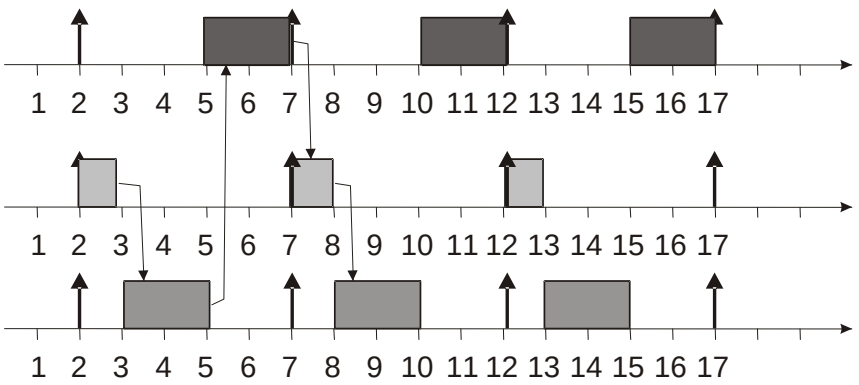
Операционные системы

Планирование процессов
Основные алгоритмы

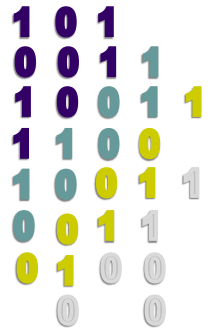
Нельзя Всем дать Все, потому как Всех много, а Всего мало, И всего на всех не хватает

Планирование

- Ресурсы ограничены
  - Вычислительные в том числе
  - Вычислительное устройство 1
  - Процессов несколько
  - Задача распределить все процессы по времени



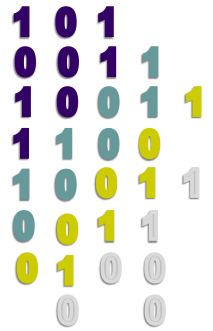
Планирование



- **Планировщик** — это модуль ОС, отвечающий за разделение времени имеющихся процессоров между выполняющимися процессами

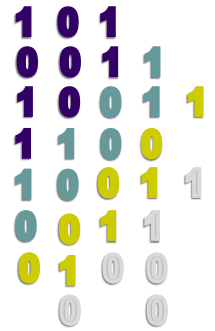
- Планировщик может принимать решения о выборе для исполнения нового процесса, из числа находящихся в состоянии ***готовность***

Уровни планирования



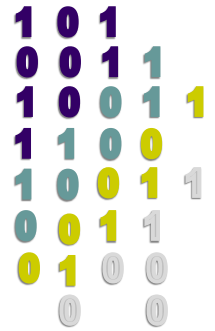
- Планирование заданий
  - В системах пакетного режима
  - Выбор какое задание из пакета сейчас будет исполняться
- Планирование использования процессора
  - Появляется только в мультипроцессных системах
    - В состоянии «готовность» могут находиться несколько процессов одновременно

Планирование заданий



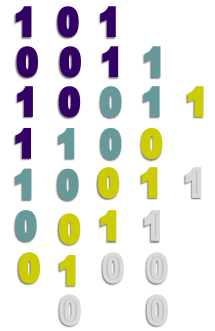
- Отвечает за порождение новых процессов
  - Определяет её степень мультипрограммирования, т.е. среднее количество процессов в системе
    - Новые процессы появляются после исчезновения старых
  - Осуществляется реже
  - **Долгосрочное планирование**
  - Существует не во всех системах
    - В системах общего назначения нет необходимости

Планирование использования процессора



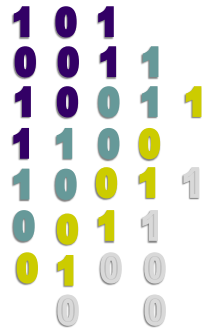
- Краткосрочное планирование
- ~ милисекунды
  - Выбор процесса для планирования оказывает влияние на работу системы до следующего выбора
- Свопинг
  - Сбрасывание процесса на диск
  - Среднесрочное планирование

Цели планирования



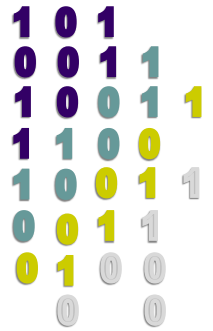
- Выбор алгоритма планирования определяется целями, которые необходимо достичь
  - Справедливость
  - Эффективность
  - Сокращение полного времени выполнения
  - Сокращение времени ожидания
  - Сокращение времени отклика

Цели планирования



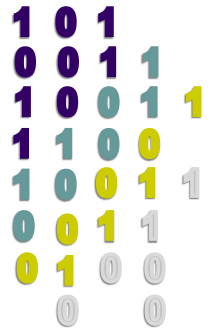
- Справедливость
  - Гарантировать каждому процессу определённую часть времени
- Эффективность
  - Занять процессор на 100% времени, без ожидания процессов готовых к исполнению
  - В реальных системах ~ от 40 до 90 процентов

Цели планирования Сокращение…



- Полного времени выполнения
  - минимизировать время между стартом процесса и его завершением
- Времени ожидания (*waiting time*)
  - минимизировать время, которое проводят процессы в состоянии ***готовность***
- Времени отклика (*response time*)
  - минимизировать время, которое требуется процессу для ответа на запрос пользователя

