Операционные системы

Курс лекций для гр. 4057/2

Лекция 2

Содержание

Раздел 1. Управление процессами

<…Лекция 1…>

1.3 Создание и уничтожение процессов

1.4 Потоки

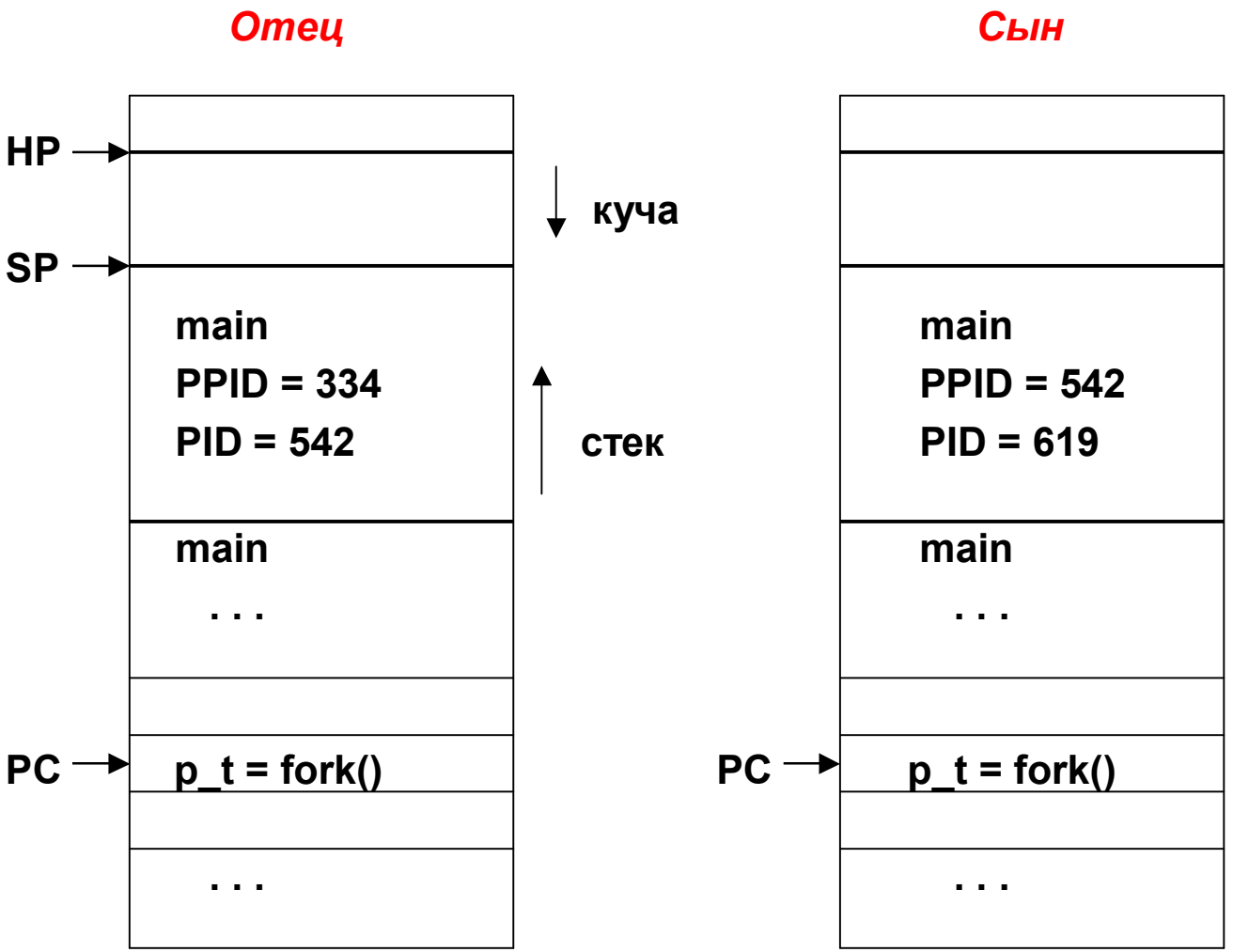
1.5 Диспетчерские состояния процессов

1.6 Диспетчеризация центрального процессора

Создание процессов

- При создании процесс получает имя, PCB, начальные ресурсы (прежде всего, область памяти) и *тело* (код). Он порождается *родительским* процессом.
- В Win32 функция API 32 *CreateProcess* активизирует файл образа процесса (.exe), передаваемый как параметр
- В Unix процесс-сын наследует тело отца, вызвавшего функцию fork()

