# 3. Планирование процессов

# 3.1. Управление исполнением процессов

# Исполнение процессов в однопрограммной ОС

- В однопрограммной операционной системе одновременно может выполняться только один процесс, которому доступны все ресурсы компьютера.

# Причины и недостатки блокировки процесса в однопрограммной ОС

- Причиной *блокировки* процесса в однопрограммной ОС является ожидание этим процессом завершения операций ввода-вывода.
- Недостатком однопрограммных операционных систем является их *низкая производительность*, так как процессор простаивает, если процесс блокирован.

# Исполнение процессов в мультипрограммных ОС

- В мультипрограммных операционных системах одновременно могут существовать несколько процессов, что повышает производительность компьютера.
- Однако в этом случае требуется некоторый механизм обслуживания этих процессов, который распределяет системные ресурсы между **конкурирующими** (параллельными) **процессами**.

# Разделение процессорного времени

- Основным ресурсом, который нужно разделять между конкурирующими процессами является **процессор**.
- Так как процессор исполняет потоки, то фактически нужен механизм для разделения процессорного времени между конкурирующими потоками.

# Допущения для простоты изложения

- Для упрощения дальнейшего изложения будем предполагать, что процесс содержит только ***один*** поток.
- В этом случае обслуживание параллельных потоков можно  отождествить с обслуживанием параллельных процессов.
- Также для простоты будем считать, что компьютер имеет только ***один*** процессор.

# Принципы обслуживания процессов в мультипрограммных операционных системах

1. Время работы процессора делится на ***кванты*** (интервалы), которые выделяются процессам для работы.
2. По истечении кванта времени исполнение процесса ***прерывается*** и процессор *назначается* другому процессу.

# Переключение контекста процесса

- Распределением квантов процессорного времени между процессами занимается специальная программа ОС, которая называется ***менеджер процессов*** или ***менеджер потоков***.
- Когда менеджер процессов переключает процессор на исполнение другого процесса, он должен выполнить следующие действия:
  1. Сохранить контекст прерываемого процесса,
  2. Восстановить контекст запускаемого процесса на момент его прерывания,
  3. Передать управление запускаемому процессу.

# Контекст процесса и состояние регистров микропроцессора

- Контекст процесса это содержимое памяти, с которой работает процесс.
- Адреса оперативной памяти, состояния микропроцессора и команды микропроцессора хранятся в регистрах микропроцессора.
- Поэтому в каждый момент времени работы процесса, его контекст полностью определяется содержимым регистров микропроцессора в этот момент времени.

- Отсюда следует:
  - для сохранения контекста процесса необходимо *сохранить* содержимое регистров микропроцессора на момент прерывания процесса,
  - при восстановлении контекста процесса необходимо *восстановить* содержимое этих регистров.

# 3.2. Планирование потоков

- Под ***планированием*** потоков понимается алгоритм, используемый для планирования (установки) приоритета потока.
- Менеджер потоков может изменять приоритет прерванного потока.

# Очереди потоков

- Прерванные потоки становятся в ***очередь*** к процессору на обслуживание.
- В общем случае можно сказать, что очередь содержит ***заявки*** на обслуживание некотором устройством.
- Алгоритмы диспетчеризации очередей изучаются математической дисциплиной, которая называется *теория массового обслуживания*.

# Критерии оптимизации планирования процессов

- Время загрузки микропроцессора работой должно быть ***максимальным.***
- Пропускная способность системы должна быть ***максимальной.***
- Время нахождения процесса в системе должно быть ***минимальным.***
- Время ожидания потока в очереди должно быть ***минимальным.***
- Время реакции системы на обслуживание заявки должно быть ***минимальным***.

# Оптимальный интервал обслуживания процессов

- Для каждой системы должен быть выбран ***оптимальный интервал*** *процессорного времени* для обслуживания процессов, который снижает затраты на переключение контекстов процессов.
- Если переключение между процессами происходит очень часто, то много процессорного времени тратится на перестановку контекстов процессов.