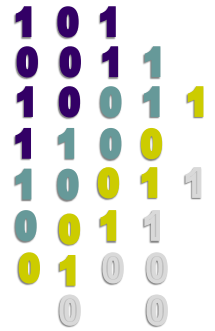
Критерии эффективности алгоритмов



- Предсказуемость
  - Одинаковые задания – одинаковое время
- Минимальные накладные расходы
  - «100 миллисекунд»
- Равномерная загрузка различных ресурсов
- Масштабируемость
  - Рост накладных расходов при росте количества планируемых процессов

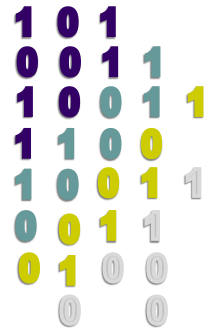
Идеального алгоритма быть не может!

Параметры планирования



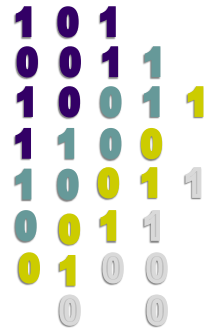
- Для выполнения планирования необходимо учитывать параметры
  - Параметры
    - Системы
    - Процессов
  - Статические
    - Не изменяются в процессе функционирования
  - Динамические

Параметры системы



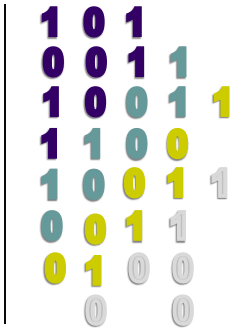
- Статические
  - Значения ресурсов
    - частота процессора
    - размер оперативной памяти
    - максимальное количество памяти на диске
    - количество подключённых устройств ввода-вывода
    - …
  - Динамические
    - количество свободных ресурсов в текущий момент времени

Параметры процессов



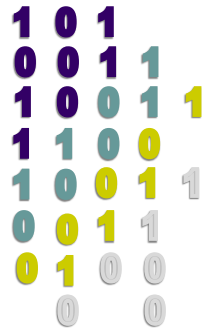
- Статические — известные на этапе загрузки
  - владелец процесса
  - приоритет
  - сколько процессорного времени запросил пользователь
  - какая часть времени тратится на I/O операции
  - каков объем ресурсов необходимых заданию
    - оперативная память, устройства ввода-вывода, специальные библиотеки и системные программы

Параметры процессов



- Динамические
  - Объем занимаемой оперативной памяти
  - Время ожидания процессора
    - Время выгрузки в своп
  - Время использования процессора
  - «Интерактивность»
  - …
- Используются краткосрочными (среднесрочными) алгоритмами

Динамические параметры процессов



```
a=1;
read(b, c);
<input b>
<input c>
a=a^2 + b/c;
print(a);
<output a>
```

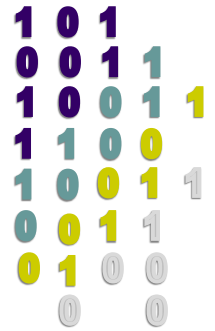
CPU burst

I/O burst

CPU burst

I/O burst

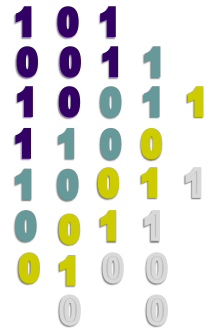
Параметры



- ***CPU burst*** — *п*ромежуток времени непрерывного использования процессора

- ***I/O burst*** — промежуток времени непрерывного ожидания ввода-вывода

Планирование



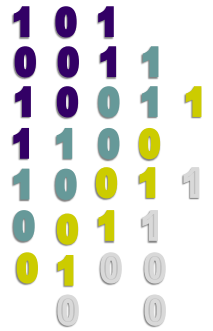
- Планировщик включается в работу при переходе работы между состояниями

  - ***исполнение*** → ***завершение***
  - ***исполнение*** → ***ожидание***
  - ***исполнение*** → ***готовность***
  - ***ожидание*** → ***готовность***