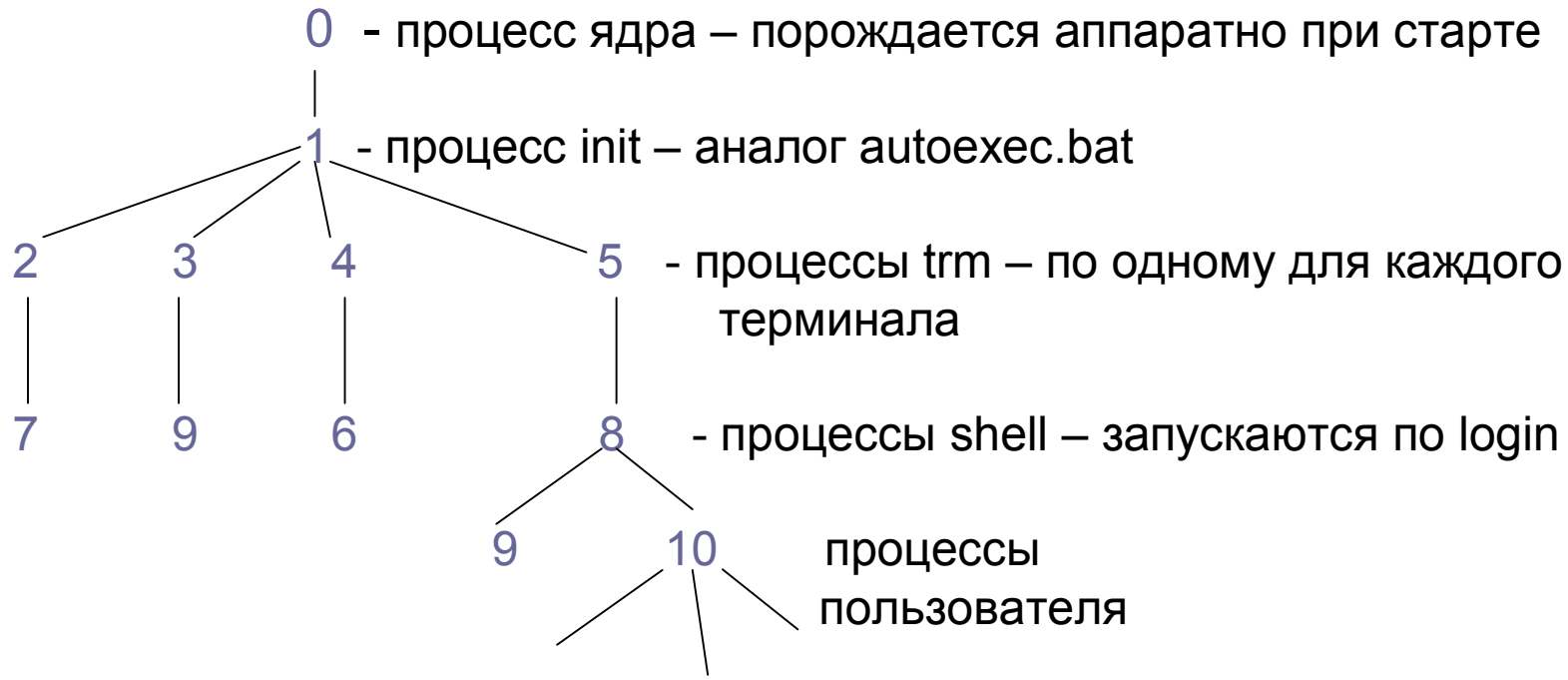
Создание процесса в Unix



HP - указатель кучи,
SP - указатель стека
PC - счетчик команд
PID – идентификатор (порядковый номер процесса),
PPID – идентификатор отца

fork() возвращает PID сына отцу, 0 – сыну и -1 при неудаче.

Генеалогия процессов в Unix



✓ Для выполнения каждой введенной команды shell порождается новый процесс, который уничтожается после выполнения команды

✓ Если ввести & после команды, то она будет выполняться параллельно с shell, иначе следующая команда должна ждать, пока не выполнится первая

Уничтожение процессов

- При уничтожении процесса ОС освобождает его ресурсы. Процесс может уничтожить себя при завершении (в Unix – вызовом exit, в Win32 - exitProcess) или другой процесс (в Unix – вызовом kill)
- В некоторых ОС процесс не может завершиться, пока не завершились все его сыновья
- В Unix если отец уничтожается, его сыновья тоже уничтожаются
- В W2k функция ExitProcess завершает процесс с уведомлением всех подключенных DLL (т.е., сыновних процессов), а функция GetExitCodeProcess возвращает код его завершения

Потоки

- *Потоки* или *нити* (threads) – это «облегченные процессы»: потоки управления, действующие в одном и том же контексте памяти обычного процесса и совместно использующие его ресурсы: адресное пространство (тело процесса), время процессора, открытые файлы и т.д.
- Преимущества потоков перед процессами:
  - Поток имеет сокращенный контекст (только PSW и стек), поэтому переключение потоков гораздо быстрее (на 1-2 порядка)
  - Взаимодействие между потоками одного процесса через общее пространство памяти проще по сравнению с передачей сообщений между процессами или явным объявлением совместно используемой памяти

Потоки хорошо соответствуют следующим задачам:

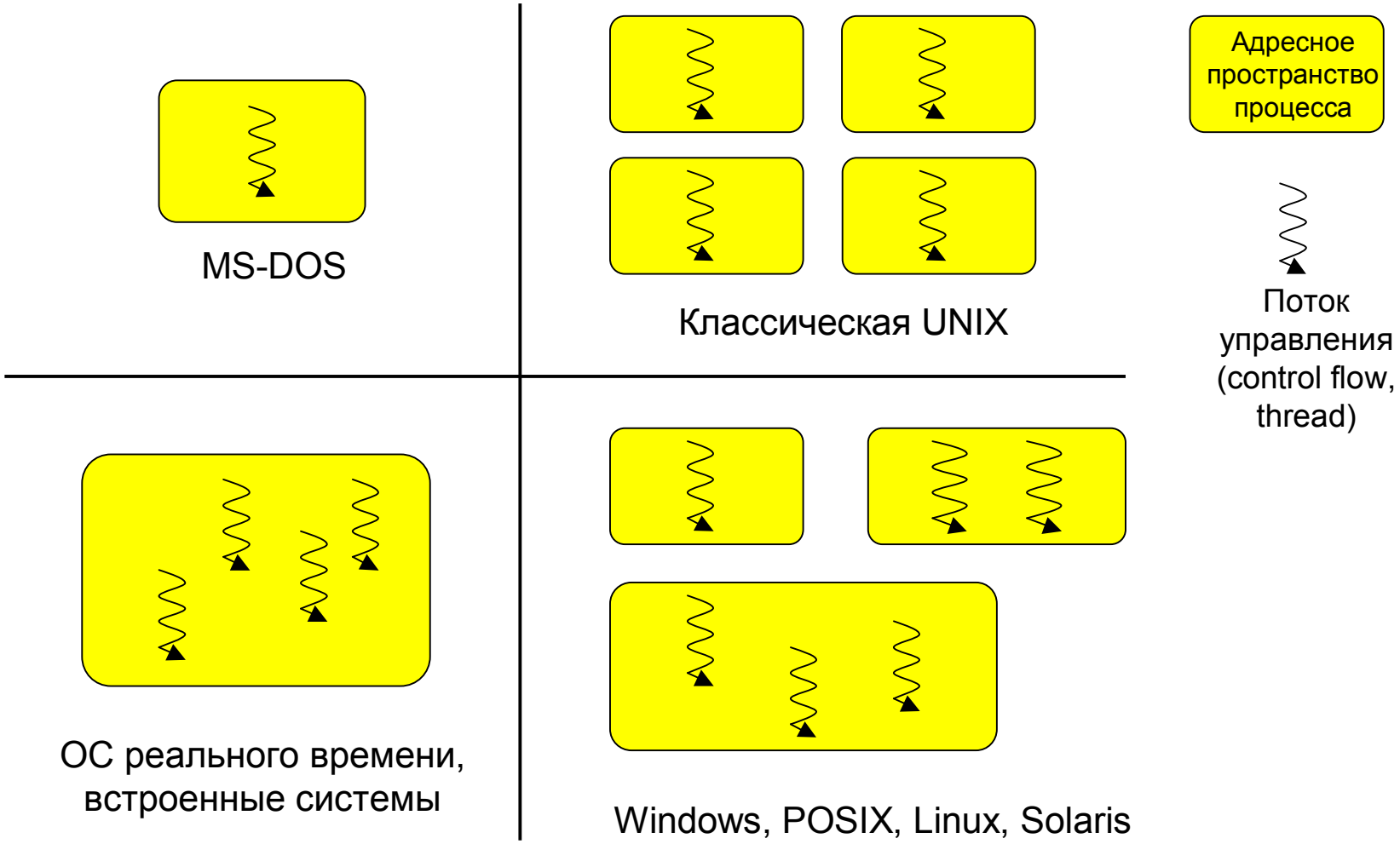
- Распараллеливание активности внутри одного процесса: потоки выполняют разные части одной и той же программы параллельно с целью:
  - повышения производительности (Вопрос 1)
  - лучшей структуризации кода

(Вопрос 2)

- Выполнение одного и того же кода для нескольких параллельных задач (напр., обработка сервером нескольких клиентских запросов одновременно)

(Вопрос 3)

Потоки и процессы: варианты отношений



Два вида потоков

- *Потоки ядра* – ОС управляет ими и диспетчеризует наравне с процессами
- *Потоки пользователей* – невидимы для ОС. Их создание/удаление и диспетчеризацию осуществляет сам процесс – в коде, использующем библиотеку потоков, которая поддерживает таблицу потоков процесса и функции работы с ними – все это в пространстве пользователя

Преимущества вторых перед первыми:

- Скорость: при их переключении нет системных вызовов, поэтому оно гораздо (на один-два порядка) быстрее («сверхлегкие процессы»)
- Гибкость: каждый процесс может применять свой особенный алгоритм диспетчеризации своих потоков

Недостатки потоков пользователей:

- Незащищенность от ошибочного взаимовлияния
- ОС не знает о существовании потоков, поэтому возможны плохие диспетчерские решения:
  - процессу с простаивающими потоками выделяется время процессора
  - блокирование одного потока приводит к ожиданию остальных в том же процессе

(Вопрос 4)

