#### Практическая работа №7

#### Приоритетное планирование

*Алгоритмы SJF* и *гарантированного планирования* представляют собой частные случаи *приоритетного планирования*. При ***приоритетном планировании*** каждому процессу присваивается определенное числовое значение – *приоритет*, в соответствии с которым ему выделяется процессор. Процессы с одинаковыми *приоритетами* планируются в порядке *FCFS*. Для *алгоритма SJF* в качестве такого *приоритета выступает* оценка продолжительности следующего *CPU burst*. Чем меньше значение этой оценки, тем более высокий *приоритет* имеет процесс. Для алгоритма *гарантированного планирования* *приоритетом* служит вычисленный коэффициент справедливости. Чем он меньше, тем больше у процесса ***приоритет*** .

Алгоритмы назначения *приоритетов* процессов могут опираться как на *внутренние параметры*, связанные с происходящим внутри вычислительной системы, так и на внешние по отношению к ней. К внутренним параметрам относятся различные количественные и качественные характеристики процесса такие как: ограничения по времени использования процессора, требования к размеру памяти, число открытых файлов и используемых устройств ввода-вывода, отношение средних продолжительностей *I/O burst* к *CPU burst* и т. д. *Алгоритмы SJF* и гарантированного планирования используют *внутренние параметры*. В качестве *внешних параметров* могут выступать важность процесса для достижения каких-либо целей, стоимость оплаченного процессорного времени и другие политические факторы. Высокий внешний *приоритет* может быть присвоен задаче лектора или того, кто заплатил $100 за работу в течение одного часа.

*Планирование* с использованием *приоритетов* может быть как *вытесняющим*, так и *невытесняющим*. При *вытесняющем планировании* процесс с более высоким *приоритетом*, появившийся в очереди готовых процессов, вытесняет исполняющийся процесс с более низким *приоритетом*. В случае *невытесняющего планирования* он просто становится в начало очереди готовых процессов. Давайте рассмотрим примеры использования различных режимов *приоритетного* *планирования*.

Пусть в очередь процессов, находящихся в состоянии готовность, поступают те же процессы, что и в примере для *вытесняющего* *алгоритма SJF*, только им дополнительно еще присвоены *приоритеты* (см. табл. 1). В вычислительных системах не существует определенного соглашения, какое значение *приоритета* – 1 или 4 считать более *приоритетным*. Во избежание путаницы, во всех наших примерах мы будем предполагать, что большее значение соответствует меньшему *приоритету*, т. е. наиболее *приоритетным* в нашем примере является процесс p3, а наименее *приоритетным* – процесс p0.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 1 | | | |
| **Процесс** | **Время появления в очереди** | **Продолжительность очередного** ***CPU burst*** | ***Приоритет*** |
| **p0** | 0 | 6 | 4 |
| **p1** | 2 | 2 | 3 |
| **p2** | 6 | 7 | 2 |
| **p3** | 0 | 5 | 1 |

Как будут вести себя процессы при использовании *невытесняющего* *приоритетного планирования*? Первым для выполнения в момент времени t = 0 выбирается процесс p3, как обладающий наивысшим *приоритетом*. После его завершения в момент времени t = 5 в очереди процессов, готовых к исполнению, окажутся два процесса p0 и p1. Больший *приоритет* из них у процесса p1, он и начнет выполняться (см. табл. 2).

Затем в момент времени t = 8 для исполнения будет избран процесс p2, и лишь потом – процесс p0.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 2. | | | | | | | | | | | | | | | | | | | | | |
| **Время** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| **p0** | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И | И | И | И | И | И |
| **p1** |  |  | Г | Г | Г | И | И |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **p2** |  |  |  |  |  |  | Г | И | И | И | И | И | И | И |  |  |  |  |  |  |
| **p3** | И | И | И | И | И |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Иным будет предоставление процессора процессам в случае *вытесняющего* *приоритетного планирования* (см. табл. 3). Первым, как и в предыдущем случае, начнет исполняться процесс p3, а по его окончании – процесс p1. Однако в момент времени t = 6 он будет вытеснен процессом p2 и продолжит свое выполнение только в момент времени t = 13. Последним, как и раньше, будет исполняться процесс p0.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 3 | | | | | | | | | | | | | | | | | | | | | |
| **Время** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| **p0** | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И | И | И | И | И | И |
| **p1** |  |  | Г | Г | Г | И | Г | Г | Г | Г | Г | Г | Г | И |  |  |  |  |  |  |
| **p2** |  |  |  |  |  |  | И | И | И | И | И | И | И |  |  |  |  |  |  |  |
| **p3** | И | И | И | И | И |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