Планирование обязательно

Планирование НЕ обязательно

Планирование с вытеснением и без

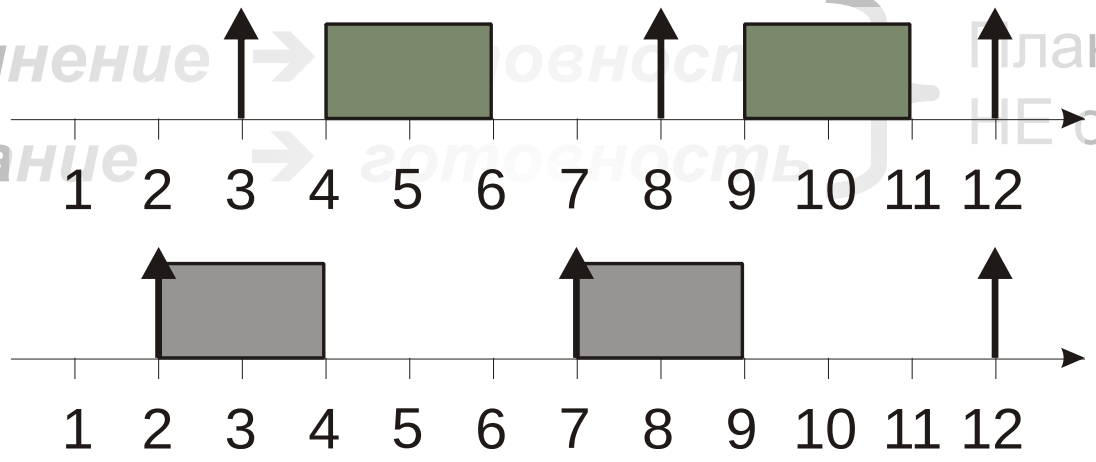


- Планирование без замещения
  - Если задача начала выполняться она выполняется до полного завершения
  - Приводит к более низкой планируемости
  - Меньшие накладные расходы благодаря малому количеству переключений контекста
  - Планирование **только** в ситуациях 1 и 2
    - *nonpreemptive scheduling*

Невытесняющее планирование

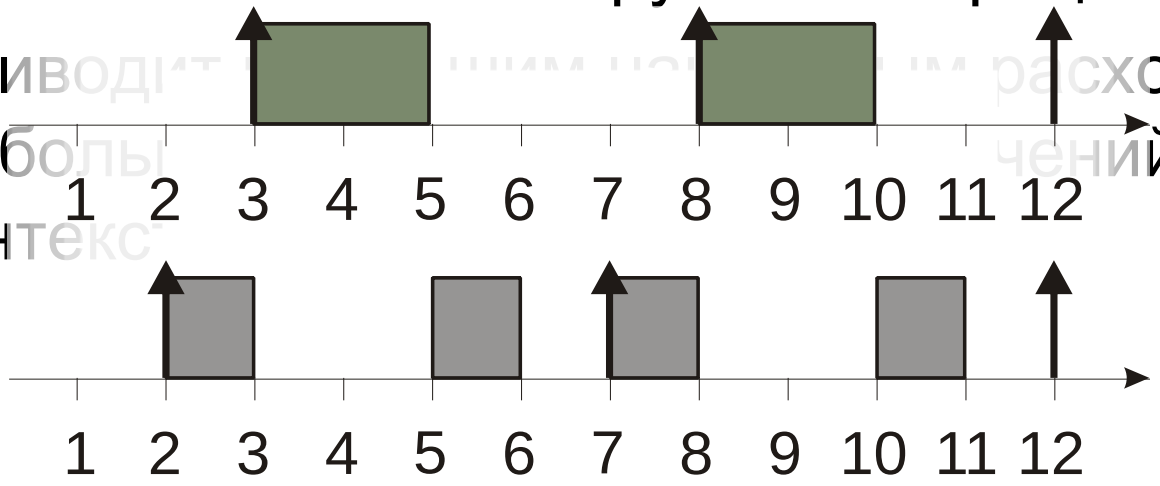
- Планировщик включается в работу при переходе работы между состояниями
  - ***исполнение → завершение***
  - ***исполнение → ожидание***

Планирование обязательно

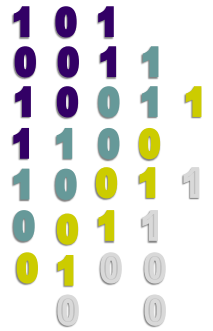


Планирование с вытеснением

- Выполнение задания прерывается и продолжается позже
  - Вытеснение осуществляется для выполнения более высокоприоритетного процесса
- Более высокая планируемость процессов
- Приводит к дополнительным накладным расходам из-за боль чений контекс



Планирование



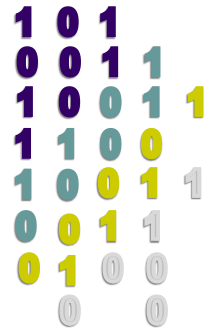
- Планировщик включается в работу при переходе работы между состояниями

  - ***исполнение*** → ***завершение***
  - ***исполнение*** → ***ожидание***
  - ***исполнение*** → ***готовность***
  - ***ожидание*** → ***готовность***

Планирование обязательно

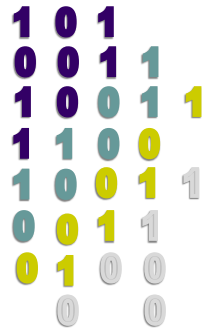
Планирование НЕ обязательно

Алгоритмы планирования



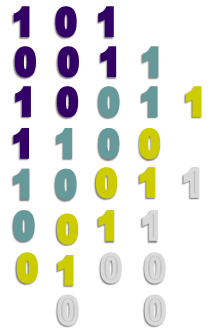
- Предназначены для достижения различных целей
- Эффективны для разных классов задач
- Многие могут быть использованы на нескольких уровнях планирования
  - FCFS
  - RR
  - SJF
  - …

First-Come, First-Served (FCFS)



- Невытесняющее планирование
- Когда процесс переходит в состояние **готовность**, ссылка на его PCB, помещается в конец очереди
- Задачи извлекаются из начала очереди
  - Запись удаляется из очереди
  - FIFO — это дисциплина работы алгоритма FCFS