Параллелизм в вычислительных системах

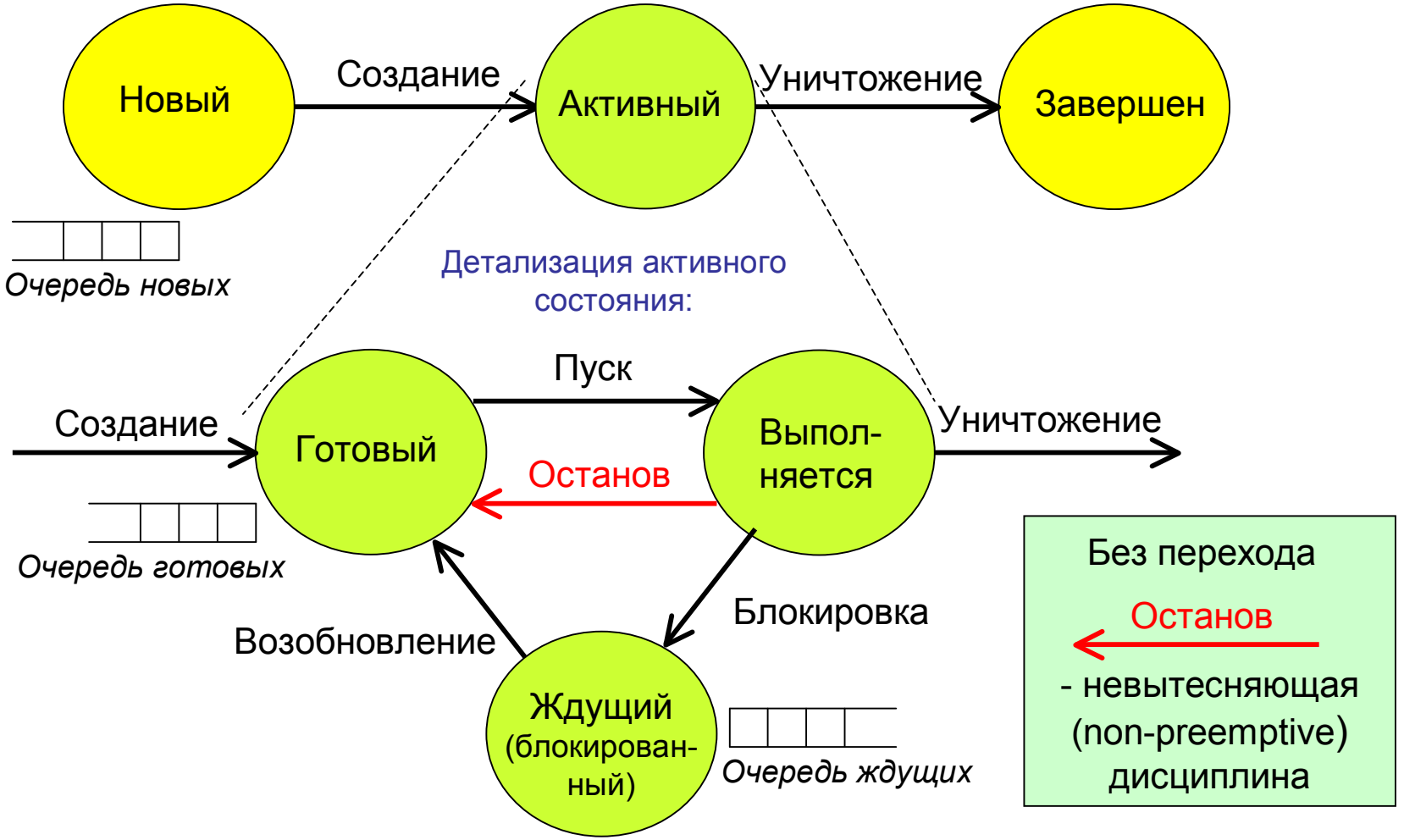
Параллельными (concurrent) называют одновременные или квазиодновременные (путем разделения времени процессора) активности

Виды параллелизма:

- Мультипрограммирование (многозадачность): одновременное выполнении команд в процессоре для одной программы и ввод-вывод на внешние устройства или ожидание – для других
- Многопоточность – параллельное выполнение нескольких потоков одной программы (multithreading) – многозадачность на уровне потоков
- Мультипроцессирование – мультипрограммирование в многопроцессорной системе
  - многозадачность и многопоточность в многоядерных процессорах
  - массивная параллельная обработка в суперкомпьютерах (на десятках, сотнях процессоров и более)
- Распределенная обработка в сетевых системах: взаимосвязанные и взаимодействующие программы (процессы) одновременно выполняются на различных компьютерах сети

Параллельное программирование (concurrent or parallel programming) – область компьютерной науки, посвященная методам программной реализации этих видов параллелизма

Диспетчерские состояния процессов



Вытесняющая (preemptive) дисциплина диспетчеризации (Вопросы 5, 6)

Планирование (scheduling) процессов

- Долгосрочное (long-term) планирование – управление очередью *новых* процессов
  - Долгосрочный планировщик работает в системах пакетной обработки и мощных серверах; его цель – максимальная производительность (коэффициент мультипрограммирования)
- Оперативное (short-term) планирование = диспетчеризация центрального процессора – управление очередью *готовых* процессов
  - Программа-диспетчер (или планировщик, scheduler) – одна из центральных программ ядра

Критерии эффективности алгоритмов диспечеризации

Пользовательские

1. Время обслуживания, или оборота (turnround time): длительность пребывания процесса в системе от его создания до завершения
2. Время отклика (response time): длительность промежутка времени между моментом перехода процесса в готовое состояние до его следующего запроса вв/вы → реактивность

