Планирование процессов

Переключения контекста это не есть операция планирования, это техническая операция

- Происходит прерывание
- Поток вызывает исключение или ловушку (trap)
- После этого выбирается другой поток

Т.е. в процессе переключения контекста нужно четко выбрать, кому передать управление.

Уже был затронут вопрос об очереди готовых процессов. **Решение о том, кому дать следующий квант времени процессора определяет планирование**

Планирование

- Процесс выбора – кто будет исполняться следующим и как долго это будет исполняться называется **планированием процессов** в ОС.
- Не путать с переключением контекста, что является просто механизмом передачи управления.

Классы планировщиков
Пакетный

1) **Пакетный** – ориентирован на длительные задачи, которые требуют больших вычислительных ресурсов, где не требуется частое прерывание. Т.е. подразумевают обработку больших задач большими пакетами, не ограничения на время выполнения.

Классы планировщиков Интерактивный

2) **Интерактивный** – ориентирован на снижение времени отклика, т.е. чтобы система казалась”отзывчивой”. Обычные абонентские системы на ПК – это интерактивные системы, когда в ответ на действие пользователя (например перемещение мыши) ОС что-то делает. И всегда пользователю хочется, чтобы этот ответ происходил как можно быстрее.

Главное чтобы на поступающий в систему запрос был получен максимально быстро ответ. Запрос – это любое взаимодействие с компьютером.

Классы планировщиков реального времени

3) **Реального времени** – специализированные класс, ориентированный на дедлайн – предельный срок завершения какой-либо работы.

Главное, чтобы определенное действие завершалось к определенному сроку, это понятие называется **дедлайн**.

Поступающий запрос должен быть обработан не более, чем в определенный промежуток времени.

Классический пример СРВ – управление ядерным реактором, в котором превышение времени отклика приведет к аварийной ситуации.

Уровни планирования

Долговременное(догосрочное) – решает какие новые задачи будут добавлены (концептуальные вопросы).

Среднесрочное – решает нужно ли временно выгружать программу во вторичную память (какую и вообще нужно ли это).

Краткосрочный – решает, какому потоку дать следующий квант процессорного времени и какой длины. Координирует выполняющиеся потоки на разных ЦП.

Уровни планирования

Основной задачей планирования процессов в ОС является обеспечение высокой производительности ОС.

Существуют разные метрики, которыми оценивается эта производительность.

Эти метрики зачастую противоречивы.

Что концептуально требуется при проектировании планировщика ОС?

-Максимизировать использование ЦП (чтобы он максимально работал на задачах)

-Максимизировать пропускную способность(число выполненных запросов в единицу времени)

-Минимизировать среднее время отклика (среднее время от подачи запроса до завершения обработки ответа)

-Минимизировать среднее время ожидания (среднее время от подачи запроса до начала его выполнения)

-Минимизировать энергии (джоулей на инструкцию)

