ЛЕКЦИЯ №8

ТЕМА: Алгоритмы планирования в системах пакетной обработки. Планирование в интерактивных системах. Общее планирование реального времени.

ЦЕЛЬ: изучить основные виды планирования, понять принцип их работы в реальных проектах.

Существуют следующие алгоритмы планирования процессов:

1. **FIFO**— классика – первым пришел, первым ушел
2. **Кратчайшая работа первой (SJF)**, т.е. следующей выбирается работа, которая требует наименьшего времени завершения
3. **Циклическое планирование**
4. **Планирование с приоритетом**
5. **Round-robin**
6. **Многоуровневая очередь**
7. **Многоуровневая очередь с обратной связью**

Рассмотрим названные алгоритмы планирования процессов.

Планирование в системах пакетной обработки

1) FIFO (First In, First Out (первым вошел, первым вышел)

Процессы ставятся в очередь по мере поступления.

Преимущества:

- Простота
- Справедливость (как в очереди покупателей, кто последний пришел, тот оказался в конце очереди)

Недостатки:

- Процесс, ограниченный возможностями процессора может затормозить более быстрые процессы, ограниченные устройствами ввода/вывода.

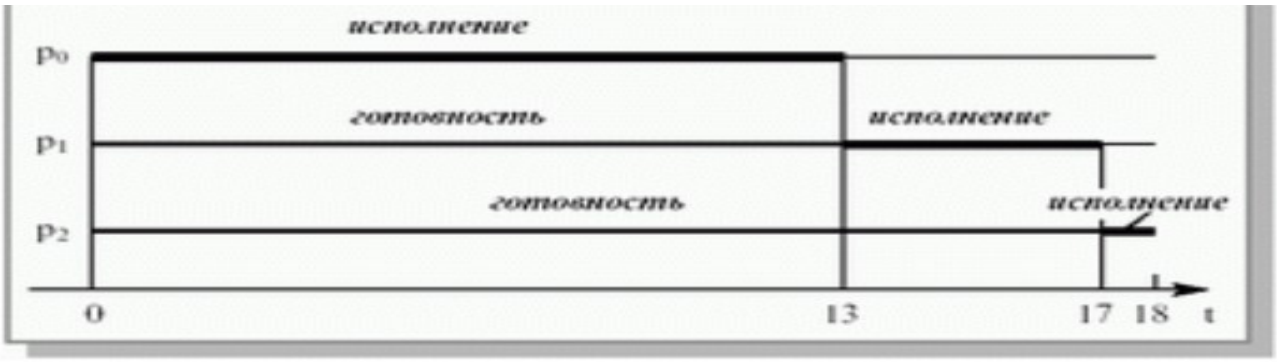
В данном случае мы будем его понимать как невытесняющую многозадачность:

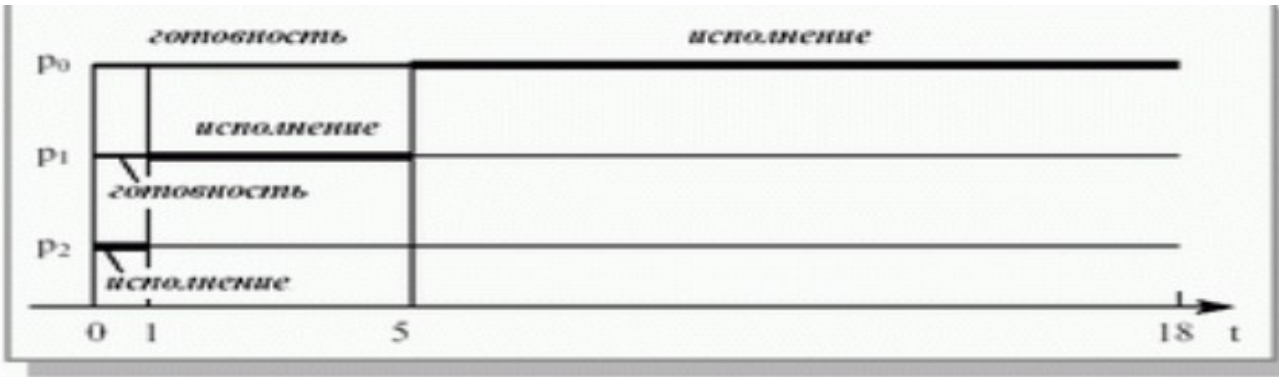
1. Процессы планируются по мере их поступления;
2. Время выполнения не учитывается (никак совсем);

3. Другие процесс с меньшим временем Tr(временем обработки) вынуждены ждать (снижается отзывчивость системы);

4. Когда процесс переходит в состояние готовности он перемещается в конец очереди.