Системные

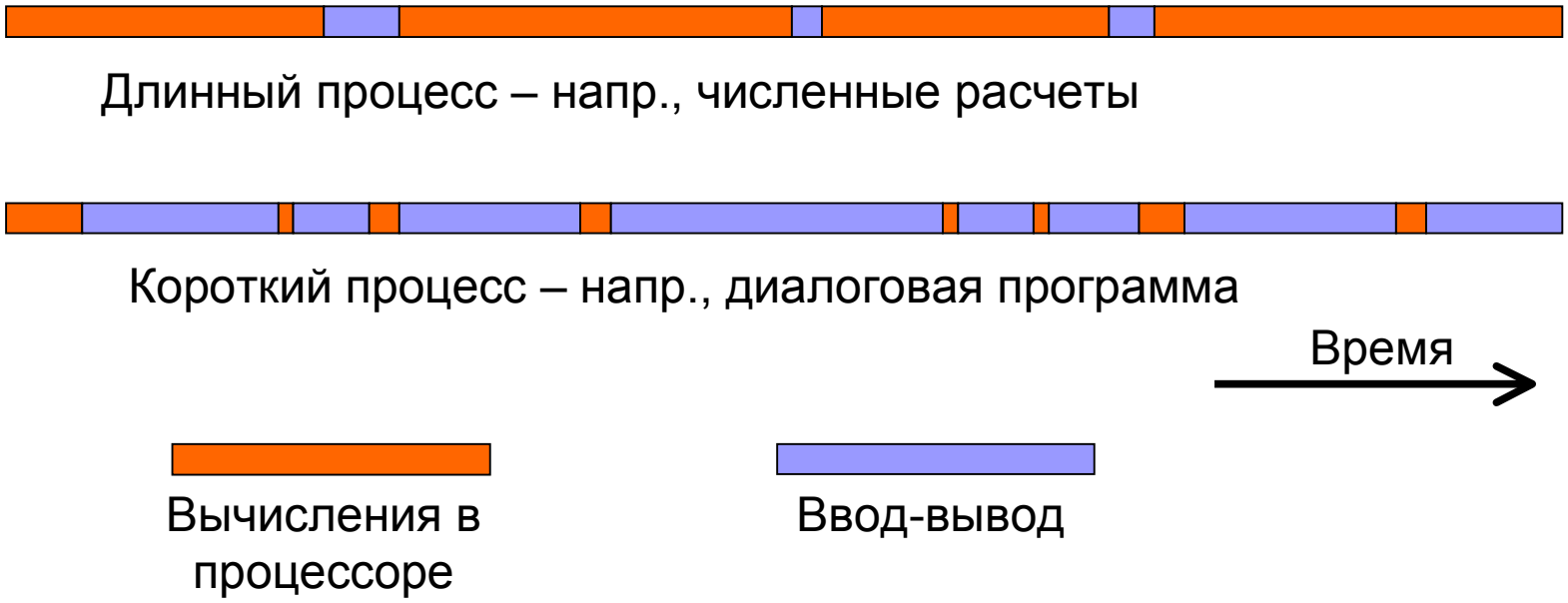
3. Пропускная способность: число процессов, выполняемых в единицу времени → производительность
4. Справедливость: предоставление каждому процессу справедливой доли процессорного времени (напр., пропорционально «длине» процесса) и отсутствие *«голодания»* - неопределенно долгого ожидания процессора

- Критерии статистические; оцениваются средние значения (мат. ожидания) и разброс (дисперсии) величин
- Обычно ищется компромисс между максимизацией критерия 3 и минимизацией 1-2, с учетом 4

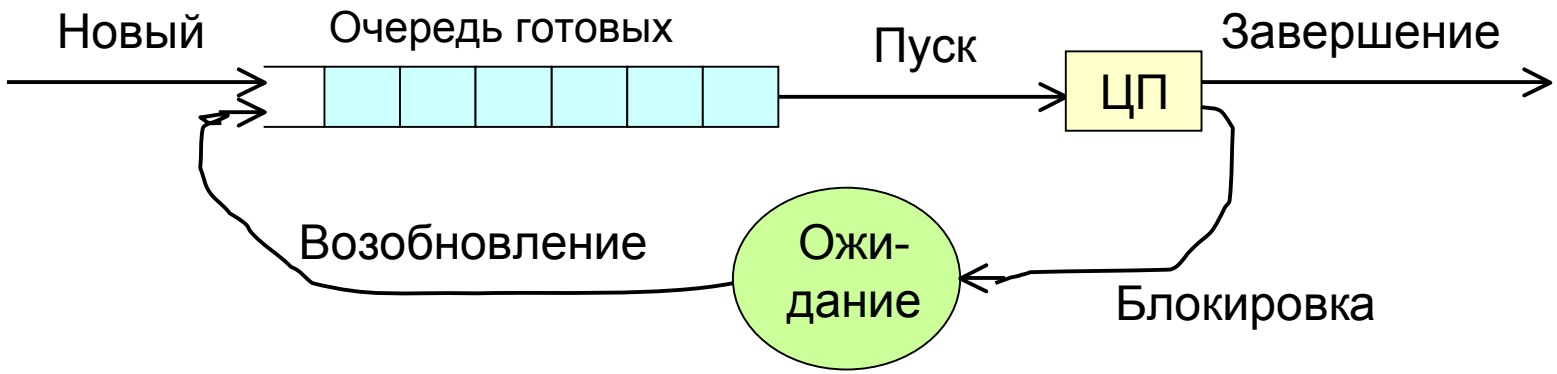
(Вопрос 7)

Уточняющие предположения

- Однопроцессорные системы
- Процессы независимы и с одним потоком каждый
- Выделим два типа процессов: длинные и короткие:



Алгоритм FCFS
(First Come, First Served)



Процессы выполняются на процессоре в порядке поступления, причем дисциплина диспетчеризации - невытесняющая