FCFS

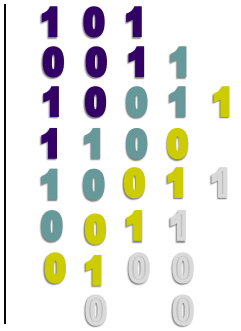


- Невытесняющее планирование
- Процесс занимает процессор до истечения своего текущего CPU burst
  - После этого для выполнения выбирается новый процесс из начала очереди

+ Лёгкость реализации

– Низкая эффективность с точки зрения времён реакции и ожидания

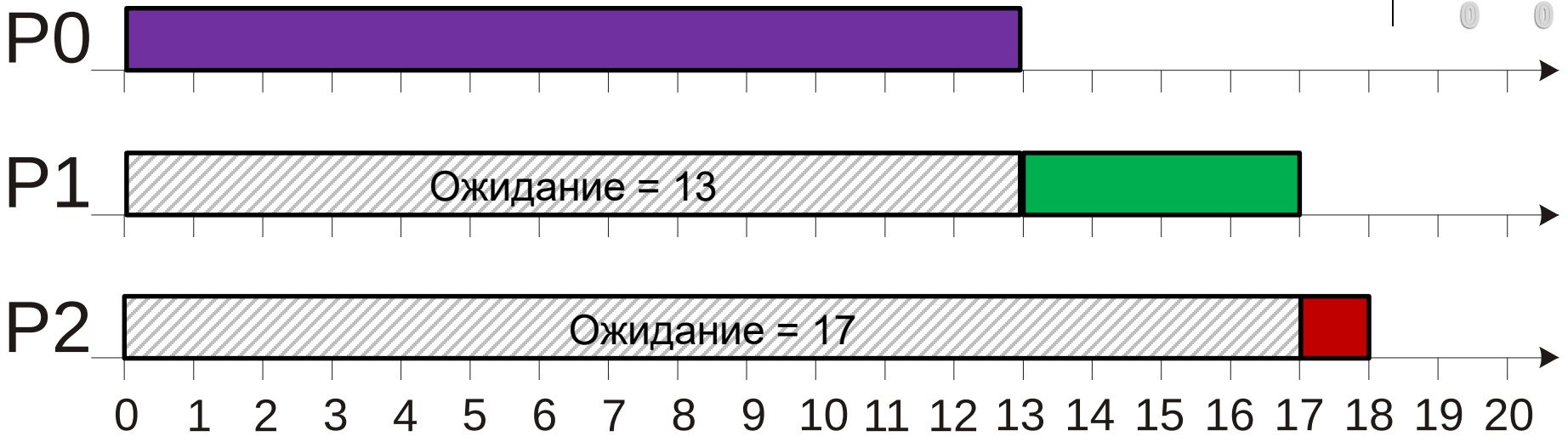
Пример FCFS



- Рассмотрим набор процессов
  - P_0 — CPU burst = 13
  - P_1 — CPU burst = 4
  - P_2 — CPU burst = 1

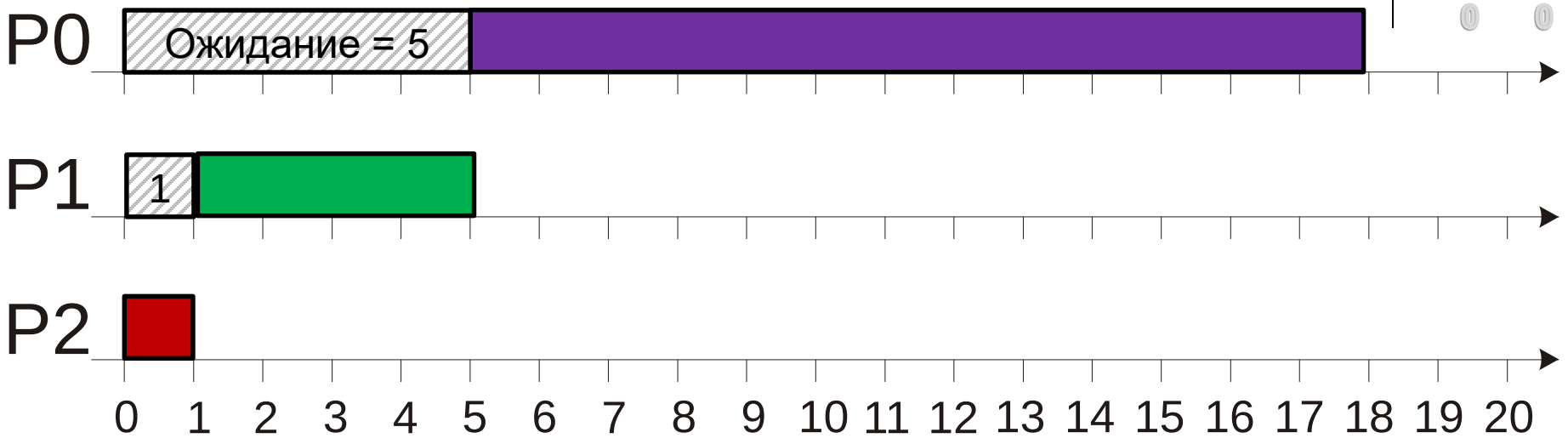
- Только один CPU burst
  - Нет I/O burst

P_0 (13) → P_1 (4) → P_2 (1)