# Зависимость времени исполнения процесса от его трудоемкости

- В общем случае разделение времени работы процессора между параллельными процессами позволяет:
  - быстрее выполнять процессы, которые требуют *немного* времени на своё исполнение;
  - замедляет исполнение *трудоемких* процессов.

# 3.3. Алгоритмы планирования непрерываемых процессов

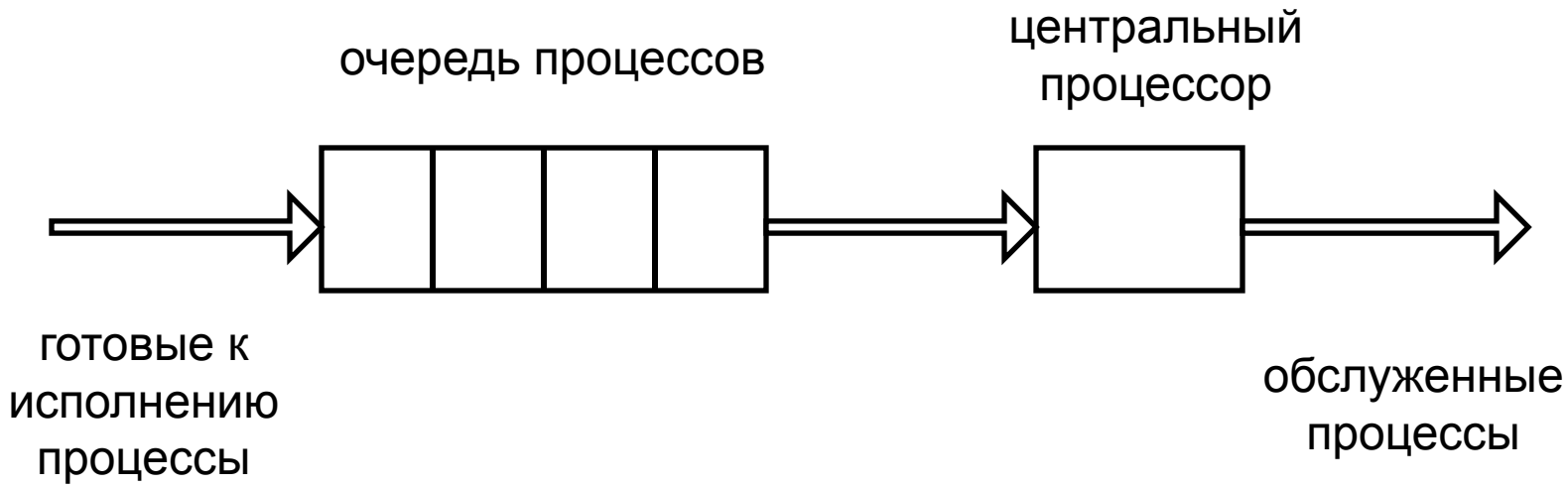
# Непрерываемый процесс

- Предположим, что работа процесса ***не прерывается*** во время его исполнения.
- В этом случае для обслуживания процессов применяются следующие стратегии.

# Стратегия FCFS

- Простейшая стратегия планирования непрерываемых процессов заключается в следующем:
  - процессы, готовые к исполнению, становятся в **очередь** на исполнение;
  - первым обслуживается процесс, который **первый** поступил на обработку (стал в очередь);
  - по завершении исполнения процесса из очереди выбирается процесс, который находится в очереди **дольше** других процессов.

- Такое обслуживание процессов называется **FCFS** (first come – first served) — первым пришел - первым обслужен.



# Стратегия SPN

- Чтобы отложить обработку длинных процессов, при выборе процессов из очереди применяется стратегия **SPN** (shortest process next) :
  - для исполнения выбирается процесс с **наименьшим** ожидаемым временем исполнения.
- При использовании этой стратегии нужно решить проблему оценки времени работы процесса.