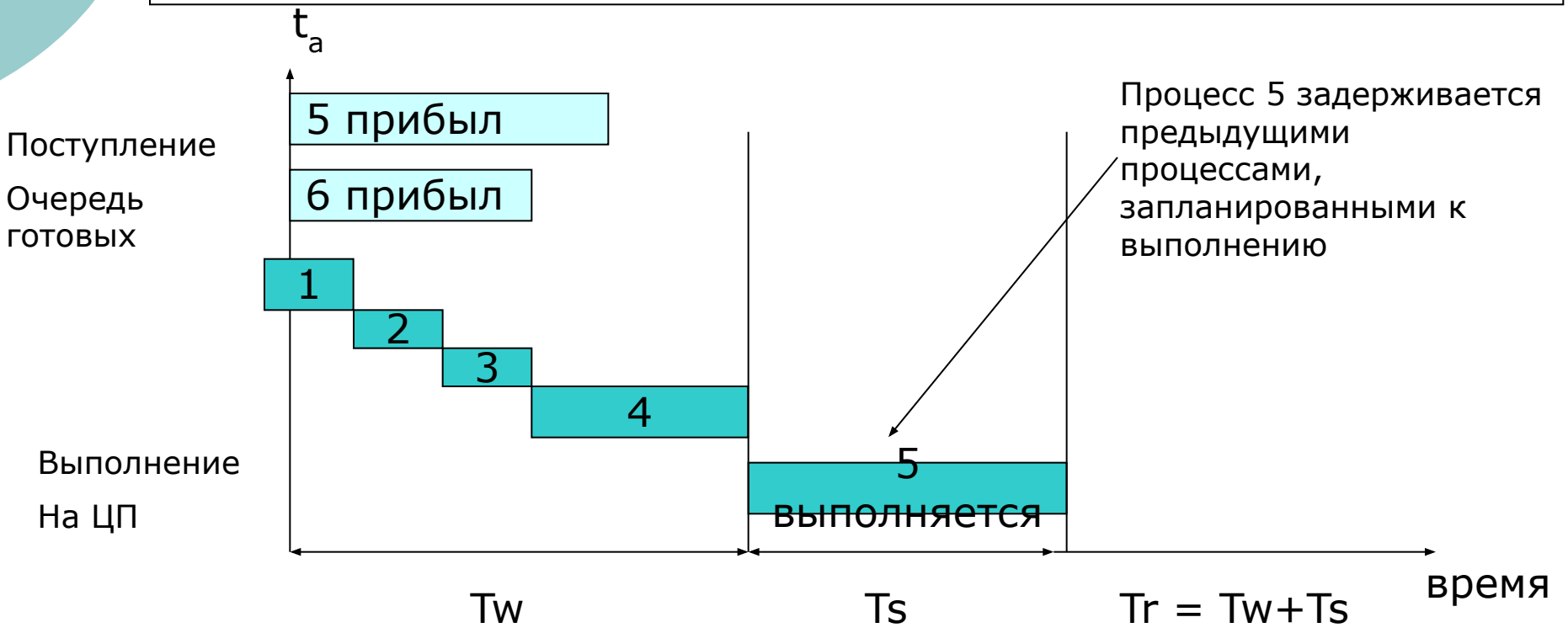
Метрики планирования. Пример

t_a - время поступления процесса (когда процесс становится готовым к выполнению)

Tw – время ожидания (которое тратит процесс в очереди на выполнение)

Ts – время выполнения ЦП

Tr – время оборота (общее время на ожидание ивыполнение)



Метрики планирования. Пример

- На схеме 5 и 6 процессы поступили в очередь готовых процессов.
- 5 задержался из-за 1-4 процесов. Для пятого процесса Tw – время ожидания Ts – время выполнения ЦП. Время оборота это время от момента его поступления до момента, когда завершилось его выполнение Tr=Tw+Ts

Метрики планирования
Как выбрать какой процесс будет работать дольше?

- **FIFO**- классика – первым пришел, первым ушел
- **Кратчайшая работа следующей**, т.е. следующей выбирается работа, которая требует наименьшего времени завершения
- **Round-robin**
- **Многоуровневая очередь**
- **Многоуровневая очередь с обратной связью**

FIFO

В данном случае мы будем его понимать как невытесняющую многозадачность

- Процессы планируются по мере их поступления
- Время выполнения не учитывается (никак совсем)
- Другие процесс с меньшим временем Tr вынуждены ждать (снижается отзывчивость системы)
- Когда процесс переходит в состояние готовности он перемещается в конец очереди

Пример FIFO

Допустим есть 3 процесса, которые пребывают в одно и тоже время t=0 в порядке P1,P2,P3.

У каждого из процессов существует время, которое ему нужно для выполнения части задачи. Эту часть задачи, которую ему необходимо выполнить назовем английским словом Burst. У трех процессов она разная.

Burst(P1)=24 (усл.ед.времени)

Burst(P2)=3

Burst(P3)=3

Тогда Время ожидания

Wait(P1)=0

Wait(P2)=24

Wait(P3)=27

Среднее время ожидания = **17**

Пример FIFO

Если эти 3 поступившие процесса запланировать по другому, можно сильно снизить время отклика системы.

Допустим процессы поступают в порядке P2,P3,P1

Тогда время ожидания

P2=0,

P3=3,

P1=6

Среднее время ожидания =3 - Оно резко снизилось за счет того, что мы изменили порядок работы процессов, поступивших в одно и тоже время.

За данным простым примером скрыта вся мощь и важность алгоритмов планирования в ОС.

Обобщения по FIFO

Он больше других подходит для длительных, требовательных к времени ЦП процессов;

Плохое использование ЦП и устройств вв/выв

Среднее время ожидание сильно варьируется

Кратчайшая работа следующей

Условимся, имеется в виду невытесняющая политика планирования – сколько квант времени запрашивает процесс, столько ему и выделяется.