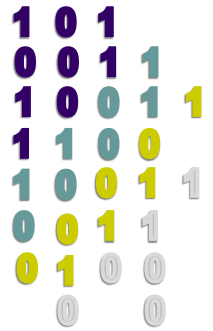
- Среднее время ожидания
  - (0 + 13 + 17) / 3 = **10**
- Среднее полное время выполнения
  - (13 + 17 + 18) / 3 = **16**

P_2 (1) → P_1 (4) → P_0 (13)



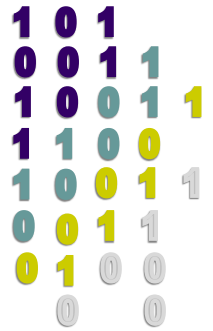
- Среднее время ожидания
  - (0 + 1 + 5) / 3 = **2** **В 5 раз лучше!**
- Среднее полное время выполнения
  - (18 + 5 + 1) / 3 = **8** **В 2 раза лучше!**

FCFS



- Временные характеристики существенно зависят от порядка процессов в очереди
- Времена ожидания могут быть огромны
- Не подходит для систем с разделением времени
  - Интерактивные приложения становятся неинтерактивными

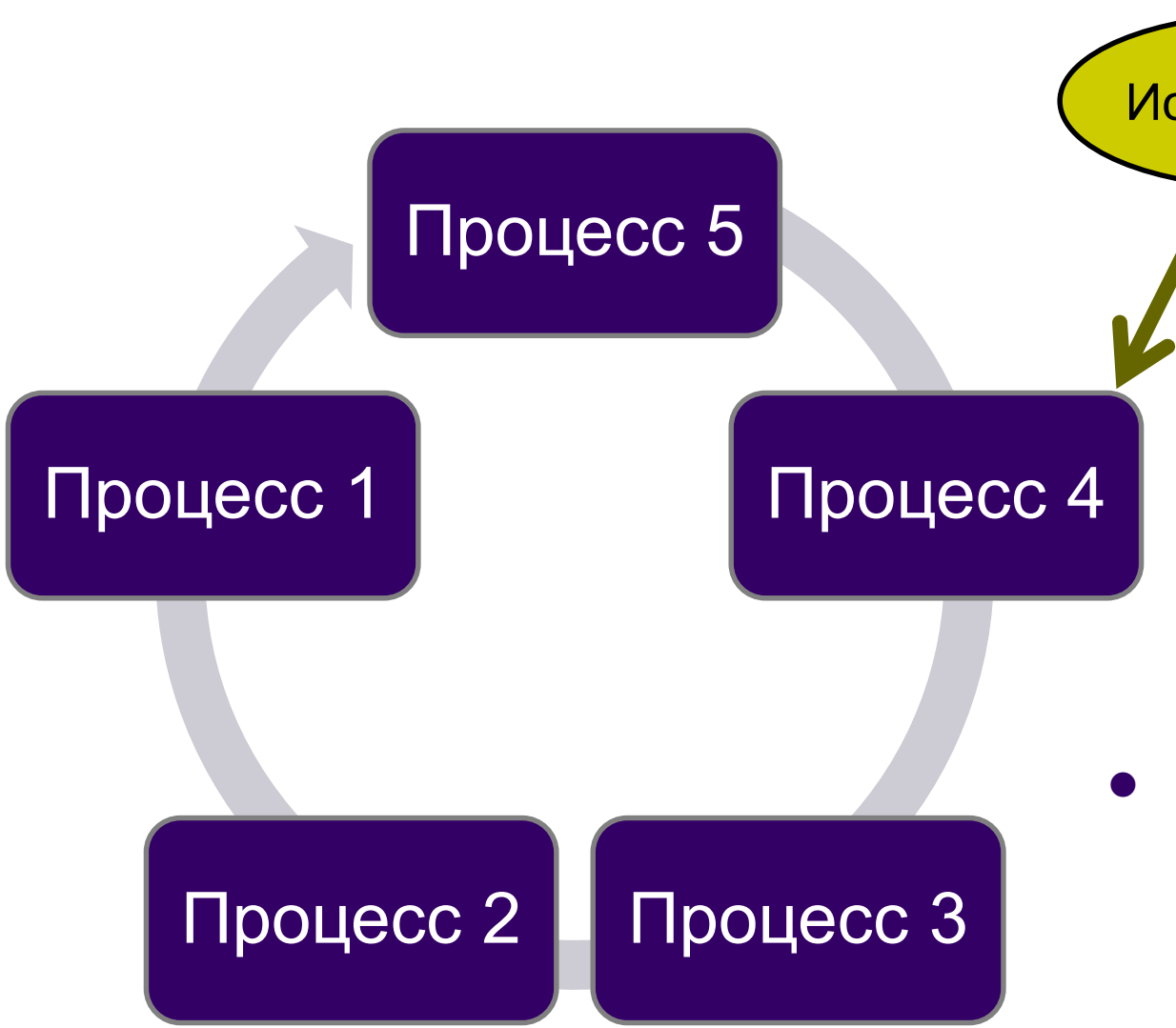
Round Robin (RR)



- Вид детской карусели в США

- Модификация алгоритма FCFS
  - Только в режиме вытесняющего планирования
  - Множество готовых процессов организованным циклически
    - Процессы «сидят на карусели»
    - Каждый процесс «проезжает рядом с процессором» заранее отведённый квант времени: 10 – 100 млс