Пример FIFO





Обобщения по FIFO:

- Он больше других подходит для длительных, требовательных к времени ЦП процессов;
- Плохое использование ЦП и устройств ввода/вывода;
- Среднее время ожидание сильно варьируется.

Среднее время ожидания и среднее полное время выполнения для этого алгоритма существенно зависят от порядка расположения процессов в очереди. Если у нас есть процесс с длительным временем , то короткие процессы, перешедшие в состояние готовность после длительного процесса, будут очень долго ждать начала выполнения.

Поэтому алгоритм FCFS практически неприменим для систем разделения времени – слишком большим получается среднее время отклика в интерактивных процессах.

2) Shortest-Job-First Кратчайшая работа первой (SJF)

Суть процесса — первым запланируется тот процесс, который требует **наименьшего времени для** своего **выполнения,** т.е. процесс имеющий **самое короткое время обработки.**

Сложности:

Нужно оценивать требуемое время обработки для каждого процесса.

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

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

4 мин.	6 мин.	2	4 мин.	2	2

2	2	2	4 мин.	4 мин.	6 мин.

Нижняя очередь выстроена с учетом этого алгоритма

***Преимущества*:**

- Уменьшение оборотного времени
- Справедливость (как в очереди покупателей, кто без сдачи проходит в перед)
- Квантование времени не применяется.

Нижняя очередь выстроена с учетом этого алгоритма

Недостатки:

- Длинный процесс занявший процессор, не пустит более новые краткие процессы, которые пришли позже.

SJF алгоритм краткосрочного планирования может быть как вытесняющим, так и невытесняющим.

При невытесняющем SJF планировании процессор предоставляется избранному процессу на все требующееся ему время, независимо от событий происходящих в вычислительной системе.

При вытесняющем SJF планировании учитывается появление новых процессов в очереди готовых к исполнению (из числа вновь родившихся или разблокированных) во время работы выбранного процесса. Если CPU burst нового процесса меньше, чем остаток CPU burst у исполняющегося, то исполняющийся процесс вытесняется новым.

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

Обобщение по «Кратчайшая работа первой»:

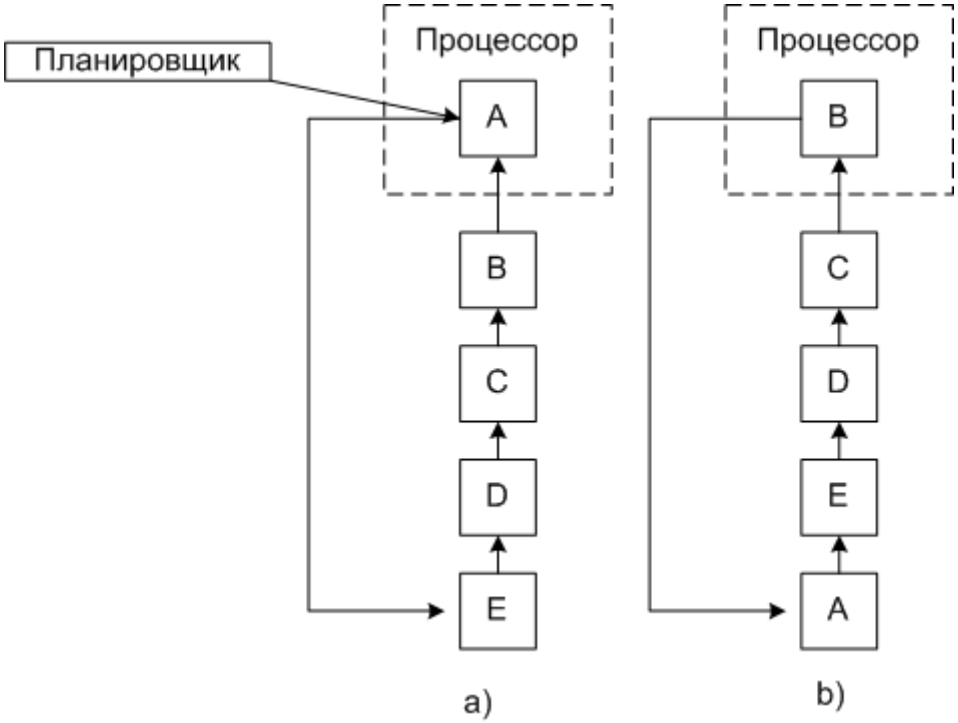
1. Процессы, уже выполняющиеся на ЦП вытесняются самым близким к завершению заданием;
2. Меньше общее время оборота процесса;
3. Меньше время ожидания ЦП.