# Оценка времени работы процесса

- В простейшем случае время работы процесса оценивается по следующей формуле:
- $$S_{n+1} = \frac{1}{n} T_n + \frac{n-1}{n} S_n$$, где
  - $S_1$ – начальное предсказанное время исполнения процесса;
  - $T_n$ – действительное время работы процесса при его n-ом запуске;
  - $S_{n+1}$ – предсказанное время работы процесса при его (n+1) – запуске.

# 3.4. Алгоритмы планирования прерываемых процессов

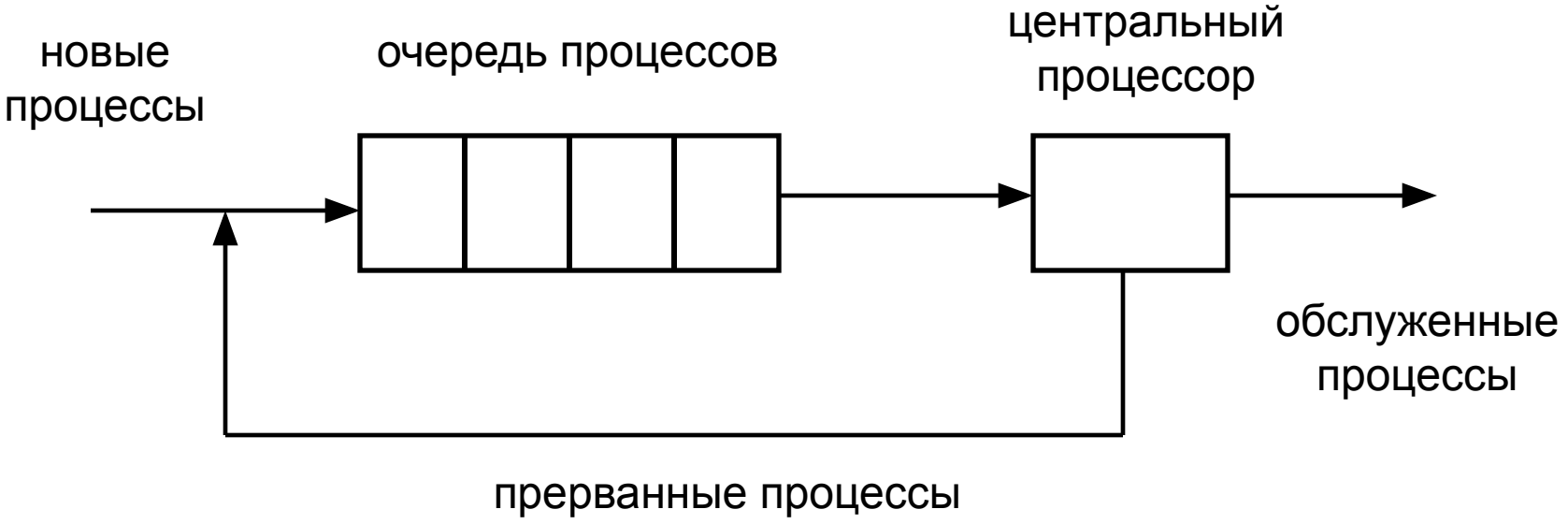
# Прерываемый процесс

- Предположим, что исполнение процесса может быть *прервано* по истечении кванта времени процессора, выделенного этому процессу (потоку этого процесса).
- Кроме того, предположим, что все процессы имеют *одинаковый приоритет*.
- В этом случае для обслуживания процессов применяются следующие стратегии.

# Стратегия RR

- Простейшая дисциплина обслуживания прерываемых процессов заключается в следующем:
  - все процессы выстраиваются в **одну очередь** на обслуживание к процессору;
  - процессор обслуживает процессы в порядке **FIFO** (first in – first out — первым пришел - первым вышел);
  - прерванные процессы становятся в **конец очереди**.

- Такая дисциплина обслуживания процессов называется *циклическим* (круговым) *планированием* (RR – round robin).