RR



Исполнение

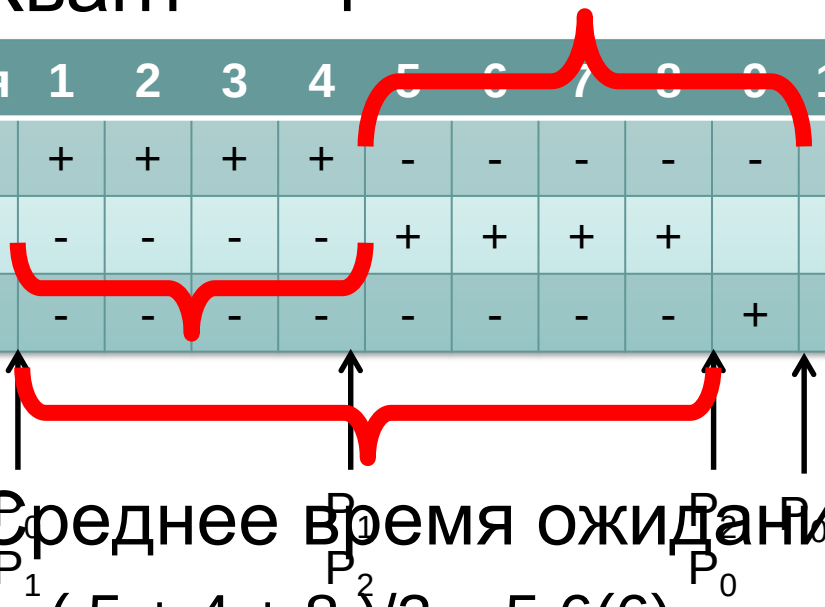
Процессор

- Освобождение процессора
  - Принудительное
  - Самостоятельное

RR пример

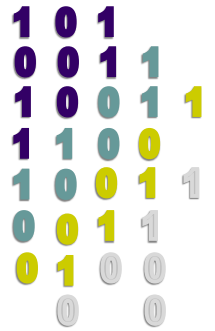
- P_0 — 13; P_1 — 4; P_2 — 1
  - Порядок P_0 → P_1 → P_2
- Квант — 4

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P_0	+	+	+	+	-	-	-	-	-	+	+	+	+	+	+	+	+	+
P_1	-	-	-	-	+	+	+	+										
P_2	-	-	-	-	-	-	-	-	+									



- Среднее время ожидания
  - (5 + 4 + 8)/3 = 5,6(6)
- Среднее полное время выполнения
  - (18 + 8 + 9)/3 = 11,6(6)

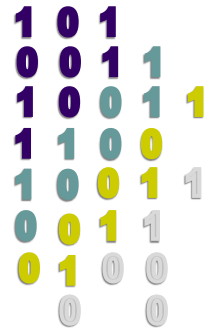
RR



- Эффективность для обратного порядка процессов не хуже чем у FCFS

- Эффективность
  - Также зависит от порядка процессов
  - Существенно зависит от величины кванта времени

RR пример

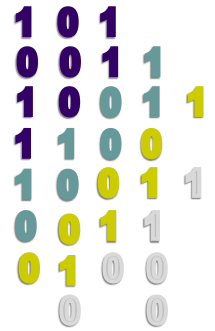


- P_0 — 13; P_1 — 4; P_2 — 1
  - Порядок $P_0 \rightarrow P_1 \rightarrow P_2$
- Квант — 1

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P_0	+	-	-	+	-	+	-	+	-	+	+	+	+	+	+	+	+	+
P_1	-	+	-	-	+	-	+	-	+									
P_2	-	-	+															

- Среднее время ожидания
  - (5 + 5 + 2)/3 = 4
- Среднее полное время выполнения
  - (18 + 9 + 3)/3 = 10

Величина кванта RR

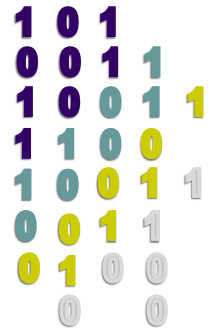


- При очень больших величинах вырождается

RR ~ FCFS

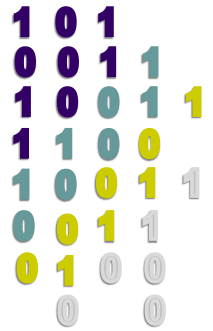
- При очень малых величинах создаётся иллюзия параллельной работы нескольких процессов
  - Высокие накладные расходы

Shortest-Job-First (SJF)



- Эффективность предыдущих алгоритмов сильно зависела от порядка процессов
  - Лучше раньше исполнять короткие задачи
  - Теория расписаний
- Если знать CPU burst процессов в состоянии **готовность**
  - Можно выбирать «кротчайшую» работу
  - При равенстве применять FCFS
    - Отсутствует квантование

SJF



- Алгоритм краткосрочного планирования
- Невытесняемый
- Вытесняемый
  - Если в системе появился процесс с CPU burst меньше чем осталось исполнить текущему процессу
  - То текущий процесс замещается

Пример SJF

Процесс	1	2	3	4
CPU Burst	5	3	7	1

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P_0	-	-	-	-	+	+	+	+	+									
P_1	-	+	+	+														
P_2	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+		
P_3	+																	

