Лабораторная работа №2
Средства межпроцессного взаимодействия ОС UNIX/Linux: сообщения. Барьерная синхронизация.

Теоретическая часть

Основной полезный эффект, получаемый в результате параллельного программирования, связан с асинхронной обработкой информации множеством независимых, параллельно выполняющихся процессов. Однако в ряде случаев бывает необходимо, чтобы параллельные процессы обменивались информацией между собой, передавая друг другу данные и сообщения о событиях. Кроме того, поскольку не исключены ситуации, когда несколько процессов обращаются к одному и тому же ресурсу (аппаратному, программному, информационному), необходимо установить порядок получения доступа к ресурсам.

Так возникает задача синхронизации, которая состоит в обеспечении взаимодействия параллельных процессов.

Введем ряд терминов и определений.

Разделяемым ресурсом называется ресурс, равно доступный всем взаимодействующим процессам. В качестве разделяемого ресурса могут выступать данные, организованные в виде файлов баз данных или разделяемой памяти в UNIX-системах, аппаратное обеспечение (принтеры, каналы связи, дисковые накопители), программные единицы (модули, библиотеки, задачи).

“Гонки” - это ситуация, при которой несколько процессов одновременно получают доступ к общему ресурсу. Результат в этом случае непредсказуем и определяется соотношением скоростей процессов. Особенно драматичны ситуации, когда асинхронные процессы могут изменять состояние общего ресурса.

Участок программы, в котором реализуется доступ к ресурсу, общему для нескольких процессов, называется критической секцией.

С понятием критической секции связано понятие взаимной блокировки. Различают два вида взаимных блокировок.

Первая разновидность, которую собственно и называют блокировкой (deadlock, clinch), соответствует ситуации, когда группа процессов не может продолжить выполнение, ожидая сообщения об освобождении критической секции. При этом такое сообщение никогда не поступит, поскольку критическая секция пуста. Классический пример возникновения блокировки приведен на рисунках 3 и 4. Другой разновидностью взаимных блокировок является отталкивание (livelock), при котором асинхронные процессы заняты только тем, что пересылают друг другу сообщение об освобождении критической секции.

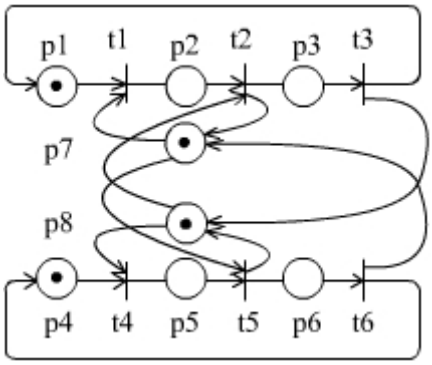

Рисунок 3 – Исходное состояние перед возникновением блокировки.

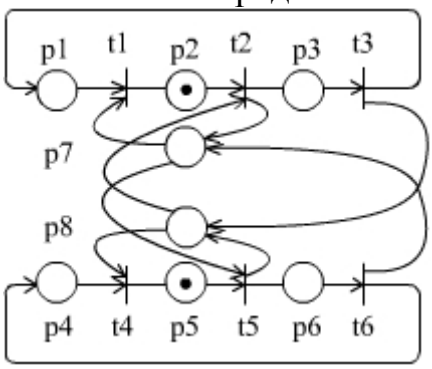

Рисунок 4 – Блокировка, возникшая после срабатывания переходов t_1 и t_4.

Как правило, при работе с разделяемым ресурсом устанавливается дисциплина доступа, при которой в каждый момент времени в критической секции может находиться не более одного процесса. Такая дисциплина носит название взаимного исключения (mutual exclusion, mutex). Она позволяет исключить ситуацию “гонок”. Надежная реализация дисциплины взаимного исключения обеспечивается в модели семафора Дейкстра (рисунок 1).

Барьером называется способ координации выполнения нескольких процессов, при котором все они должны достигнуть некоторой общей программной точки, прежде чем каждому из них будет позволено продолжить выполнение.

В данной лабораторной работе требуется вручную реализовать способ барьерной синхронизации параллельных процессов. В некоторых языках и средах программирования барьерная синхронизация поддерживается с помощью встроенных средств, как например в пакете MPI.

Также в данной работе потребуется реализовать одновременную передачу всем процессам общего и равнодоступного сигнала о наступлении события. При этом, хотя сигнал и представляет собой разделяемый ресурс, доступ к нему разрешен всем процессам одновременно.

Экспериментальная часть.

1. Разработать программу на языке C/C++ для ОС UNIX/Linux, имитирующую соревнования гоночных автомобилей.

Гоночный заезд включает в себя три этапа, очередной этап начинается после того, как все автомобили завершат предыдущий этап. Победитель определяется по итогам всех трех этапов. После того, как последний автомобиль достигнет финиша, производится подсчет баллов и распределение мест.

В гонках принимают участие пять автомобилей. Система начисления баллов определяется разработчиком. Необходимо также разработать интерфейс, позволяющий следить за ходом гонок в наглядной форме. В конце соревнования, а также в конце каждого этапа, вывести итоги, включающие в себя:

- расстановку машин по местам в итоге этапа или соревнования;
- набранные каждым участником баллы.

2. Гоночный автомобиль моделируется отдельным процессом. Процессы выполняются параллельно. Необходим также одни управляющий процесс, который запускает остальные процессы и отслеживает момент их завершения, а затем подсчитывает и выводит на экран итоги (арбитр).

3. Сигнал к началу каждого заезда подается арбитром однократно, в единственном экземпляре для всех участников. Способ реализации подачи сигнала к началу заезда – на усмотрение разработчика.

4. Для решения задачи необходимо использовать барьерную синхронизацию. Способ реализации барьерной синхронизации – на усмотрение разработчика, можно воспользоваться массивом, каждый элемент которого соответствует состоянию одного процессов, моделирующих гоночные автомобили.

5. Программа должна обладать интерфейсом, позволяющим в удобной форме наблюдать за ходом каждого заезда, то есть за движением машин к финишу.

5. Рекомендуемые функции: ftok, msgget, msgsnd, msgrcv msgctl. Описания функций приведены во встроенной справочной системе UNIX/Linux и в [1].