# Операционные системы Лекция 4

…Для этой очереди выбирается квант, таким образом, чтобы … или успело завершиться. Если процесс за выделенный ему квант времени не сделал ни то, ни другое, он переходит в более низкоприоритетную очередь. Переходят так, пока не попадут в самую низкоприоритетную очередь, которая выполняется по алгоритму RR. В итоге, в этой очереди оказываются процессы, требующие много проц. Времени (вычислительные процессы). Также этой очереди, крутится холостой процесс, т.к. система не может ничего не делать.

Вернёмся к диаграмме состояний процесса:

Рисунок

Из состояния выполнения в состояние готовности процесс может перейти из-за вытеснения или истёк квант. Из выполнения в блокировку при запросе ресурсов.

Системный вызов выполняет процесс, когда ему нужно обслуживание. (см. Лекция 3). Аппаратные прерывания – тоже переключает в режим ядра. Или исключения.

Когда процесс переходит в режим ядра, происходит переключение аппаратного контекста. (аппаратный контекст – это содержимое регистров. Их нужно сохранить, чтобы можно было продолжить процесс)

Полный контекст состоит из аппаратного контекста и из информации о выделенных процессу ресурсов.

Поддержание аппаратного контекста поддерживается аппаратно (pusha, …)

Полный контекст — это также указатели на структуры ядра, которые описывают выделенные процессу ресурсы. То-есть переключение полного контекста – затратное действие. Для сокращения затратности этого действия предложили использовать потоки.

Поток – это часть кода программы, которая может выполнятся параллельно с другими частями кода программы. (для «одарённых»: непрерывная часть кода)

У потока имеется ряд характерных особенностей. Но сначала нам надо классифицировать потоки. По большому счёту существует два типа потоков, в современном мире.

* Потоки уровня пользователя (буквально: потоки пользователя)
* Потоки уровня ядра (буквально: потоки ядра)

Потоки уровня пользователя, поддерживаются с помощью специальных библиотек, уровня пользователя. Никакой поддержки ядра у таких потоков нет. Ядро ничего о них не знает.

(Пара пропущенных предложений…) Операционная система управляет потоками.

### Потоки уровня пользователя

Рисунок (см. тетрадь)

Библиотека должна предоставлять функции для работы с потоками (Создание… переключение…)

Если поток запросил, например, ввод/вывод, блокироваться будет вся программа. (система ничего не знает о потоках – они не могут выполняться)

### Потоки уровня ядра

Рисунок (см. тетрадь)

То-есть программа делится на поток. Потоки получают процессорное время. В очередь к процессору стоят потоки. (поток владеет счётчик команд - поток владеет аппаратным контекстом) Но поток не имеет своего адресного пространства и выполняется в адресном пространстве процесса. В результате выполняет - поток, запрашивает дополнительную память - поток, но этот ресурс принадлежит процессу. И так во всём. Процесс владеет ресурсами, всеми. Запрашивают – потоки, выполняют – потоки, а ресурсами владеет процесс. (поток владеет только аппаратным контекстом)

Система не может выбирать, что сначала будут стоять потоки одного процесса, а потом потоки другого процесса. То-есть в очередь выстраиваются потоки, но никак не учитывается то, к какому процессу принадлежит поток. (что было бы очень выгодно – переключается только часть кода, аппаратный контекст) Средне статистически всё равно существует выигрыш.

Для поддержки потоков уровня пользователя, существуют специальные библиотеки. ?Аналогично, потоки уровня пользователя владеют…?

## Однопоточная и многопоточная модели процесса

Однопоточная модель, имеет право на существования, потому что при появлении потоков, для единообразия управления, даже если в программе не создаётся никакой поток, считается что в этой программе выполняется один главный поток. Это сделано для единообразного управления выполнением работы в системе.

Рис

В системе существует структуры описывающая процесс (блок управления процессом – на самом деле это дескриптор управления процессом). И также для управления потоками, существует структура, описывающая поток. Описывает ту часть, которую не описывает структуру процесса. (Содержит инф. для переключения и приоритеты потоков) Еще раз повторим: владельцем ресурсов является процесс, поэтому процесс называется единицей декомпозиции ?…?

