Проблемы

Две главные проблемы (задачи):

а) **Распределить эти глобальные общие ресурсы** между процессами – это довольно сложная задача. Нам нужно очень хорошо подумать, потому что тут уже не простая ситуация, когда есть процесс и ему нужно вот столько-то мегабайт и он закончится, есть другой процесс, ему нужно столько-то мегабайт и он закончится – в реальности процессов огромное количество, им нужна разная комбинация разных ресурсов, эти ресурсы также нужны другим программам и процессам. Это огромный, довольно хаотичный набор задач и ресурсов, которые нужны этим задачам. И чтобы всё это происходило оптимально нужно очень-очень хорошо подумать.

б) И проблема, возможно вторичная, но всё же довольно большая – когда мы пишем программу, которая имеет несколько процессов, которые одновременно делают что-то, которым нужна синхронизация – **очень сложно обнаружить ошибки в такой программе**, особенно если ошибки вызваны проблемами параллелизации. Такие ошибки сложно воспроизвести иногда потому, что те условия, в которых эти ошибки возникают – не детерминистические, они зависят от каких-то параметров, которые не всегда можно обнаружить, не всегда можно понять, что именно из-за этих параметров была проблема. И если один раз ваша параллельная программа выпала с ошибкой, то не всегда возможно воспроизвести эту ситуацию, возможно вообще она была уникальной.

Простой пример

Две программы выполняются на параллельных процессорах, но обе они выводят что-то на один общий экран. И если они одновременно завершились и одновременно пытаются вывести на экран (допустим экран это позволит), то получится какая-то «белиберда», получится текст на тексте. Если мы выводим на экран, то нам нужно, самое очевидное **решение – это разрешить доступ только одному процессу в один момент времени**. Так что если оба процесса одновременно завершились, то нужно будет выбрать: какой-то из них будет первый, какой-то второй. Если у нас есть такой экран, то к нему можно в один момент времени пускать только один процесс. Это очевидно, это правильно, но оказывается это не очень просто реализовать.

Заботы ОС

Естественно всё то, о чем мы говорим, всё то, о чем вспоминаем – всё это забота операционной системы (по большей части). Многие из этих проблем решаются на уровне архитектуры приложения, многие решаются на уровне железа, но очень много работы лежит на операционной системе. Это хорошо, потому что программистам не нужно очень много думать о том, как же им запускать их процессы. Процессам нужны ресурсы, и с этой точки зрения программисты такие же потребители, как и их программы: им нужны ресурсы, им нужно вот столько-то времени, им нужен доступ к файлам, и к памяти и они хотят решить свои задачи.

Задача параллелизации – это важная задача, но это последствия. Это не та задача, которую решают, когда пишут программу. Чаще всего программы пишут, чтобы решить какие-то проблемы и задачи из реальной жизни. При написании программ приходится работать с параллельными системами, параллельными процессами, и проблемы, которые возникают из этого – это уже проблемы инструментов. (Это также, как если бы мы строили дом и у нас появилась бы проблема тупых пил и нам пришлось бы еще тратить какое-то время на то, чтобы точить пилы, или менять их. Это тоже может быть неплохая задача, но это не та задача, ради которой мы здесь собрались. Мы строим дом.) Самое главное в таких углублениях, которые постоянно возникают в информатике и программировании – это не закопаться в обслуживании своих инструментов.

В заботы операционной системы, когда дело касается параллельных программ, входит: **слежение за процессами, освобождение и выдача ресурсов, защита ресурсов от других процессов,** то есть ничего нового, всё то же самое, что мы изучали раньше. Но тут дополнительно появляется то, что операционная система должна **сделать так, чтобы результат выполнения не зависел от скорости выполнения** и от последовательности выполнения. То есть если мы запустили

программу, то она должна дать один ответ независимости от того, будет ли она работать одна на всем компьютере пять минут или она будет полгода там работать с миллионами других процессов – результат должен быть всегда один и тот же.

Конкуренция за ресурсы

Когда процессы конкурируют за ресурсы, то есть они пытаются получить нужное количество разных ресурсов себе, то существует три главные проблемы (это с точки зрения операционной системы, задача которой сделать так, чтоб всем было хорошо):

- нам **необходимо взаимное исключение.** Те **критические секции**, о которых говорилось ранее, когда программы используют какие-то общие ресурсы – эти секции должны быть сделаны так, чтобы в один момент времени только одна программа могла быть в секции за какой-то ресурс. То есть если программа А начинает писать что-то в определенный файл, то в этот файл в этот момент времени никто не должен писать, это просто не должно быть возможно на уровне операционной системы;
- нам нужна **взаимная блокировка** (deadlock), точнее нам нужны инструменты, чтобы с этой взаимной блокировкой справляться. То есть нам нужно знание взаимной блокировки, нам нужна дедлок концепция, нам нужно понимать её и знать её свойства;
- нам нужно понимать и знать **свойства ресурсного голодания (starvation)**, чтобы как-то решить эту проблему.