5.3 Планирование в интерактивных системах

3) Циклическое планирование

Самый простой алгоритм планирования и часто используемый.

Каждому процессу предоставляется квант времени процессора. Когда квант заканчивается процесс переводится планировщиком в конец очереди. При блокировке процессор выпадает из очереди.



Пример циклического планирования

Преимущества:

1) Простота

2) Справедливость (как в очереди покупателей, каждому только по килограмму)

Недостатки:

1) Если частые переключения (квант - 4мс, а время переключения равно 1мс), то происходит уменьшение производительности.

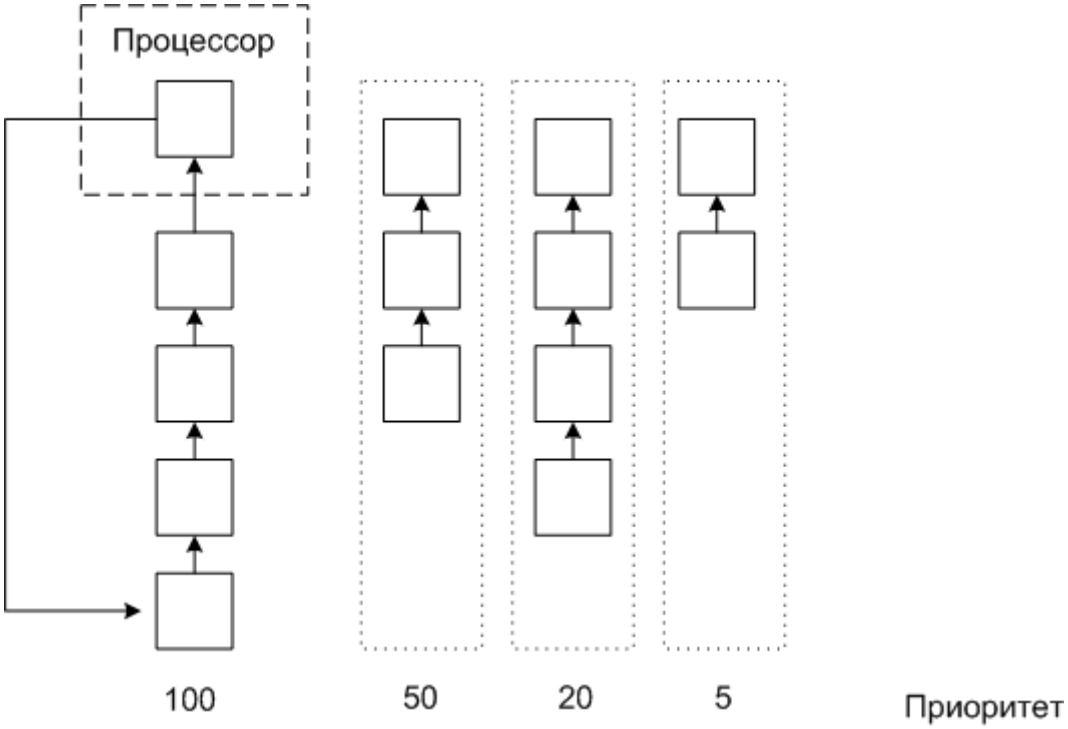
2) Если редкие переключения (квант - 100мс, а время переключения равно 1мс), то происходит увеличение времени ответа на запрос.

4) Планирование с приоритетами

Каждому процессу присваивается **приоритет(числовое значение)**, и управление передается процессу с самым высоким приоритетом.

Приоритет может быть динамический и статический.

Часто процессы объединяют по приоритетам в группы, и используют приоритетное планирование среди групп, но внутри группы используют циклическое планирование.



Приоритетное планирование 4-х групп

Тот же алгоритм кратчайшей работы следующей можно представить, как планирование с приоритетом, где приоритет – наименьшее время работы. Чем меньше до конца обработки процесса, тем выше приоритет.

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.

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

Старвацию ее можно представить в виде очереди и кто привилегированный лезет вне очереди.

Проявляется в алгоритме КРС(кратчайшая работа следующей). Низкоприоритетные запросы могут никогда не выполниться.

Решение проблемы:

Ввести понятие «**старения**»: по мере течения времени увеличивать приоритет процесса.

5)Round-robin

суть метода. Пусть имеется *N* объектов, способных выполнить заданное действие, и *M* задач, которые должны быть выполнены этими объектами. Подразумевается, что объекты *n* равны по своим свойствам между собой, задачи *m* имеют равный приоритет. Тогда первая задача (*m* = 1) назначается для выполнения первому объекту (*n* = 1), вторая — второму и т. д., до достижения последнего объекта (*m*= *N*). Тогда следующая задача (*m* = *N*+1) будет назначена снова первому объекту и т. п.

