

Процессы



Процессы



Процессы



Процессы. Пары состояний процесса

- создание процесса – завершение процесса ;
- приостановка процесса (перевод из состояния исполнение в состояние готовность) – запуск процесса (перевод из состояния готовность в состояние исполнение);
- блокирование процесса (перевод из состояния исполнение в состояние ожидание) – разблокирование процесса (перевод из состояния ожидание в состояние готовность).

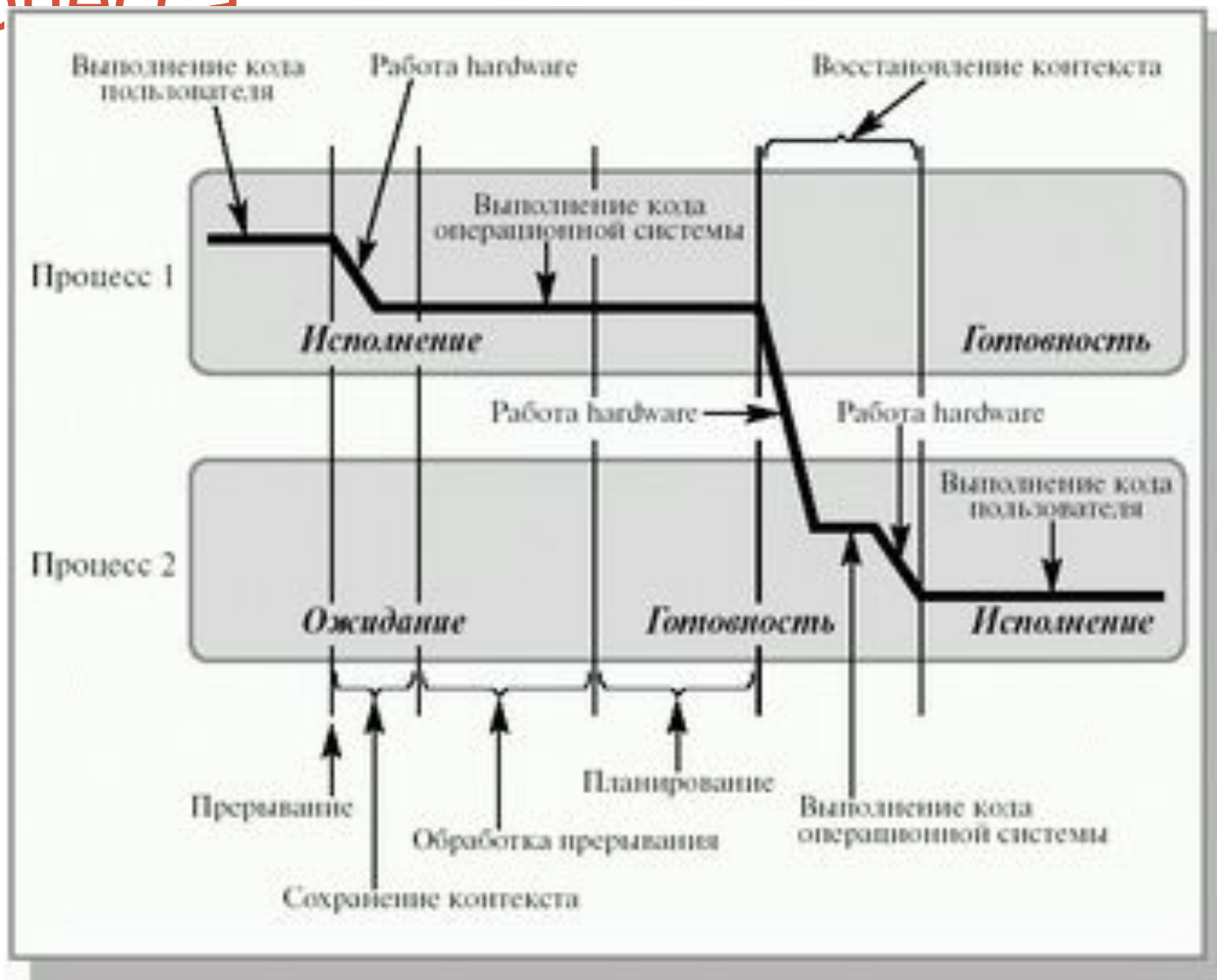
Процессы. Process Control Block

- состояние, в котором находится процесс ;
- программный счетчик процесса или, другими словами, адрес команды, которая должна быть выполнена для него следующей;
- содержимое регистров процессора;
- данные, необходимые для планирования использования процессора и управления памятью (приоритет процесса, размер и расположение адресного пространства и т. д.);
- учетные данные (идентификационный номер процесса, какой пользователь инициировал его работу, общее время использования процессора данным процессом и т. д.);
- сведения об устройствах ввода-вывода, связанных с процессом (например, какие устройства закреплены за процессом, таблицу открытых файлов).

Процессы. Генеалогический лес.



Процессы. Разблокирование процесса



Процессы. Деятельность процесса

A=1
B=2
Read C

CPU Burst

Ожидание окончания
ввода

I/O Burst

A=A+C*B
Print A

CPU Burst

Ожидание окончания
вывода

I/O Burst

Процессы. Случаи выбора нового процесса

- Когда процесс переводится из состояния исполнение в состояние закончил исполнение.
- Когда процесс переводится из состояния исполнение в состояние ожидание.
- Когда процесс переводится из состояния исполнение в состояние готовность (например, после прерывания от таймера).
- Когда процесс переводится из состояния ожидание в состояние готовность (завершилась операция ввода-вывода или произошло другое событие).

