Практическое задание №2

Самостоятельно выберите вариант задания, выполните его и сдайте в машзале. Можно сделать несколько вариантов задания.

Во всех задачах необходимо позаботиться об отсутствии «зомби», не использовать активное ожидание. Если речь идет о файлах, то нельзя предполагать, что их содержимое целиком поместится в памяти. Все общие требования к программам, сформулированные в практическом задании №1, актуальны и для этого задания.

Вариант 1. Напишите программу, которая каждые N секунд передает на стандартный вывод количество тех и только тех строк файла F, в которых встречается подстрока T. Программа завершается с кодом 0 получением сигнала SIGTERM. Параметры N, F, T передаются через командную строку. Программа не должна сама читать файл, вместо этого она должна использовать системные утилиты (grep, wc и другие). Считать, что эти утилиты успевают проработать за N секунд.

Вариант 2. Напишите программу, которая реализует следующую команду шелла: (pr1 arg1 > f) && (pr2 | pr3 args…) . Аргументы командной строки вашей программы такие: pr1 arg1 f pr2 pr3 args… (args – это все остальные аргументы). Продемонстрируйте работоспособность вашей программы. Если отцовский процесс получает сигнал SIGTERM, он должен послать его всем запущенным им процессам.

Вариант 3. Напишите несколько программ.

Программа 1 получает в качестве аргумента командной строки путь к файлу и оставляет в нем только те строки, которые не начинаются с символа ‘#’. Если используются промежуточные файлы, то их имена не должны остаться в системе.

Программа 2 получает в качестве аргумента командной строки путь к файлу, каждая строка которого – путь к другому файлу. В каждом из них должны быть удалены строки, начинающиеся с символа ‘#’, при помощи программы 1. Обработка должна быть параллельной, но одновременно должны быть запущены не более N процессов (N тоже задается в командной строке) (при завершении дочернего процесса ядро ОС посылает родительскому процессу сигнал SIGCHLD).

Вариант 4. Напишите программу для параллельного умножения двух матриц. Программа получает в командной строке имена трех файлов A, B, C. Файл с матрицей содержит ее размерности и все элементы. Размерности могут быть достаточно большими. Программа должна разместить в файле C матрицу, равную A * B, если это возможно. Умножение матриц должно быть распараллелено. Используйте асинхронность при помощи сигналов (для уведомления процессов) и каналов (для передачи полезной информации). Опишите проведенные вами эксперименты по получению значений различных констант в вашей программе, дающих наилучшую производительность.

Вариант 5. Напишите программу, при помощи которой можно эффективно вычислять мультипликативные выражения из матриц. Эти выражения состоят из матриц, операций умножения и скобок. Матрицы хранятся в файлах. Для повышения эффективности используйте параллелизм и асинхронное выполнение. Для понижения сложности программы можно выделить умножение двух матриц в отдельную программу (см. предыдущий вариант) и параллельно вычислять различные части выражения.

Вариант 6. Напишите программу, имитирующую подсистему печати. Есть процессы-принтеры (имитируют работу устройства-принтера). Для каждого процесса-принтера есть процесс-контроллер. Процесс-принтер получает данные от своего контроллера и выводит их, возможно с некоторой обработкой. Процесс-контроллер упорядочивает задания печати и отправляет их на принтер, не допуская перемешивания частей заданий между собой. Процессы, которые хотят напечатать, взаимодействуют с соответствующим контроллером, передавая ему свои задания. У контроллеров можно узнать текущую очередь заданий и информацию о каждом из них. Кроме того, должна быть возможность встраивания в работу контроллеров без их перезапуска (например, добавить водяной знак при печати), причем контроллер не должен знать заранее, какой процесс будет встраиваться и что он делает. Все процессы не родственные. При отсутствии задач процессы должны засыпать. Процессы завершаются с кодом 0 при получении SIGTERM. Процессы-принтеры и контроллеры не должны читать свой стандартный ввод.

Вариант 7. Эксперименты с сервис-ориентированной архитектурой. Идея в том, что программа, которую нужно написать, разбивается на набор компонентов. Каждый компонент решает некоторую определенную задачу («предоставляет определенный сервис»). Компоненты взаимодействуют друг с другом для решения исходной задачи.

Если одновременно работают несколько программ, которые используют одни и те же компоненты (целиком или частично), то эффективнее будет иметь каждый компонент в единственном экземпляре. Он будет выполнять запросы от других компонентов и сам при необходимости запрашивать сервис от других компонентов. Примеры сервисов: база данных, пользовательский интерфейс, логирование, почта.

Выберите набор сервисов и набор информационных систем, использующих эти сервисы. Реализуйте каждый сервис в виде процесса-демона. Четко опишите способ получения сервиса (протокол взаимодействия), это позволит заменять один процесс, реализующий сервис, на другой без замены кода процесса, использующего сервис. Реализовать и продемонстрировать решение некоторой задачи в разработанной сервис-ориентированной архитектуре. Опишите все принятые решения, их реализацию, проведенные эксперименты в виде отчета по заданию.