Потоки могут разделять общие данные. Файл – это средство долговременного хранения файлов. И владельцем открытых файлов, тоже является процесс (файлы открывают потоки, но они принадлежат процессу)

Потоки одной и той же программы могут выполняться параллельно разными ядрами.

Причина:

1. Потоки полезны если процесс имеет несколько задач, которые можно выполнять, независимо друг от друга.
2. Это особенно полезно, когда одна задача может блокироваться, в то время как другие задачи могут продолжать выполняться.

Частым примером является текстовый редактор. В нём, некоторый низкоприоритетный поток, может, например, проверять грамматику. В это время другой поток, более высокого приоритета, может обрабатывать ввод (нажатие клавиш). В это же время, третий поток, может загружать изображение, жёсткого диска, а 4ый поток, может периодически выполнять резервное копирование, введённого текста в файл. (Например Word) – это пример правильного распараллеливания

Достоинства многопоточности:

1. **Отзывчивость** (один из потоков, может быстро обрабатывать действие пользователя, в то время как другой заблокирован или выполняет большой объём вычислений)
2. **Разделение ресурсов** (по умолчанию … процесса, что позволяет потокам, выполнятся параллельно, в одном адресном пространстве)
3. **Экономичность** (при использовании потоков, переключение контекста, при управлении потоками происходит значительно быстрее, чем переключение контекста при использовании процессов)
4. **Масштабируемость** (однопоточные процессы могут выполняться только на одном процессоре, в то время как многопоточные приложения, в много процессорных системах, могут выполнятся на разных процессах – соответственно время выполнения многопоточных процессов будут меньше однопоточных в многопроцессорных системах)

Проблемы при многопоточности. Для разработчиков приложений существует 5 областей в которых многопроцессорность требует решения определённых проблем:

1. **Правильное распараллеливание** (Разработчик должен справится с задачей разделения процесса на потоки)
2. **Сбалансированность** (имеется ввиду поиск задач, которые имеют близкую по объёму функциональность – не надо тратить потоки на тривиальные задачи)
3. **Разделение данных** (Неправильное разделение данных может приводить к ошибкам или потере данных при взаимодействии параллельных потоков)
4. **Зависимость данных** (одна задача может зависеть от результата другой задачи, тогда выполнение этих задач должно быть синхронизировано, для того чтобы доступ к данным выполнялся в правильном порядке)
5. **Тестирование и отладка** (очевидно более сложные, возможны гонки (race condition?). В многопоточных приложениях, сложнее найти ошибку)

Типы параллелизма.

Теоретически имеется два различных пути, для распараллеливания нагрузка. Так называем параллелизм данных и параллелизм задач.

Распараллеливание данных – это распараллеливание данных между несколькими ядрами и выполнение одной и той же последовательности обработки этих данных для каждого подмножества этих данных. Например, разбиение большого изображения на фрагменты, которые подвергаются одинаковой обработке.

Распараллеливание задач – это выделение задач, которые могут выполнятся параллельно (пример: текстовый редактор)

Рассматривается 4 модели:

Первая модель, называется **многие к одному** (см. тетрадь) (много пользовательских потоков к одному потоку ядра – блокировка одного приводит к блокировке всех)

Вторая модель **один к одному** (см. тетрадь) – создаются отдельные потоки ядра, для каждого потока пользователя (здесь нет проблемы с блокировкой, но накладные затраты будут более значительными). Большинство реализаций этой моделей, ограничивают кол-во потоков, которые могут быть созданы.

Следующая (третья) модель **многие ко многим** (см. тетрадь) – э та модель мультиплексирует любое кол-во потоков пользователей на равное или меньшее кол-во потоков ядра. При этом пользователь не имеет ни каких ограничений на кол-во потоков. Блокировка одного потока не приводит к блокировки всего процесса. Процессы могут разделятся, между множеством процессоров. При этом процесс может учитывать, кол-во потоков ядра и кол-во ядер. Популярной вариацией этой модели является так называемая **двухуровневая модель** (см. тетрадь) Некоторые системы поддерживают именно такую модель (Например HP-UX и Solaris до Solaris-9)