

Терминология

Перед тем, как мы перейдем к основной части лекции необходимо познакомиться с терминологией, так как нам нужно знать все эти названия и концепции.

Атомарные операции (atomic operation) – это самая маленькая операция, которая возможна в данной системе (самое главное здесь не размер, а неделимость). Если операцию нельзя остановить на середине и потом продолжить, то это атомарная операция. Это важно для нас, потому что, когда мы имеем дело с несколькими процессами и, допустим, переключаемся между процессами в какой-то момент, то мы всегда это делаем между какими-то операциями. И в какой-то момент нам нужно определить, что эту операцию уже нельзя поделить на две части и остановить ее посередине и продолжить позже. Мы говорим, что это атомарная операция, она неделима (или если взять термин из физики, это будет квантовая операция, это мельчайший размер операции, меньше него быть не может).

Критическая секция (critical section) – это та часть программы в которой происходит работа с каким-то ресурсом. Допустим мы обращаемся в файл и что-то записываем в файл и этим же файлом пользуется какой-то другой процесс в другой момент времени, то есть этот ресурс общий – то вот эта секция кода, где мы идем в этот файл и что-то записываем или что-то считываем это критическая секция, потому что она работает с общим ресурсом для какого-то другого процесса.

Взаимная блокировка (deadlock) – это такая ситуация, когда один процесс ждет другой процесс (по какой-то причине, по какой-то механике – не важно, он ждет, что что-то случится с другим процессом), то другой процесс ждет его и в итоге получается, что они ждут друг друга бесконечно и ничего не происходит. Такая ситуация возможна на дорогах, допустим какая-то пробка на перекрестке и все пытаются проехать, все въехали в кучу в центре и теперь поток ни слева-направо, ни снизу-вверх не может ехать, и вроде как все готовы ехать, но всем все мешают. Это такой классический дедлок.

Зацикливание системы (livelock) – это похожая концепция, но если в дедлоке все стоят, ждут и ничего не происходит, то в ливлоке все двигаются, все что-то делают, но всё равно ничего не происходит, то есть движение есть, но оно ни к чему не приводит. Самые очевидный пример, это когда процесс «жалуется», что ему не хватает памяти, он запускает новый подпроцесс, чтобы получить больше памяти, а памяти реальной в системе уже нет, и этот новый подпроцесс, естественно, новой памяти не получает, он «жалуется», что памяти нет и запускает новый подпроцесс... Можно подумать, что это проблема плохого кода, но это отчасти проблема операционной системы. В общем, если в том же примере с перекрестка все начинают двигаться, все начинают крутиться и как-то ездить, но при этом это движение ещё больше всем мешает и всё равно никто никуда не уезжает с этого перекрестка – то перед нами ливлок. И сложно еще подумать, какая из этих проблем будет лучше, какую будет легче решить.

Взаимное исключение (mutual exclusion) – это важная концепция, которую мы будем встречать еще много раз вообще, когда будем думать об операционной системе и процессах – это свойство, которое нам в некоторых случаях нужно удовлетворить. Тут опять же просится аналогия с туалетом: то, что мы сказали, что в туалете есть такое правило, что если кто-то зашел, то больше никто не может зайти, то это взаимно исключение. Это означает, что в один момент времени туда не может зайти второй человек.

Состояние гонки (race condition) – это состояние системы, когда результат зависит от порядка исполнения. А порядок исполнения не всегда одинаковый: это зависит от других условий, от каких-то процессов, которые запущены, от внешних условий – и это плохо, потому что нам нужно, чтобы результат любой операции был детерминистический, то есть если мы задали какие-то входные данные, то мы должны получить какой-то результат, и этот результат не должен зависеть от того, как там внутри процессы решили между собой соревноваться и с какой скоростью они работали, и какой процесс завершился первым. Но важно, чтобы в результате был один и тот же итог. И так же, как в

гонках что-то зависит от разных условий, и разный финиш зависит от того, с какой скоростью машины едут и как себя ведут. В программировании нам этого не хочется видеть. Нам хочется точности, поэтому если программа, которая работает, допустим, с несколькими параллельными процессами считает что-то, и она запрограммирована так, что в одних условиях процессы идут в определенном порядке и результат один, а в других условиях порядок изменяется и результат другой, то это плохо. Это называется состоянием гонки, и мы пытаемся избежать этого, когда программируем софт, и когда операционная система распределяет эти задачи и пытается раздать ресурсы этим задачам.

Ресурсное голодание (resource starvation) – это именно то, что можно придумать из названия, это когда процессу не хватает ресурсов, он голодает и не может ничего сделать, потому что система по какой-то причине никак не выдает ему ресурса. Может быть она так устроена, что там есть приоритеты и какой-то процесс в самом конце списка и постоянно до него не доходят свободные ресурсы. Такая несправедливость возможна, это проблема во многом операционной системы и это тоже естественно негативная черта, которую нужно избегать.