Суть процесса - следующим запланировать тот процесс, который требует **наименьшего времени для** своего **выполнения,** т.е. процесс имеющий самое короткое время обработки

Кратчайшая работа следующей Сложности

Сложность – нужно оценивать требуемое время обработки для каждого процесса.

- Для пакетных заданий на основе предыдущего опыта или вручную (нет гарантии, что повторится)
- Для интерактивных заданий на основе затраченного времени

Как только мы получаем метрику процессов, то короткий процесс помещается в начало очереди.

Кратчайшая работа следующей вытесняющий вариант

Существует вытесняющий вариант метода кратчайшей работы следующего.

Сортировка осуществляется по времени, которое нужно процессу для завершения своей части задачи. Если он вытесняется в процессе выполнения, то время, которое у него осталось называется **остаточным**.

По остаточному времени осуществляется сортировка и принимается решение, какой процесс будет выполняться следующим.

Соответственно те процессы, которые выполняются на ЦП вытесняются тем процессом, который близкий к завершению, чтобы отработать его и распрощаться с ним.

А те процессы, которым еще долго работать, оставить на потом и заняться ими “в плотную”. Логика в этом есть

Кратчайшая работа следующей обощение

- Процессы, уже выполняющиеся на ЦП вытесняются самым близким к завершению заданием
- Меньше общее время оборота процесса
- Меньше время ожидания ЦП

Планирование с приоритетами

- Тот же алгоритм кратчайшей работы следующей можно представить, как планирование с приоритетом, где приоритет – наименьшее время работы.
- Суть – каждому процессу сопоставляется некоторое число, которое характеризует, определяет приоритет этого процесса. Чем меньше это число, тем выше приоритет.
- Проблема старвации – это проблема “зависания”, “голодания” – если процессу с высоким приоритетом приспичит выполнить очень длительную работу, то все остальные процессы будут “висеть” и ждать

Планирование с приоритетами

- Время ЦП выделяется процессу с наивысшим приоритетом (вытесняющим или невытесняющим) Процесс с низким приоритетом может вообще никогда не выполниться, до него не дойдет очередь.

Старвация – ее можно представить как всех стоящих в очереди и кто привилегированный лезет вне очереди.

- Проявляется в алгоритме КРС
- Низкоприоритетные запросы могут никогда не выполниться.

Планирование с приоритетами

Решение проблемы

- Ввести понятие «старения»: по мере течения времени увеличивать приоритет процесса
- Приоритет = Оценочное время выполнение на ЦП – время ожидания
- **Приоритеты** – это инструмент с помощью которого необходимо сделать общий процесс планирования эффективным

Round-robin

- Данный алгоритм планирования обозначает циклический алгоритм с вытесняющим планированием.
- Каждый процесс получает фиксированный квант процессорного времени (фиксированную единицу процессорного времени)
- После истечения кванта времени процесс принудительно вытесняется и помещается в очередь готовых к выполнению.
- Процесс всегда планируются в том же порядке и каждый процесс получает одинаковый квант времени
- Не сложно подсчитать: если квант времени равен **q** и **n**-процессов в очереди, то каждый получит **1/n** времени ЦП, кусками максимум по **q**
- Никакой из процессов не ожидает больше, чем

(n-1)*q единиц времени

Производительность Round-robin

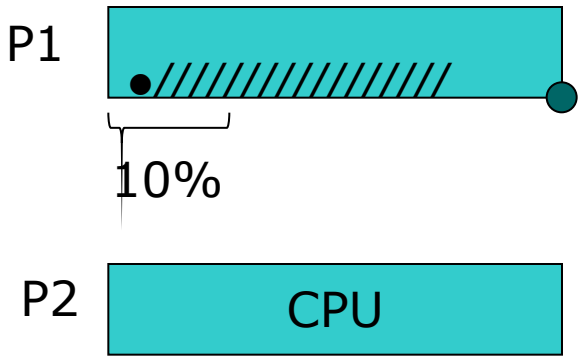
Если q большое(стремиться к ∞), то RR перерождается в алгоритм FIFO

Если q малое (но не стремится к 0, иначе ПК будет только переключать процессы и больше не выполнять вообще ничего), то все хорошо

- Нет старвации
- Появляется высокая отзывчивость системы
- Равное распределение времени
- Если q меньше, чем время, затрачиваемое на переключение контекста, тогда диспетчер будет неэффективным.

Недостаток Round-robin

Процессы с интенсивным вв/выв(т.е. заблокированные в ожидании вв/выв) полностью не используют свой квант времени, поэтому процессы с интенсивным использованием ЦП получают преимущество.



