Операционные системы и среды
Л.р.5. Управление процессами и взаимодействие процессов.

Цель:

Изучение основных особенностей подсистемы управления процессами и средств взаимодействия процессов в Unix.
Практическое проектирование, реализация и отладка программных комплексов из нескольких взаимодействующих процессов.

Теоретическая и методическая часть

Порождение процессов: *fork*, *exec*, *fork-exec*
Завершение процессов, код завершения
Сигналы Unix (надежные, ненадежные)
«Базовые» средства IPC: каналы, семафоры, сообщения, разделяемая память.
Типичные задачи, проблемы, подходы к решению, модели организации взаимодействия процессов.

Практическая часть

Общая постановка задачи:

Написать программу (программы) в соответствии с вариантом задания. Спланировать и обеспечить тестирование (демонстрацию) выполнения – для нескольких взаимодействующих процессов это может быть существенно более сложно и трудоемко.

Желательно продолжать использовать *make* (и сценарии *makefile*) для управления обработкой проекта.

Многие из вариантов имеют сходство с заданиями по дисциплине СП.

Варианты заданий:

1) Процесс-демон – протоколирование сигналов
2) Самовосстанавливающийся процесс
3) Распределенная обработка (потоки или пакеты данных)
4) Распределенная обработки (данные в разделяемой памяти)
5) «Параллельный» анализатор файловой системы
6) Процесс-демон – анализатор файловой системы
7) Процесс – сервер с многопользовательским доступом
8) Реализация модели взаимодействия параллельных процессов
9) ...

1 Процесс-демон – протоколирование сигналов

Процесс, преобразующийся в «демон» и выполняющий единственную функцию: прием и протоколирование (запись в файл) заданных сигналов.

Список протоколируемых сигналов задается файлом конфигурации, считывается процессом при запуске и по сигналу SIGHUP.

Сигналы SIGHUP и SIGTERM сохраняют свой эффект для процесса (реконфигурация и завершение соответственно), но также могут и протоколироваться наравне с прочими.

Опционально: предусмотреть возможность выгрузки демона из памяти по команде, передаваемой ему при запуске исполняемого файла с соответствующей опцией, например ***`mydaemon -q`*** (или **`--quit`** и т.п.). Для связи нового («транзитного») процесса с демоном можно использовать, например, тот же сигнал SIGTERM, т.е. консольный вызов служит только в качестве «интерфейса пользователя».

2 Самовосстанавливающийся процесс

Процесс, который при получении сигнала, стандартно вызывающего завершение, создает свою копию, которая продолжает выполняться с прерванного места, и лишь после этого завершается, избегая таким образом безусловного «уничтожения» неперехватываемым сигналом.

Примечание: в Unix-системах, в отличие от Windows, реализация такой возможности достигается гораздо легче и естественнее.

В качестве демонстрации «живости» процесса и его выполнения можно использовать произвольные действия, повторяющиеся периодически и дающие заметный результат. Для консольных приложений это может быть, например, счетчик, значение которого обновляется с заданной частотой и записывается в файл.

3 Параллельная обработка (потоки или пакеты данных)

Построение в целом соответствует схеме «агент-менеджер».

Процесс-«менеджер»:
– получает (или генерирует) задание;
– порождает процессы-«агенты» и интерфейсы для взаимодействия с ними;
– декомпозирует задание на фрагменты (подзадания) и раздает их «агентам»;
– принимает от «агентов» частичные результаты и собирает из них итоговый;
– ведет учет подзаданий и «агентов».

Процессы-«агенты» (копии процесса-«менеджера, выполняющиеся по другой ветви алгоритма, или отдельные исполняемые файлы):
– принимают от «менеджера» фрагменты заданий;
– выполняют свои подзадания;
– возвращают «менеджеру» частичные результаты.

Реализуемый алгоритм обработки – произвольный, достаточно трудоемкий и удобный для распараллеливания: вычисления над массивами (матрицами), сортировка, криптография и т.д.

4 Параллельная обработка (данные в разделяемой памяти)

Аналогично предыдущему, но совместно обрабатываемые данные размещаются в общей разделяемой памяти и передаются между участниками по ссылке, номеру блока памяти и т.п.
Необходимо решить задачу контроля свободности-занятости блоков и предотвращения «столкновений».

Предположительно, потребуется деление общего массива памяти на несколько блоков, каждый из которых может быть в конкретный момент времени занят одним набором данных и использоваться одним процессом.

5 Параллельный анализатор файловой системы

Одна из типичных задача «системных» программных средств – мониторинг состояния файловой системы и отдельных файлов в ней. Это можно организовать, настроив события («триггеры» на изменения в файлах и директориях (будем рассматривать это как отдельную задачу) или сравнивая файлы с «эталоном». В последнем случае для минимизации объема хранимых эталонных данных сравнивается не содержимое файлов, а их сигнатуры.

Пусть задача упрощена до вычисления и сбора сигнатур файлов в заданных директориях. В конечном итоге должен быть сформирован список записей вида: ***<pathname>-<signature>-<timestamp>***.
Будем считать эту задачу достаточно трудоемкой, чтобы ожидать пользы от распараллеливания.

Процесс-«менеджер», собирающий список сигнатур, обходит дерево каталогов и, встречая заданные для проверки директории, запускает в них процесс-«агент», который непосредственно вычисляет сигнатуры файлов.
Предварительно создается интерфейс «агента». Посредством этого интерфейса «агент» возвращает менеджеру записи для списка, «менеджер» собирает общий список и выводит его в файл или поток. В качестве интерфейса могут быть, например, каналы или очереди сообщений.

Надо также предусмотреть наглядную демонстрацию распараллеливания, например путем добавления к записям списка идентификаторов (порядковых номеров) приславших их «агентов».

Сигнатуры – в простейшем случае сумма – логическая «по модулю 2» (XOR) или арифметическая (с игнорированием переполнения и переноса) – всех слов файла и, возможно, некоторых его атрибутов.

6 Процесс-демон – анализатор файловой системы

Аналогично предыдущему варианту, но вместо ручного запуска «стартовый» процесс загружается как демон и периодически инициирует проверку, записывая результаты в файл.
Для сокращения объема выполняемой им работы файловая система контролируется избирательно – список проверяемых каталогов в файле конфигурации (обработка сигнала реконфигурирования SIGHUP).

7 Процесс – сервер с многопользовательским доступом

Задача, противоположная «распределенной» параллельной обработке: процесс-«сервер» принимает запросы (задания) от множества процессов-«клиентов», выполняет их и возвращает результаты «заказчикам». Сервер должен организовать одним из способов обработку запросов от нескольких клиентов одновременно.

В качестве интерфейса между сервером и клиентами наиболее подходящими представляются каналы (FIFO) или очереди сообщений (MQ), но можно воспользоваться и иными «локальными» IPC.

Надо предусмотреть наглядное отображение хода выполнения, например протоколируя в файл выполняемые шаги с временнЫми метками и идентификаций источников запросов.

8 Реализация модели взаимодействия параллельных процессов

Одна из моделей взаимодействия параллельных (конкурирующих) процессов, причем процессы (участники взаимодействия в модели) моделируются непосредственно процессами. Т.е. надо построить систему взаимодействующих по определенным правилам процессов и снабдить ее средствами управления и наблюдения результатов.

Результаты могут представлять собой, например, протокол событий, происходящих в модели и простейшую статистику состояний: счетчики обработанных заявок и отказов, процент времени простоя и занятости, и т.д. Результаты собираются по нескольких «сеансам» моделирования, в т.ч. и варьируя параметры модели.