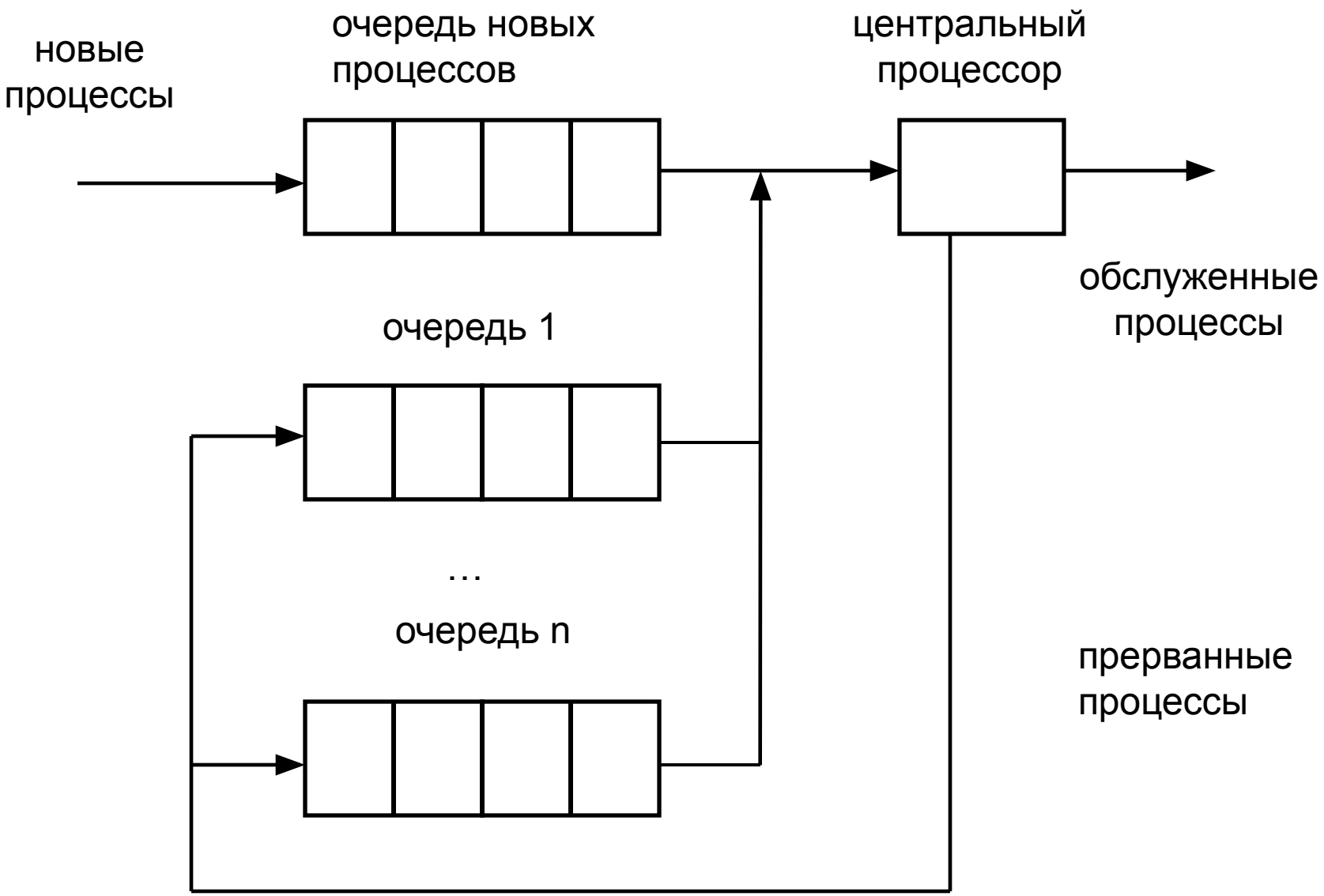


# Стратегия SRT

- Расширением циклического планирования является стратегия наименьшего остающегося времени **SRT**
  – shortest remaining time:
  – менеджер (планировщик) выбирает из очереди процесс с *наименьшим* ожидаемым временем до завершения его работы.