Есть 2 процесса P1 и P2

Процесс P1 ожидает вв/вывод в точке (●), пока этот вв/выв не завершится часть отмеченного штриховкой времени процесс P1 потратит «впустую», он вытиснится только в точке ● по истечении кванта времени. P2 в это время активно использует ЦП, например, считает.

Проблема RR в том, что не учитываются задержки, и полезное время работы P1 составляет только 10%.

Многоуровневые очереди

- Выделяется несколько разных очередей, например
  - Очередь интерактивных процессов, т.е. тех, которые требуют малого времени отклика
  - Очередь фоновых процессов, требующих много выч.ресурсов, но для которых не важно быстрое время отклика(пакетная обработка), нужно много повычислять.
- У каждой очереди сопоставляется свой алгоритм планирования, т.о. имеем некий «баланс сил»
  - у интерактивных процессов RR
  - у фоновых - FIFO

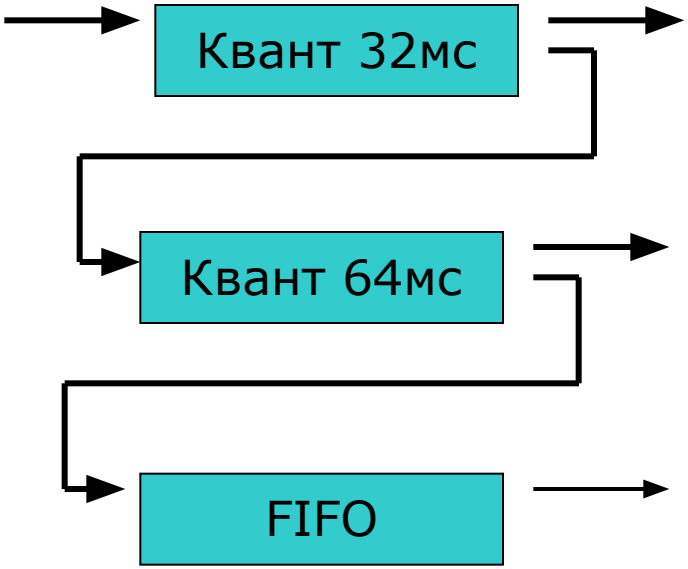
Многоуровневые очереди

Но в случае многоуровневых очередей нужно планирование не просто внутри каждой очереди, но и планирование между очередями. Получается «накрученный» планировщик, можно предложить много вариантов:

- **Планирование с фиксированным приоритетом**
  - Вначале обслужить все интерактивные процессы, потом все фоновые
  - Возможна старвация
- **Разделение времени**
  - Каждой очереди выделяется часть времени ЦП, которую она может распланировать между своими процессами
  - Например 80% времени ЦП на интерактивные процессы через RR, 20% на фоновые через FIFO.
- **Многоуровневая очередь с обратной связью**

Многоуровневая очередь с обратной связью

Планирование на основе затраченного времени, если процесс затратил определенный квант времени, то он помещается в определенную очередь – динамически перепланируются очереди



- Если он выполнился достаточно быстро, то он попадает в первую очередь «быстрых» процессов.
- Если он средний по времени выполнения, то в среднюю.
- Если он требует много времени выч. ресурсов, то он помещается в последнюю очередь FIFO.

За счет этого процессы постоянно перемещаются между очередями, т.о. заранее не нужно смотреть, куда помещать процесс и сопоставлять ему какое-то свойство

Многоуровневая очередь с обратной связью

Планировщик определяется с многими параметрами:

- Числом очередей
- Алгоритмами планирования в каждой очереди
- Методом, используемым для определения принадлежности процесса к той или иной очереди

Многоуровневая очередь с обратной связью. Пример

Есть 3 очереди:

- Q_0 –RR с квантом времени t=16мс
- Q_1 –RR с квантом времени t=32мс
- Q_2 –FIFO

<u>Планирование:</u>

Новый процесс помещается в конец первой очереди Q_0

- Когда процесс из этой очереди получает ЦП, то выделяется квант времени t=16мс
- Если процесс выполняется дольше, не вернул управление ОС, то он принудительно вытесняется и помещается в конец очереди Q1

В очереди Q1

- Когда процесс из этой очереди получает ЦП, то выделяется квант времени t=32мс
- Если процесс выполняется дольше, то он принудительно вытесняется и помещается в конец очереди Q2 и выполняется по FIFO пока не закончится