В рассмотренном выше примере *приоритеты* процессов с течением времени не изменялись. Такие *приоритеты* принято называть статическими. Механизмы статической *приоритетности* легко реализовать, и они сопряжены с относительно небольшими издержками на выбор наиболее *приоритетного* процесса. Однако статические *приоритеты* не реагируют на изменения ситуации в вычислительной системе, которые могут сделать желательной корректировку порядка исполнения процессов. Более гибкими являются динамические *приоритеты процессов*, изменяющие свои значения по ходу исполнения процессов. Начальное значение динамического *приоритета*, присвоенное процессу, действует в течение лишь короткого периода времени, после чего ему назначается новое, более подходящее значение. Изменение динамического *приоритета процесса* является единственной операцией над процессами, которую мы до сих пор не рассмотрели. Как правило, изменение *приоритета процессов* проводится согласованно с совершением каких-либо других операций: при рождении нового процесса, при разблокировке или блокировании процесса, по истечении определенного *кванта времени* или по завершении процесса. Примерами алгоритмов с динамическими *приоритетами* являются *алгоритм SJF* и алгоритм *гарантированного планирования*. Схемы с динамической приоритетностью гораздо сложнее в реализации и связаны с большими издержками по сравнению со статическими схемами. Однако их использование предполагает, что эти издержки оправдываются улучшением работы системы.

Главная проблема *приоритетного планирования* заключается в том, что **при ненадлежащем выборе механизма назначения и изменения** ***приоритетов* низкоприоритетные процессы могут не запускаться неопределенно долгое время**.

Обычно случается одно из двух. Или они все же дожидаются своей очереди на исполнение. Или вычислительную систему приходится выключать, и они теряются (при остановке IBM 7094 в Массачусетском технологическом институте в 1973 году были найдены процессы, запущенные в 1967 году и ни разу с тех пор не исполнявшиеся).

**Решение этой проблемы может быть достигнуто с помощью увеличения со временем значения** ***приоритета процесса*, находящегося в состоянии готовность**.

Пусть изначально процессам присваиваются *приоритеты* от 128 до 255. Каждый раз по истечении определенного промежутка времени значения *приоритетов* готовых процессов уменьшаются на 1. Процессу, побывавшему в состоянии исполнение, присваивается первоначальное значение *приоритета*. Даже такая грубая схема гарантирует, что любому процессу в разумные сроки будет предоставлено право на исполнение.

**Задания**

Практическая работа содержит 3 задания и оценивается в 3 балла.

***Задание №1***

Используя алгоритм *невытесняющего* планирования с приоритетами на основе SJF алгоритма выполнить построение таблицы исполнения для следующих процессов (см. таблицу 4):

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 4 | | | |
| **Процесс** | **Время появления в очереди** | **Продолжительность очередного *CPU burst*** | ***Приоритет*** |
| **p0** | 0 | 6 | 1 |
| **p1** | 2 | 2 | 2 |
| **p2** | 6 | 7 | 2 |
| **p3** | 0 | 5 | 4 |

Рассчитать среднее время ожидания и среднее полное время выполнения.

Решение:

…

***Задание №2***

Используя алгоритм *вытесняющего* планирования с приоритетами на основе SJF алгоритма выполнить построение таблицы исполнения для следующих процессов (см. таблицу 5):

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 5 | | | |
| **Процесс** | **Время появления в очереди** | **Продолжительность очередного *CPU burst*** | ***Приоритет*** |
| **p0** | 0 | 9 | 1 |
| **p1** | 3 | 2 | 2 |
| **p2** | 5 | 8 | 2 |
| **p3** | 0 | 5 | 1 |

Рассчитать среднее время ожидания и среднее полное время выполнения.

Незабываем, что за основу принят алгоритм SJF.

***Задание №3***

Выполнить программную реализацию алгоритма SJF с приоритетами. Программное обеспечение должно:

1. Обеспечить появление процессов в очереди (15 процессов) со случайным временем появления, процесс должен иметь случайный приоритет и случайное время исполнения.
2. Реализовывать гарантийное выполнение процессов (динамическое изменение приоритета процесса).
3. Выполнять построение таблицы работы процессов;
4. Рассчитать среднее время ожидания и среднее полное время выполнения.