# Обслуживание процессов с приоритетами

- Если процессам назначаются ***приоритеты***, то для управления процессами используются более сложные дисциплины обслуживания с несколькими очередями.
- В этом случае каждая очередь включает процессы, которые имеют ***одинаковый*** приоритет.
- Первыми обслуживаются процессы, которые имеют ***наивысший приоритет***.


новые процессы
очередь новых процессов
центральный процессор
обслуженные процессы
очередь 1
…
очередь n
прерванные процессы

# 3.5. Обслуживание потоков в Windows

- Операционная система Windows обслуживает *потоки*.
- Процессорное время выделяется потокам в соответствии с их *приоритетами*.
- Первым обслуживается поток с наивысшим приоритетом.

# Относительность приоритета потока

- Приоритеты потоков в Windows определяются *относительно приоритета процесса*, в контексте которого они исполняются.

# Диапазон абсолютных значений приоритета потока

- Абсолютное значение приоритета потока изменяется в диапазоне:
  - от 0 (низший приоритет)
  - до 31 (высший приоритет).

# Установка приоритета процесса в Windows

- Приоритет процессов устанавливается при их создании функцией *CreateProcess*, используя параметр *dwCreationFlags* этой функции.
- Для установки приоритета процесса, в этом параметре нужно установить один из следующих флагов:
  - IDLE_PRIORITY_CLASS
  - BELOW_NORMAL_PRIORITY_CLASS
  - NORMAL_PRIORITY_CLASS
  - ABOVE_NORMAL_PRIORITY_CLASS
  - HIGH_PRIORITY_CLASS
  - REAL_TIME_PRIORITY_CLASS

# Типы процессов в Windows

- Предполагается, что операционная система Windows различает четыре типа процессов в соответствии с их приоритетами:
  - фоновые процессы (IDLE_PRIORITY_CLASS);
  - процессы с нормальным приоритетом (NORMAL_PRIORITY_CLASS);
  - процессы с высоким приоритетом (HIGH_PRIORITY_CLASS);
  - процессы реального времени (REAL_TIME_PRIORITY_CLASS).

# Фоновые процессы

- Фоновые процессы выполняют свою работу, когда нет активных пользовательских процессов.
- Обычно, эти процессы следят за состоянием системы.
- Приоритет таких процессов устанавливается флагом

  IDLE_PRIORITY_CLASS.

# Процессы с нормальным приоритетом

- Процессы с нормальным приоритетом это обычные пользовательские процессы.
- Приоритет таких процессов устанавливается флагом

  NORMAL_PRIORITY_CLASS.
- Этот приоритет назначается пользовательским процессам по умолчанию.
- Приоритет обычных пользовательских процессов может также устанавливаться флагами

  BELOW_NORMAL_PRIORITY_CLASS
- или

  ABOVE_NORMAL_PRIORITY_CLASS,
- которые соответственно немного понижают или повышают приоритет пользовательского процесса.

# Процессы с высоким приоритетом

- Процессы с высоким приоритетом это, как правило, системы, работающие на платформе Windows.
- Например, система управления базами данных.
- Приоритет таких процессов устанавливается флагом

HIGH_PRIORITY_CLASS.

# Процессы реального времени

- Процессы реального времени – это такие процессы, работа которых происходит в масштабе реального времени и связана с реакцией на внешние события.
- Эти процессы должны работать непосредственно с аппаратурой компьютера.
- Приоритет таких процессов устанавливается флагом

REAL_TIME_PRIORITY_CLASS.

# Функции для управления приоритетами процессов

- В Windows для управления приоритетами процессов предназначены следующие функции:
  - *SetPriorityClass* – позволяет изменить приоритет процесса;
  - *GetPriorityClass* – позволяет узнать приоритет процесса.