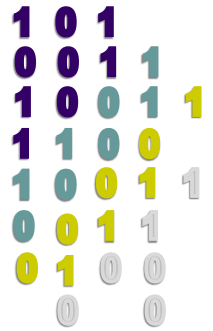
- Среднее время ожидания
  - (4 + 1 + 9 + 0)/4 = 3,5

FCFS

7

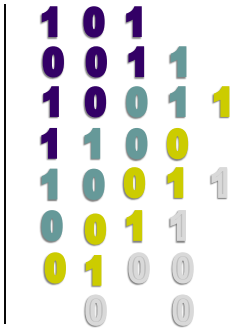
- Является оптимальным с точки зрения минимизации среднего времени ожидания среди класса всех невытесняющих алгоритмов

SJF



- Основная проблема — необходимо знать **CPU Burst**
  - В системах пакетного режима пользователь заявляет примерное время исполнения
    - Чем больше задание, тем позже оно запустится
  - Но для кратковременного планирования приходится опираться на предыдущее поведение процесса

Предсказание CPU Burst



- Производится с помощью рекуррентного соотношения

$$T_{n+1} = \alpha \cdot \tau_n + (1 - \alpha)T_n$$

- Если α=0, то игнорируем поведение процесса

$$T_{n+1} = T_n = T_{n-1} = \cdots = T_0$$

  - Предсказание по первому значению

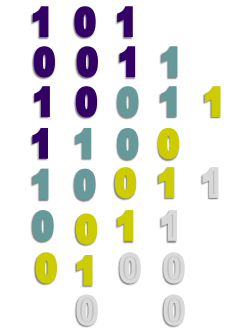
- Если α=1, то забываем про всю историю

$$T_{n+1} = \tau_n$$

  - Предсказание по предыдущему значению

- Обычно $\alpha = 1/2$

Гарантированное планирование



- Пусть в системе N пользователей
  - Они должны получить 1/N времени
  - T_i — длительность сеанса пользователя i
  - τ_i — выделенное процессорное время

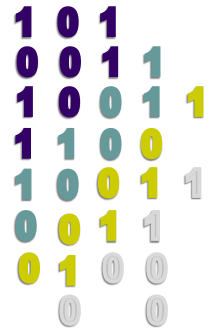
$$\tau_i \sim \frac{T_i}{N}$$

  - Коэффициент справедливости

$$\frac{\tau_i N}{T_i}$$

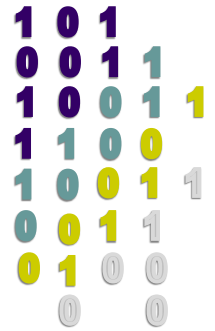
    - Выбирать процесс с наименьшей величиной

Приоритетное планирование



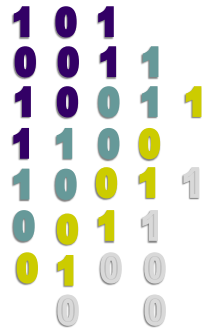
- Каждому процессу присваивается определённое числовое значение — **приоритет**
  - Алгоритмы SJF и гарантированного планирования — частные случаи
- Принципы назначения приоритетов
  - Внешние и внутренние критерии
- Может быть вытесняющим и невытесняющим
  - Процесс с высоким приоритетов замещает другие

Виды приоритетов



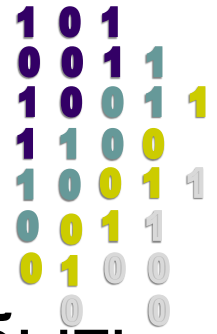
- Статические
  - Не изменяются во времени
  - Проще реализовать
  - Меньше накладных расходов
- Динамические
  - Приоритет изменяется в моменты изменения состояния системы
  - Более гибкое планирование
  - Высокая эффективность планирования
  - SJF и алгоритм гарантированного планирования

Особенности приоритетного планирования



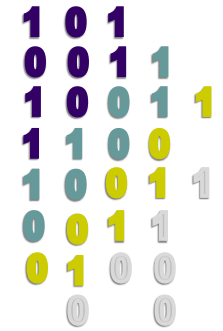
- Достаточно гибко, но…
- Нет гарантии, что низкоприоритетные процессы запустятся
  - При остановке IBM 7094 в MIT в 1973 году были найдены процессы, запущенные в 1967 году и ни разу с тех пор не исполнявшиеся
- Для решения проблемы применяется схема с повышением приоритетов

Многоуровневые очереди (Multilevel Queue)



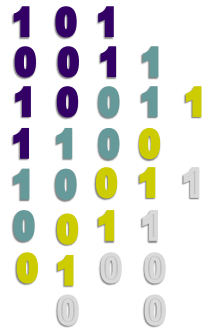
- Для систем, в которых процессы могут быть рассортированы на разные группы
  - Для каждой группы создаётся очередь
  - Очереди получают свой приоритет (вытесняемо)
  - Внутри группы процессы планируются по своему
    - Процессы из группы «фоновые» — FCFS
    - «интерактивные» — RR
    - И т.д.
  - Большая гибкость планирования
  - Подходящий алгоритм, для разных процессов

