это возможно, необходимо использовать ДИНАМИЧЕСКОЕ выделение и освобождение памяти, а также осуществлять работу через УКАЗАТЕЛИ.
4) Все функции должны быть сгруппированы по соответствующим отдельным файлам и вынесены в отдельную библиотеку.
5) Все функции должны быть самодостаточные, т.е. при их разработке необходимо придерживаться принципа ***Single Responsibility Principle***.
6) Для компиляции, компоновки и выполнения программы использовать средства автосборки (к примеру, ***Makefiles*** и утилиту **make**).
7) При разработке программ придерживайтесь соглашений по написанию кода на C/C++ (Code-Convention).
8) Контрольные вопросы по лабораторной работе и ответы на них должны быть записаны в конспект.

Основное задание

Приложение, разработанное при выполнении лабораторной работы № 8 (или предыдущих лабораторных работ), запустить как дочерний процесс из другого родительского процесса.

Родительский процесс должен:

- передавать соответствующие данные (к примеру, целевые строки) дочернему процессу;
- получать результат выполнения дочернего процесса;
- дожидается завершение дочернего процесса, а затем выводить результат выполнения дочернего процесса в терминал и/или в файл;
- перед запуском дочернего процесса устанавливать приоритет для соответствующего запускаемого дочернего процесса, а также иметь возможность привязать выполнение дочернего процесса на конкретном ядре процессора ПК.

Дочерний процесс должен выполнять основную бизнес-логику.

Дополнительное задание

1) Написать программу, которая будет превращать в демона сама себя.
2) Написать программу-фильтр, которая читает из стандартного потока ввода и пишет в свой стандартный поток вывода, выполняя при этом полезное преобразование.
3) Необходимо написать программу, которая выводит список выполняемых текущим пользователем процессов, а также процессов, выполняемых другими пользователями и системных процессов (или всех процессов в системе);
4) Написать программу *mini-shell* (эмулятор *shell*), которая создаёт процессы, которые могут параллельно выполнять Linux-команды с поддержкой возможности ввода аргументов командной строки.

Best of LUCK with it, and remember to HAVE FUN while you're learning :)

Victor Ivanchenko

Контрольные вопросы

1. Основы многозадачности *(multitasking)* и мультипрограммирования в системах пакетной обработки, в системах разделения времени и в системах реального времени? Мультипроцессорная обработка?
2. Понятие процесса как второй фундаментальной абстракции в Linux-системах, основные концепции, поддержка процессов ядром ОС Linux? Древовидная иерархия процессов в Linux, процесс ***init***, Linux-команды для мониторинга процессов и просмотра их иерархии?
3. Планирование процессов и их приоритет? Уступчивость процесса и использование системного вызова ***nice***?
4. Структура данных Linux-процесса, идентификатор и атрибуты процесса? Способы получения информации о процессе? Использование соответствующих системных вызовов? Как получить имя пользователя из его числового имени, т.е. из ***UID***?
5. Какие бывают типы процессов (порождающий (родитель), порождённый (потомок), демон, зомби, приведение)?
6. Состояние процесса, время выполнения процесса, понятие очереди обработки процессов? Подходы, используемые для обработки очередей: квантование, приоритет, смешанный?
7. Как осуществить порождение дочернего процесса? Концепция развилки, создание (ветвление) и исполнение новых процессов (передача управления)? Использование системных вызовов ***fork*** *(****vfork****)* и семейства вызовов ***exec*** *(****execl****, ****execlp****, ****execle****, ****execv****, ****execvp****, ****execve****)?*
8. Как осуществить ожидание выполнения дочернего процесса? Использование системных вызовов ***wait*** и ***waitpid***?
9. Как завершить выполнение процесса? Использование системного вызова ***_exit*** и библиотечной С-функции ***exit***? Что произойдет с работающим потомком при завершении родительского процесса?
10. В каких целях используют библиотечную функцию ***system***? Как она работает?