# 3.6. Управление приоритетами потоков в Windows

# Базовый приоритет потока

- Приоритет потока, который учитывается системой при выделении потокам процессорного времени, называется ***базовым*** (base) или ***основным приоритетом потока***.
- Всего существует 32 базовых приоритета потока, значение которых изменяются от 0 до 31.
- Базовый приоритет потока определяется как ***сумма*** *приоритета процесса*, в контексте которого исполняется поток, и *уровня приоритета потока*.

# Диспетчеризация потоков в Windows

- Для каждого базового приоритета существует очередь потоков.
- При диспетчеризации потоков интервал процессорного времени выделяется потоку, который стоит первым в очереди с наивысшим базовым приоритетом.

# Значения уровня приоритета потока

- Уровень приоритета потока может принимать одно из следующих значений, которые мы разобьем на две группы:

  1.
  - THREAD_PRIORITY_LOWEST
  - THREAD_PRIORITY_BELOW_NORMAL
  - THREAD_PRIORITY_NORMAL
  - THREAD_PRIORITY_ABOVE_NORMAL
  - THREAD_PRIORITY_HIGHEST

  2.
  - THREAD_PRIORITY_IDLE
  - THREAD_PRIORITY_TIME_CRITICAL

# Уровни приоритета потока

- Значения уровня приоритета потока из первой группы в сумме с приоритетом процесса, в контексте которого этот поток выполняется,
  - уменьшают,
  - или оставляют неизменным,
  - или увеличивают
- значение базового приоритета потока соответственно на –2, -1, 0, 1, 2.

# Уровни приоритета потока

- Уровень приоритета потока

  THREAD_PRIORITY_IDLE

- устанавливает базовый приоритет потока равным:
  - 16, если приоритет процесса, в контексте которого выполняется поток, равен REAL_TIME_PRIORITY_CLASS ,
  - 1 – в остальных случаях.

# Уровни приоритета потока

- Уровень приоритета потока

  THREAD_PRIORITY_TIME_CRITICAL
- устанавливает базовый приоритет потока равным:
  - 31, если приоритет процесса, в контексте которого выполняется поток, равен

    REAL_TIME_PRIORITY_CLASS,
  - 15 – в остальных случаях.

# Установка приоритета потока при его создании

- При создании потока его базовый приоритет устанавливается как сумма приоритета процесса, в контексте которого этот поток выполняется, и уровня приоритета потока

  THREAD_PRIORITY_NORMAL.

# Функции для управления уровнем приоритета потока

- Для управления уровнем приоритета потока в Windows используются следующие функции:
  - *SetThreadPriority* – изменяет уровень приоритета потока;
  - *GetThreadPriority* – позволяет определить уровень приоритета потока.

# 3.7. Динамическое изменение приоритетов потоков в Windows

# Допустимый диапазон, причины и величина повышения и понижения приоритета

- Базовый приоритет потока может динамически изменяться системой, если этот приоритет находится в пределах между уровнями 0 и 15.
- Система повышает базовый приоритет потока на 2 в двух случаях:
  - при получении потоком сообщения;
  - при переходе потока в состояние готовности.
- В процессе выполнения базовый приоритет такого потока понижается на 1, с каждым отработанным квантом времени, но никогда не опускается ниже исходного базового приоритета.

# Функции для динамического управления приоритетом потока

- Для динамического управления приоритетами потоков в Windows предназначены следующие функции:
  - *SetProcessPriorityBoost* – позволяет отменить или установить режим динамического изменения базового приоритета всех потоков процесса;
  - *GetProcessPriorityBoost* – позволяет узнать, разрешен ли режим динамического изменения базовых приоритетов потоков процесса;

- *SetThreadPriorityBoost* – позволяет отменить или установить режим динамического изменения базового приоритета только одного потока;
- *GetThreadPriorityBoost* – позволяет узнать, разрешен ли режим динамического изменения базового приоритета потока.