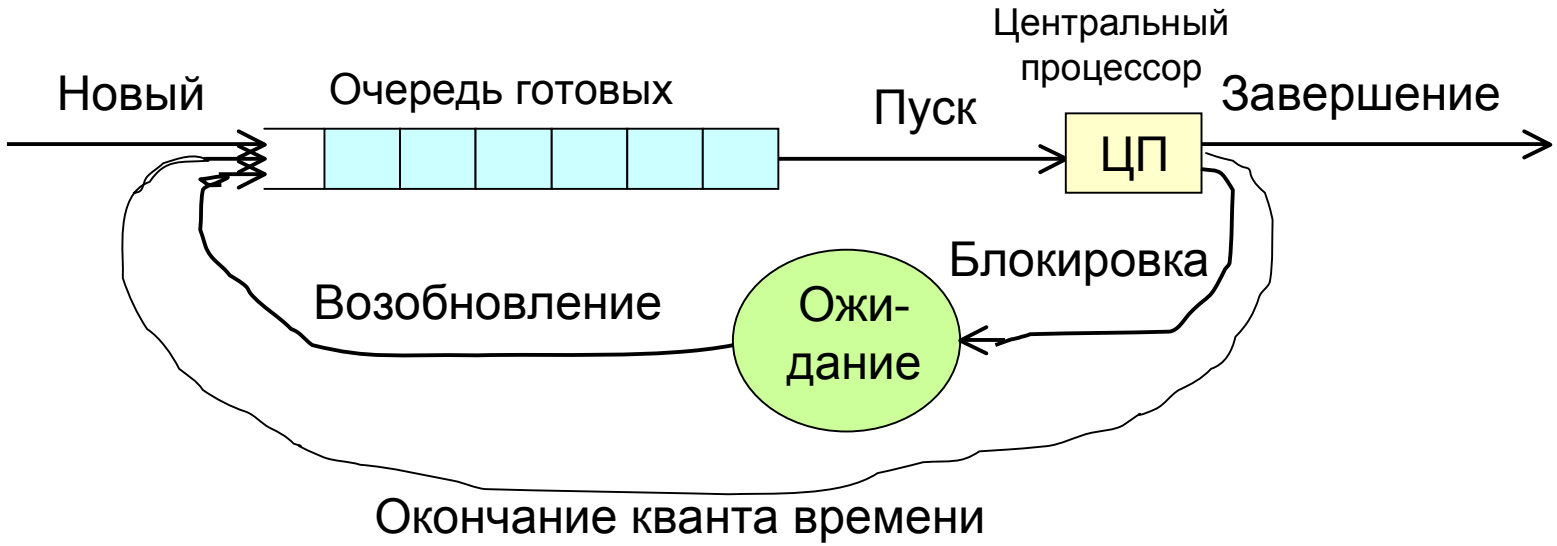
Достоинство: простота алгоритма, высокая производительность

Недостаток: большая дисперсия времени обслуживания:

время ожидания в очереди готовых может быть велико, если короткие процессы оказываются в очереди за длинными (несправедливость!); поэтому же мало совмещение вв/вы и вычислений в ЦП

Алгоритм используется в основном в системах пакетной обработки

Круговой алгоритм (RR, Round Robin)



- Достоинства по сравнению с FCFS:
  - справедливость: каждый процесс имеет равную долю времени ЦП, и время обслуживания пропорционально «длине» процесса
  - лучше совмещение во времени вычислений и вв-вы
- Недостаток: велико среднее время ожидания в очереди $\longrightarrow$ общая производительность ниже, чем при FCFS (если не учитывать вв-вы)

Выбор длительности кванта: слишком велик $\rightarrow$ FCFS, слишком мал $\rightarrow$ велики накладные расходы на переключение контекстов. Баланс: время переключения контекста процесса < 1% от кванта, т.е. типичная длительность кванта 10 - 100 мс

Приоритет – самому короткому процессу (SJF – Shortest Job First)

- Запускается тот процесс из очереди, у которого наименьшее требуемое время процессора до следующего запроса вв/вы или до завершения - CPU burst (назовем его CPU порцией)
  - при невытесняющем планировании процессор предоставляется избранному процессу на все необходимое ему время
  - при вытесняющем планировании алгоритм называется SRTF (Shortest Remaining Time First): если CPU порция нового процесса в очереди готовых меньше, чем остаток CPU порции у текущего, то последний вытесняется новым
- Достоинство: алгоритм доказательно оптимален в смысле минимума суммарного (а значит, и среднего) времени ожидания (Вопрос 8)
- Недостатки:
  - Опасность *голодания* длинных процессов
  - Точный SJF недостижим из-за невозможности *точного* предсказания времени до следующего запроса вв/вы (Вопрос 9)

Приоритетное планирование

- Каждому процессу назначается приоритет, и ЦП отдается готовому процессу с наивысшим приоритетом
- Обычно процессы группируются в *классы* с одинаковым приоритетом, а внутри класса выбираются в порядке FCFS или RR.
- Таким образом, очередь готовых процессов распадается на несколько очередей: MLQ (MultiLevel Queues)
- Более высокие приоритеты назначаются классам более коротких процессов

- Достоинства: высокая реактивность для срочных процессов
- Недостаток: опасность голодания низкоприоритетных процессов, которые не получают времени процессора, если есть хотя бы один готовый процесс большего приоритета

Статические приоритеты

- В ранних ОС *фиксированные* приоритеты назначались *статически* разным классам, так что процессы постоянно закреплены за очередями к процессору

  Например, такие классы процессов в порядке убывания приоритетов:

	Пример переменной длины кванта, сек
• Системные процессы	0,1
• Интерактивное редактирование	0,2
• Интерактивные вычисления	0,4
• Оперативная пакетная обработка	0,8
• Фоновая пакетная обработка	1,6 или FCFS

Удлинение кванта для более длинных процессов уменьшает издержки на переключение процессов

(Вопрос 10)

Динамические приоритеты

- В современных ОС приоритеты могут изменяться динамически. В частности, SJF - это алгоритм, где динамический приоритет процесса тем меньше, чем больше (ожидаемая) нужная ему очередная порция времени ЦП
- MLFB (Multilevel Feedback Queues) - многоуровневые очереди с динамическими приоритетами на основе обратной связи - приближение к SJF путем *адаптивного* предсказания будущего поведения процесса на основе его прошлого

Схема диспетчеризации MLFB:

- Несколько очередей с различными приоритетами
- Диспетчеризация RR на каждом уровне приоритетов; запускается процесс из очереди с наивысшим приоритетом
- Низкоприоритетный процесс запускается, только если нет готового процесса с более высоким приоритетом (NB: при этом опасность голодания!)
- Размер кванта времени увеличивается по мере уменьшения приоритета
- На самом нижнем уровне возможен алгоритм FCFS

Адаптация в MLFB

Один из возможных вариантов адаптивной настройки приоритетов процессов:

- Процесс стартует на высшем уровне приоритетов
- Если происходит прерывание по исчерпанию кванта времени, его приоритет понижается на один уровень
- Если происходит запрос вв/вы, приоритет повышается на один уровень, вплоть до высшего

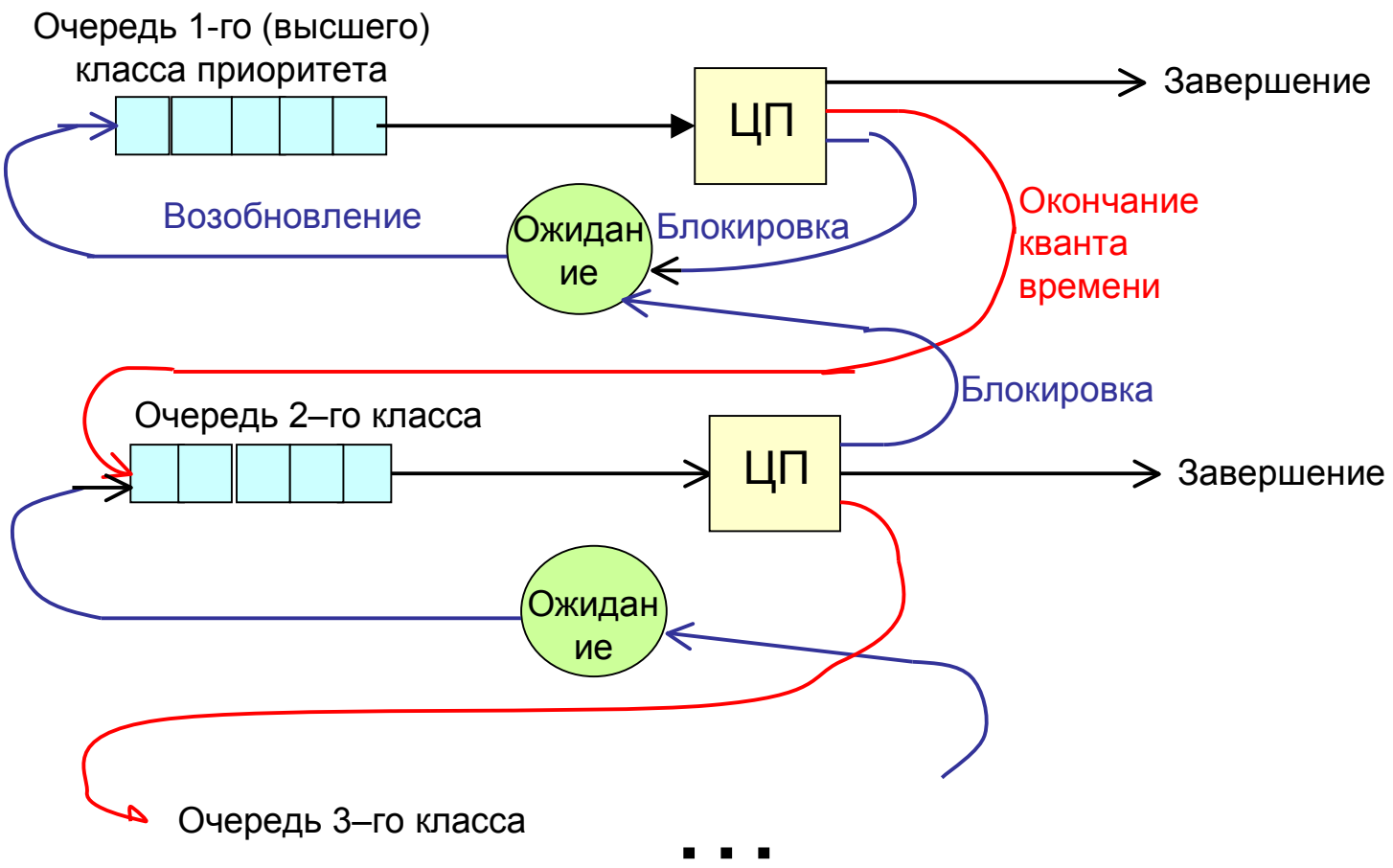
В результате длинные процессы "тонут", а короткие - "всплывают", причем отслеживается динамика потребностей процесса в течение его жизни

Автоматическое приближение к SJF !

Недостаток MLFB: несправедливое распределение времени ЦП →

голодание низкоприоритетных процессов (Вопрос 11)