Проще говоря, происходит перебор выполняющих задания объектов по циклу, или по кругу (round), и по достижении последнего объекта следующая задача будет также назначена первому объекту.

Решение задач может быть дополнительно разбито на кванты времени, причем для продолжения решения во времени нумерация объектов (и, соответственно, назначенные задачи) сдвигается по кругу на 1, то есть задача первого объекта отдается второму, второго — третьему, и т. д., а первый объект получает задачу последнего, либо освобождается для приёма новой задачи. Таким образом, алгоритм Round-robin становится алгоритмом распределения времени или балансировки нагрузки.

Данный алгоритм планирования обозначает циклический алгоритм с вытесняющим планированием.

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

1. Если q меньше, чем время, затрачиваемое на переключение контекста, тогда диспетчер будет неэффективным.

При выполнении процесса возможны два варианта:

1) **Время непрерывного использования процессора, требующееся процессу, (остаток текущего CPU burst) меньше или равно продолжительности кванта времени.** Тогда процесс по своей воле освобождает процессор до истечения кванта времени, на исполнение выбирается новый процесс из начала очереди и таймер начинает отсчет кванта заново.

2) **Продолжительность остатка текущего CPU burst процесса больше, чем квант времени**. Тогда по истечении этого кванта процесс прерывается таймером и помещается в конец очереди процессов готовых к исполнению, а процессор выделяется для использования процессу, находящемуся в ее начале

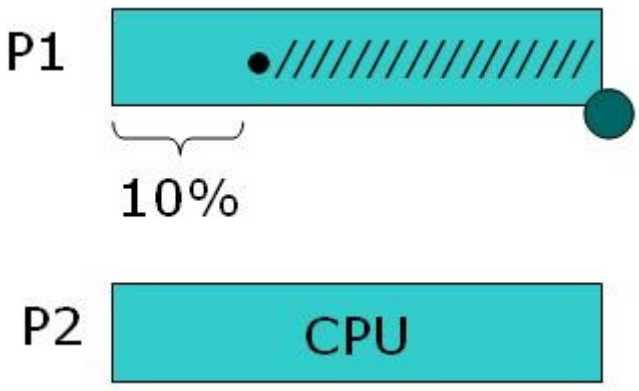
Недостаток Round-robin

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

Недостаток Round-robin Проблема RR в том, что **не учитываются задержки**, и полезное время работы P1 составляет только 10%.

Есть 2 процесса P1 и P2

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



ПРИМЕР: АЛГРИМ СЧИТАЕТ КОРЕНЬ, В ТО ВРЕМЯ ПОКА ПРОЦЕСС ВЫВОДА ОТВЕТА НА ЭКРАН,ПРОСТАЕВАЕТ.

6)Многоуровневые очереди

Для систем, в которых процессы могут быть легко рассортированы на разные группы, был разработан другой класс алгоритмов планирования. Для каждой группы процессов создается своя очередь процессов, находящихся в состоянии готовность (см. рисунок).

Этим очередям приписываются фиксированные приоритеты. Приоритет очереди процессов, запущенных студентами, — ниже, чем для очереди процессов, запущенных преподавателями. Это значит, что ни один пользовательский процесс не будет выбран для исполнения, пока есть хоть один готовый системный процесс, и ни один студенческий процесс не получит в свое распоряжение процессор, если есть процессы преподавателей, готовые к исполнению. Внутри этих очередей для планирования могут применяться самые разные алгоритмы. Так, например, для больших счетных процессов может использоваться алгоритм FCFS, а для интерактивных процессов – алгоритм RR. Подобный подход, получивший название многоуровневых очередей, повышает гибкость планирования: для процессов с различными характеристиками применяется наиболее подходящий им алгоритм.

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

1) Планирование с фиксированным приоритетом

- Вначале обслужить все интерактивные процессы, потом все фоновые
- Возможна старвация

2) Разделение времени

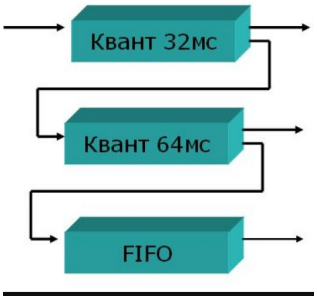
Каждой очереди выделяется часть времени ЦП, которую она может распланировать между своими процессами. Например 80% времени ЦП на интерактивные процессы через RR, 20% на фоновые через FIFO.