# 3.8. Задачи на обслуживание непрерываемых потоков

# Задача 1.

- Операционная система обслуживает процессы по алгоритму FCFS (First Come – First Served).
- В операционную систему поступают на выполнение процессы, время поступления и время исполнения которых приведены в следующей таблице.

| Номер процесса | Время поступления в систему | Время исполнения |
|---|---|---|
| 1 | 0 | 5 |
| 2 | 2 | 4 |
| 3 | 3 | 6 |
| 4 | 5 | 1 |
| 5 | 7 | 3 |

- Требуется вычислить:
  - среднее время нахождения процесса в системе;
  - среднее время ожидания процесса в очереди на исполнение.
- Строим таблицу:
  - строки – номера процессов;
  - столбцы – моменты времени.
- Обозначения:
  - Г – процесс готов к исполнению;
  - И – процесс исполняется.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | И | И | И | И | И | | | | | | | | | | | | | | |
| 2 | | | Г | Г | Г | И | И | И | И | | | | | | | | | | |
| 3 | | | | Г | Г | Г | Г | Г | Г | И | И | И | И | И | И | | | | |
| 4 | | | | | | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И | | | |
| 5 | | | | | | | | Г | Г | Г | Г | Г | Г | Г | Г | Г | И | И | И |

- Вычисляем среднее время нахождения процесса в системе:

$$Te = (5 + 7 + 12 + 11 + 12) / 5 =$$

$$= 47 / 5 = 9.4$$

- Вычисляем среднее время ожидания процесса в очереди на исполнение:

$$Tw = (0 + 3 + 6 + 10 + 9) / 5 =$$

$$= 28 / 5 = 5.6$$

# Задача 2.

- Операционная система обслуживает процессы по алгоритму SPN (Shortest Process Next).
- В операционную систему поступают на выполнение процессы, время поступления и время исполнения которых приведены в Задаче 1.

Требуется вычислить:

- среднее время нахождения процесса в системе;
- среднее время ожидания процесса в очереди на исполнение.

- Строим таблицу исполнения процессов, как это было показано в Задаче 1.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | И | И | И | И | И | | | | | | | | | | | | | | |
| 2 | | | Г | Г | Г | И | И | И | И | | | | | | | | | | |
| 3 | | | | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И | И | И | И | И | И |
| 4 | | | | | | Г | Г | Г | Г | И | | | | | | | | | |
| 5 | | | | | | | | Г | Г | Г | И | И | И | | | | | | |

- Вычисляем среднее время нахождения процесса в системе:

$$Te = (5 + 7 + 16 + 5 + 6) / 5 =$$

$$= 39 / 5 = 7.8$$

- Вычисляем среднее время ожидания процесса в очереди на исполнение:

$$Tw = (0 + 3 + 10 + 4 + 3) / 5 =$$

$$= 20 / 5 = 4$$

# 3.9. Задачи на обслуживание прерываемых процессов

# Задача 3.

- Операционная система обслуживает процессы по алгоритму RR (Round Robin).
- В операционную систему поступают на выполнение процессы, время поступления и время исполнения которых приведены в Задаче 1.

- Предполагается, что
  - переключение контекстов процессов выполняется мгновенно;
  - каждому процессу для исполнения выделяется два кванта времени;
  - прерванные процессы становятся в очередь раньше новых процессов.

- Требуется вычислить:
  - среднее время нахождения процесса в системе;
  - среднее время ожидания процесса в очереди на исполнение.
- Строим таблицу:
  - строки – номера процессов;
  - столбцы – моменты времени.
- Обозначения:
  - Г – процесс готов к исполнению;
  - И – процесс исполняется.

