Лекция 5. Взаимодействие процессов и нити исполнения. Алгоритмы синхронизации.

Потоки

Потоки -- это сущности внутри процессов, которые используют общую память и требуют меньше времени для переключения контекста по сравнению с процессами.

Идея потоков возникла, когда придумали использовать одни и те же данные: одна лента, и несколько головок.

Компоненты контекста:

- Системный
- Регистровый
- Пользовательский
    - Код
    - Данные
    - Стек

Процесс исполнения имеет свой регистровый контекст и стек, а также часть системного контекста, которая обслуживает данные, относящиеся именно к нему. Все остальные ресурсы (в т.ч адресное пространство и устройства ввода-вывода) являются общими для всех потоков процесса. Изначально создается только одна нить исполнения. Она может порождать новые нити.

Состояния процессов, если потоков много:

- Рождение
- Готовность -- если ни одна нить не находится в состоянии “Исполнение” и хотя бы одна нить находится в состоянии “Готовность”.
- Исполнение -- если хотя бы одна нить находится в состоянии исполнения.
- Ожидание -- если хотя бы одна нить находится в состоянии ожидания и ни одна нить не находится в состоянии исполнения или готовности
- Завершил исполнение -- если хотя бы одна нить находится в состоянии ожидания и ни одна нить не находится в состоянии исполнения или готовности

Реализация нитей:

1. Все планирование осуществляется в терминах нитей исполнения. Процессор предоставляется нитям исполнения, а ресурсы выделяются процессу.
2. Нити исполнения реализуются на уровне пользовательских библиотек. Ядро не будет знать, что процесс состоит из нескольких нитей. Время для занятия процессора выделяется процессу целиком, а библиотека уже делит его на нити.

Активности и атомарные операции

Активность -- это последовательность некоторых действий, направленных на достижение какой-то цели. Активности состоят из неделимых, атомарных действий.

Если выполняется несколько активностей параллельно, то атомарные операции разных активностей могут перемешиваться, но сохраняется при этом порядок внутри одной активности.

Interleaving и детерминированные и недетерминированные наборы активностей

Interleaving -- процедура чередования операций разных активностей.

Детерминированный набор активностей -- наборы активностей, которые вне зависимости от интерливинга, дают один и тот же результат.

Условие Бернстайна

Условие Бернстайна: набор активностей является детерминированным, если соблюдаются 3 условия:

- Входные переменные одной активности и выходные переменные другой активности не перекрываются
- Выходные переменные второй активности и входные переменные первой активности не перекрываются
- Множество входных переменных обеих активностей не перекрывается

Для того, чтобы набор активностей был детерминированным, интерливинг нужно ограничить: запретить взаимное использование одновременно разделяемых переменных, так чтобы доступ к разделяемым ресурсам осуществляется строго в порядке очереди -- взаимоисключение.

Если взаимоисключения нет, то доступ к переменным получает тот, кто первым ими завладел -- race condition.

Критическая секция

Критическая секция -- часть активности, выполнение которой в наборе с другими активностями, может привести к возникновению race condition.

Для крит. секции применяются след. требования:

- Программный.
- Не имеет представления о времени работы процессов и о времени прохождения ими критической секции.
- Должен обеспечивать взаимоисключение между процессами при прохождении ими критических участков.
- Условие прогресса — не может быть ситуации, когда
    - на решение о том, какой из процессов войдет в критический участок, оказывает влияние процесс не находящийся либо в ожидании входа в критической участок, либо внутри него.
    - процессы, которые принимают решение о том, кто войдет в критический участок, не должны делать это бесконечно долго.
- Условие ограниченного времени ожидания — алгоритм должен быть справедлив по отношению ко всем процессам.

Алгоритмы взаимодействия

Один из алгоритмов реализации критической секции включает в себя посылание процессору команды для запрещения прерываний. Т.е. никакой другой процесс не сможет вытеснить его и начать исполняться сам. Но такой способ недоступен пользователям и используется только для написания ядра ОС.

Можно пользоваться флагом при котором 0 -- вход в крит. секцию открыт, а 1 -- нет. Во входной секции процесс, вошедший в критическую секцию, ставит флаг

= 1, а в выходной секции -- 0. Но операции проверки значения переменной на равенство нулю и установления ее в единицу -- две отдельные операции. Бывают ситуации, когда два процесса успели увидеть, что переменная равна нулю и войти в критическую секцию до установления переменной в единицу.

Можно завести переменную turn, которая хранит в себе номер процесса, который может войти в критическую секцию. Однако этот алгоритм также ошибочен, поскольку ход может быть передан процессу, который не ожидает входа в критическую секцию.

Для улучшения этого алгоритма нужно ввести флаги готовности ко входу в критическую секцию.В прологе каждый процесс объявляет о своей готовности ко входу в критическую секцию и передает ход процессу, который объявил о своей готовности раньше. В эпилоге свой флаг готовности сбрасывается в ноль. Однако этот алгоритм также ошибочен, поскольку здесь может бесконечно приниматься решение о том, кто первый войдет в критическую секцию.