Схема варианта MLFB



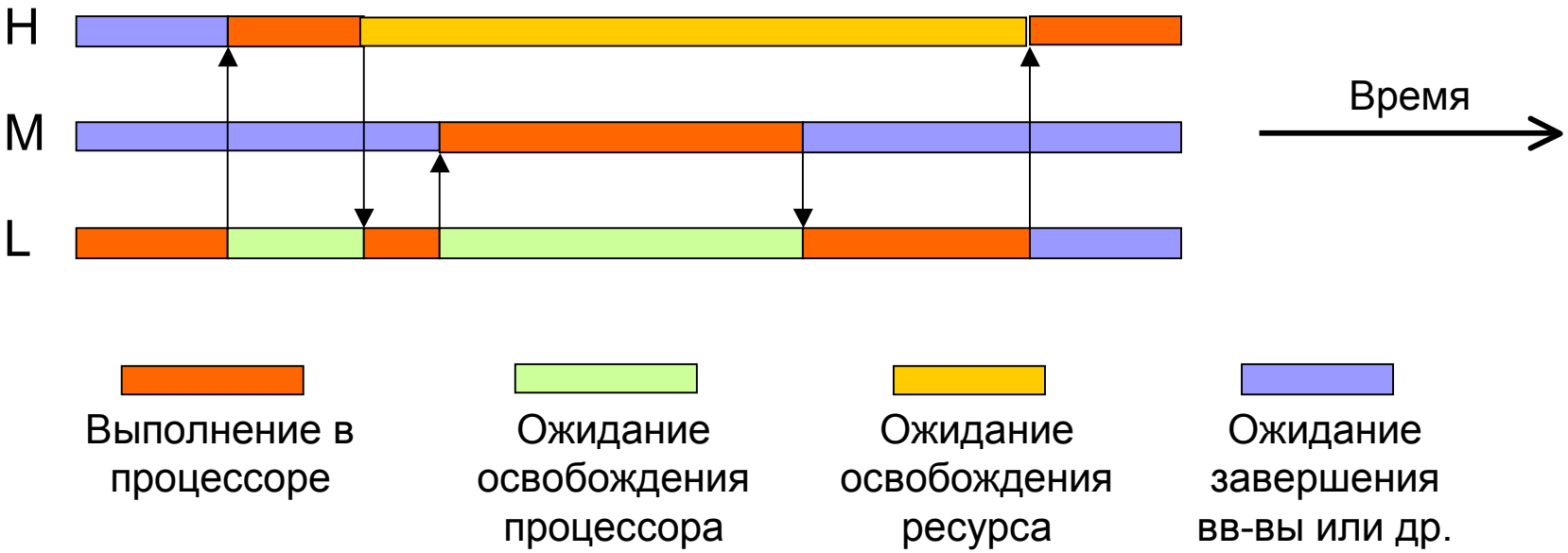
Лотерейное планирование (Lottery scheduling, LS)

- Процессы получают "лотерейные билеты" - тем больше билетов, чем короче процесс
- На каждом кванте времени диспетчер случайным образом выбирает выигравший билет
- В среднем каждый процесс получает время ЦП, пропорциональное количеству его билетов. Голодание предотвращается тем, что у любого длинного процесса есть по крайней мере один билет.
- LS в комбинации с адаптацией уровней приоритетов - это приближение к SJF, более справедливое, чем MLFB, но менее предсказуемое (из-за случайностей)

Инверсия приоритетов

- нежелательное явление изменения порядка приоритетов на обратный:

- Процесс высшего приоритета H не получает времени процессора (не переходит в состояние готовности), т.к. ждет освобождения неразделяемого ресурса (напр., принтера)
- Этот ресурс захвачен менее приоритетным процессом L
- L не получает времени процессора, т.к. вытесняется процессом промежуточного приоритета M



(Вопрос 12)

Выводы

- Процессы порождаются динамически другими процессами, образуя генеалогическое дерево
- Потоки – это «легковесные» процессы, совместно использующие общее адресное пространство
- Потоки одного процесса выполняют одну общую задачу – прикладную или системную
- Потоки пользователей – это способ реализовать собственный алгоритм диспетчеризации
- Полноценное мультипрограммирование обеспечивает только вытесняющая диспетчеризация
- В настоящее время наиболее распространены варианты алгоритма диспетчеризации многоуровневых очередей с динамическими приоритетами, где приближение к SJF - с помощью обратной связи или другим способом
- Системам пакетной обработки хорошо подходит алгоритм FCFS
- Системы с приоритетами могут вести себя парадоксально; пример – ситуация инверсии приоритетов

Вопросы для самоконтроля (1)

1. В многопроцессорной системе производительность возрастает всегда при назначении потоков разным процессам. А при каких условиях возрастает производительность многопоточного процесса в однопроцессорной системе?
2. При запуске MS Word порождаются три потока; далее их число может возрастать. Предложите, какие части Word разумно выполнять как отдельные потоки.
3. Сравните достоинства и недостатки этого подхода с двумя альтернативными: а) все клиенты обслуживаются одним процессом, б) для каждого клиента порождается отдельный процесс (для примера возьмите веб-сервер).
4. Какими должны быть потоки сервера для каждого клиента – потоками ядра или пользователя?
5. В Windows 3.1 (1991 г.) была невытесняющая диспетчеризация. В чем заключаются недостатки такого "неполноценного" мультипрограммирования ?
6. Нарисуйте диаграмму переключения состояний прикладного процесса в MS-DOS.
7. Для каких типов вычислительных систем какие критерии более приоритетны? (Рассмотрите три типа систем: пакетной обработки, интерактивные системы и реального времени.)
8. Докажите (путем простого рассуждения) указанную оптимальность алгоритма SJF.
9. Предложите *простой* способ *приближенного* предсказания этого времени на основе предыстории активности процесса.
10. В чем недостатки статических приоритетов ?

Вопросы для самоконтроля (2)

11. Как с помощью динамических приоритетов можно предотвращать голодание низкоприоритетных процессов?
12. Предложите способ предотвращения инверсии приоритетов
13. Заполните таблицы в предположении нулевого времени переключения контекста и отсутствия ввода-вывода. Процессы находятся в очереди готовых в порядке возрастания номеров и стартуют одновременно. Длительность кванта - 1 сек.

Процесс №	Длина, сек.	Время ожидания		Время обслуживания	
		FCFS	RR	FCFS	RR
1	10				
2	10				
3	10				
4	10				
Среднее:					

Вопросы для самоконтроля (3)

Процесс №	Длина, сек.	Время ожидания			Время обслуживания		
		FCFS	RR	SJF	FCFS	RR	SJF
1	40						
2	30						
3	20						
4	10						
Среднее:							