Многоуровневые очереди

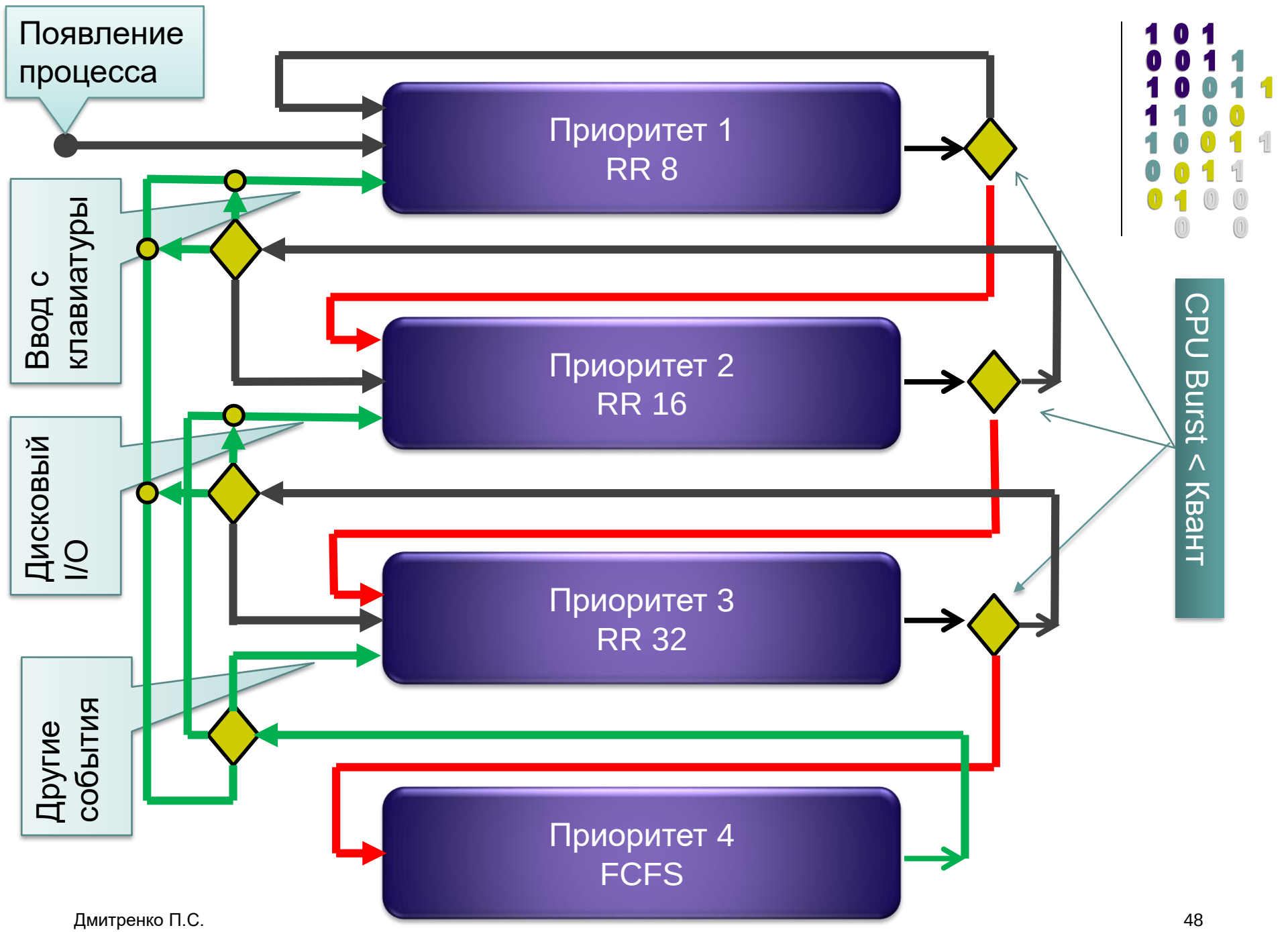


- Процессы ядра (FCFS)
- Интерактивные пользовательские (RR 10)
- Фоновые системные (FCFS)
- Фоновые пользовательские (RR 50)

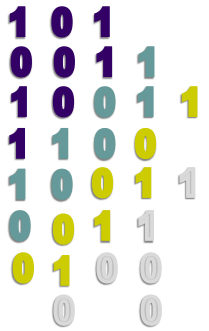
Многоуровневые очереди с обратной связью



- **Multilevel Feedback Queue**
- Процесс может переходить между очередями в зависимости от своего поведения
- Если **CPU Burst** процесса больше выделенного ему кванта, то он переводится ниже между очередями
  - Иначе не меняется


Появление процесса
Приоритет 1
RR 8
Ввод с клавиатуры
Приоритет 2
RR 16
Дисковый I/O
Приоритет 3
RR 32
Другие события
Приоритет 4
FCFS
CPU Burst < Квант

Многоуровневые очереди с обратной связью



- Трудоёмкий в реализации
- Наибольшая гибкость
- Один из наиболее общих подходов к планированию процессов
  - Количество очередей для процессов, находящихся в состоянии ***готовность***
  - Алгоритм планирования между очередями
  - Алгоритмы планирования внутри очередей
  - Правила порождения процессов
  - Правила перевода процессов из одной очереди в другую

Вопросы?