Время исполнения процессов - 1 (5), 2(4), 3(6), 4(1), 5(3)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | И | И | И | И | Г | Г | Г | Г | И | | | | | | | | | | |
| 2 | | | Г | Г | И | И | Г | Г | Г | Г | И | И | | | | | | | |
| 3 | | | | Г | Г | Г | И | И | Г | Г | Г | Г | Г | Г | И | И | Г | И | И |
| 4 | | | | | | Г | Г | Г | Г | И | | | | | | | | | |
| 5 | | | | | | | | Г | Г | Г | Г | Г | И | И | Г | Г | И | | |

- Вычисляем среднее время нахождения процесса в системе:

$$Te = (9 + 10 + 16 + 5 + 10) / 5 =$$

$$= 50 / 5 = 10$$

- Вычисляем среднее время ожидания процесса в очереди на исполнение:

$$Tw = (4 + 6 + 10 + 4 + 7) / 5 =$$

$$= 31 / 5 = 6.2$$

# Пока не готово

Время исполнения процессов - 1 (5), 2(4), 3(6), 4(1), 5(3)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | И | И | И | Г | И | Г | Г | И | | | | | | | | | | | |
| 2 | | | Г | И | Г | Г | И | Г | Г | Г | И | Г | Г | И | | | | | |
| 3 | | | | Г | Г | И | Г | Г | Г | И | Г | Г | И | Г | Г | И | Г | И | И |
| 4 | | | | | | Г | Г | Г | И | | | | | | | | | | |
| 5 | | | | | | | | Г | Г | Г | Г | И | Г | Г | И | Г | И | | |

- Вычисляем среднее время нахождения процесса в системе:

$$Te = (8 + 12 + 16 + 4 + 10) / 5 =$$

$$= 50 / 5 = 10$$

- Вычисляем среднее время ожидания процесса в очереди на исполнение:

$$Tw = (3 + 8 + 10 + 3 + 7) / 5 =$$

$$= 31 / 5 = 6.2$$

| Номер процесса | Время поступления в систему | Время исполнения |
|---|---|---|
| 1 | 0 | 5 |
| 2 | 2 | 4 |
| 3 | 3 | 6 |
| 4 | 5 | 1 |
| 5 | 7 | 3 |

| Номер процесса | Время поступления в систему | 1. Время исполнения | 2. Время ожидания | 3. Время исполнения |
|---|---|---|---|---|
| 1 | 0 | 3 | 3 | 1 |
| 2 | 2 | 2 | 4 | 2 |
| 3 | 3 | 2 | 1 | 1 |
| 4 | 4 | 1 | 2 | 3 |
| 5 | 7 | 3 | 2 | 4 |

- Операционная система обслуживает процессы по алгоритму RR (Round Robin).
- Квант времени, выделяемый процессам на обслуживание, равен 1.
- Предположения относительно приоритетов прерываний:
  - новый процесс не может поступить в момент прерывания исполняемого процесса;
  - прерывание кванта времени исполнения процесса имеет более высокий приоритет, чем прерывание на завершение ожидания.

- Требуется вычислить:
  - среднее время нахождения процесса в системе;
  - среднее время ожидания процесса в очереди на исполнение.
- Строим таблицу:
  - строки – номера процессов;
  - столбцы – моменты времени.
- Обозначения:
  - Г – процесс готов к исполнению;
  - И – процесс исполняется;
  - О – процесс находится в состоянии ожидания.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | И | И | И | О | О | О | Г | И | | | | | | | | | | | |
| 2 | | | Г | И | Г | И | О | О | О | О | | | | | | | | | |
| 3 | | | | Г | И | Г | И | | | | | | | | | | | | |
| 4 | | | | | Г | И | Г | | | | | | | | | | | | |
| 5 | | | | | | | | Г | | | | | | | | | | | |

- Вычисляем среднее время нахождения процесса в системе:

$$Te = (10 + 13 + 16 + 3 + 10) / 5 =$$

$$= 52 / 5 = 10.4$$

- Вычисляем среднее время ожидания процесса в очереди на исполнение:

$$Tw = (0 + 3 + 10 + 4 + 3) / 5 =$$

$$= 20 / 5 = 4$$