3) 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**.

Многоуровневая очередь с обратной связью:



-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 вычислительных ресурсов, то он помещается в последнюю очередь FIFO.

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

Пример многоуровневой очереди с обратной связью

Есть три очереди:

- Q0 –RR с квантом времени t=16мс
- Q1 –RR с квантом времени t=32мс
- Q2 –FIFO

Общее планирование реального времени

Используется модель, когда каждый процесс борется за процессор со своим заданием и графиком его выполнения.

Планировщик должен знать:

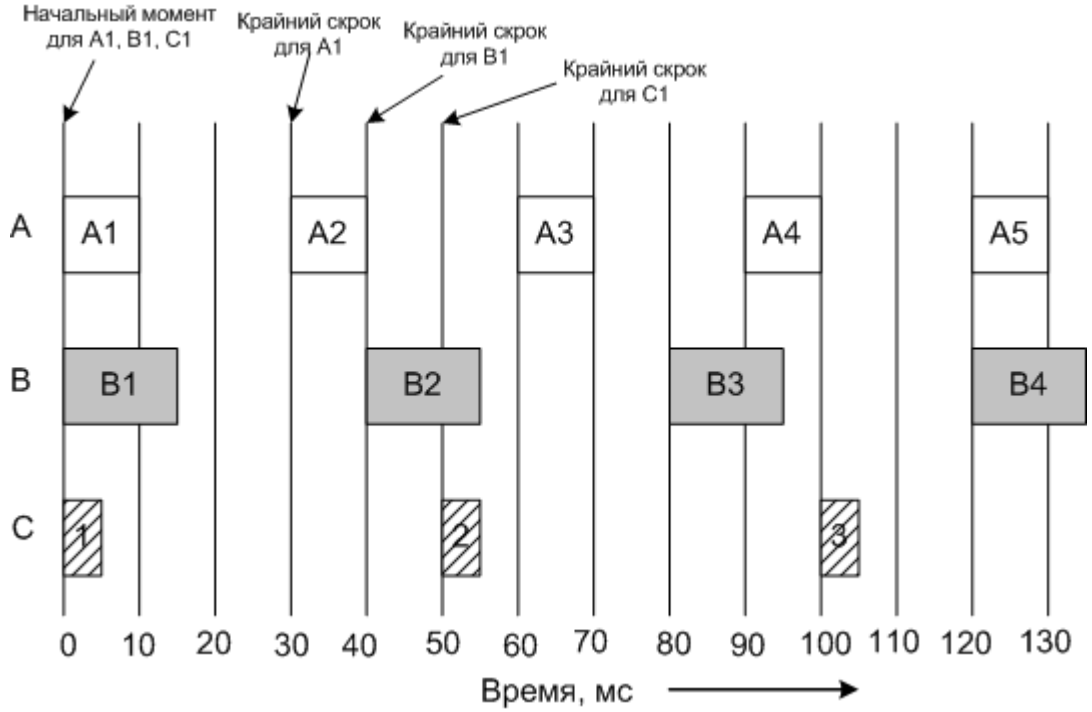
- частоту, с которой должен работать каждый процесс
- объем работ, который ему предстоит выполнить
- ближайший срок выполнения очередной порции задания

Рассмотрим пример из трех процессов.

Процесс **A** запускается каждые 30мс, обработка кадра 10мс

Процесс **B** частота 25 кадров, т.е. каждые 40мс, обработка кадра 15мс

Процесс **C** частота 20 кадров, т.е. каждые 50мс, обработка кадра 5мс



Три периодических процесса

Проверяем, можно ли планировать эти процессы.

10/30+15/40+5/50=0.808<1

Условие выполняется, планировать можно.

Можно планировать эти процессы **статическим** (приоритет заранее назначается каждому процессу) и **динамическим** методами.

Контрольные вопросы:

1) Расскажите суть алгоритм планирования FIFO
2) Расскажите суть алгоритма кратчайшей работы первой
3) Планирование с приоритетом.
4) В чем суть метода Round-robin? Недостатки метода
5) Расскажите про многоуровневую очередь с обратной связью

- Современные операционные системы, Э. Таненбаум, 2002, СПб, Питер, 1040 стр.

- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер, 2001, СПб, Питер, 544 стр.
- Алгоритмы планирования
  Источник: http://life-prog.ru/1_1057_algoritmi-planirovaniya.html
- Алгоритмы планирования
  Источник: http://baumanki.net/lectures/10-informatika-i-programmirovanie/353-operacionnye-sistemy/4804-algoritmy-planirovaniya.html