Процессы. Случаи выбора нового процесса

- Когда процесс переводится из состояния исполнение в состояние закончил исполнение.
- Когда процесс переводится из состояния исполнение в состояние ожидание.
- Когда процесс переводится из состояния исполнение в состояние готовность (например, после прерывания от таймера).
- Когда процесс переводится из состояния ожидание в состояние готовность (завершилась операция ввода-вывода или произошло другое событие).

Алгоритмы планирования

First-Come, First-Served (FCFS)

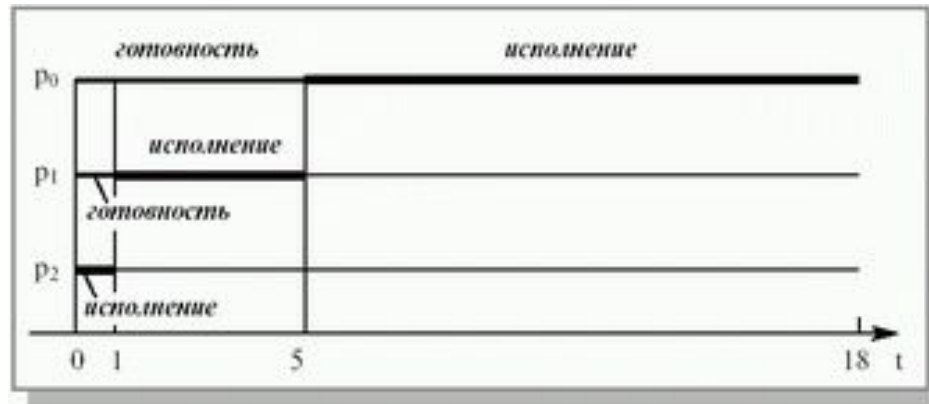
Таблица 3.1.

Процесс	p_0	p_1	p_2
Продолжительность очередного <i>CPU burst</i>	13	4	1



Если процессы расположены в очереди процессов, готовых к исполнению, в порядке p_0, p_1, p_2 , то картина их выполнения выглядит так, как показано на рисунке. Первым для выполнения выбирается процесс p_0 , который получает процессор на все время своего *CPU burst*, т. е. на 13 единиц времени. После его окончания в состояние исполнение переводится процесс p_1 , он занимает процессор на 4 единицы времени. И, наконец, возможность работать получает процесс p_2 . Время ожидания для процесса p_0 составляет 0 единиц времени, для процесса p_1 – 13 единиц, для процесса p_2 – $13 + 4 = 17$ единиц. Таким образом, среднее время ожидания в этом случае – $(0 + 13 + 17) / 3 = 10$ единиц времени. Полное время выполнения для процесса p_0 составляет 13 единиц времени, для процесса p_1 – $13 + 4 = 17$ единиц, для процесса p_2 – $13 + 4 + 1 = 18$ единиц. Среднее полное время выполнения оказывается равным $(13 + 17 + 18) / 3 = 16$ единицам времени.

Алгоритмы планирования



Если те же самые процессы расположены в порядке p_2 , p_1 , p_0 , то картина их выполнения будет соответствовать рисунку. Время ожидания для процесса p_0 равняется 5 единицам времени, для процесса p_1 – 1 единице, для процесса p_2 – 0 единиц. Среднее время ожидания составит $(5 + 1 + 0)/3 = 2$ единицы времени. Это в 5 (!) раз меньше, чем в предыдущем случае. Полное время выполнения для процесса p_0 получается равным 18 единицам времени, для процесса p_1 – 5 единицам, для процесса p_2 – 1 единице. Среднее полное время выполнения составляет $(18 + 5 + 1)/3 = 8$ единиц времени, что почти в 2 раза меньше, чем при первой расстановке процессов.

Алгоритмы планирования

Round Robin



Таблица 3.2.

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
p_0	И	И	И	И	Г	Г	Г	Г	Г	И	И	И	И	И	И	И	И	И
p_1	Г	Г	Г	Г	И	И	И	И										
p_2	Г	Г	Г	Г	Г	Г	Г	Г	И									

Рассмотрим предыдущий пример с порядком процессов p_0 , p_1 , p_2 и величиной кванта времени равной 4. Выполнение этих процессов иллюстрируется таблицей. Обозначение "И" используется в ней для процесса, находящегося в состоянии исполнение, обозначение "Г" – для процессов в состоянии готовность, пустые ячейки соответствуют завершившимся процессам. Состояния процессов показаны на протяжении соответствующей единицы времени, т. е. колонка с номером 1 соответствует промежутку времени от 0 до 1.

Алгоритмы планирования. Round Robin

Таблица 3.2.

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

Первым для исполнения выбирается процесс p_0 . Продолжительность его *CPU burst* больше, чем величина *кванта времени*, и поэтому процесс исполняется до истечения *кванта*, т. е. в течение 4 единиц времени. После этого он помещается в конец очереди готовых к исполнению процессов, которая принимает вид p_1, p_2, p_0 . Следующим начинает выполняться процесс p_1 . Время его исполнения совпадает с величиной выделенного *кванта*, поэтому процесс работает до своего завершения. Теперь очередь процессов в состоянии готовности состоит из двух процессов, p_2 и p_0 . Процессор выделяется процессу p_2 . Он завершается до истечения отпущенного ему процессорного времени, и очередные *кванты* отмеряются процессу p_0 – единственному не закончившему к этому моменту свою работу. Время ожидания для процесса p_0 (количество символов "Г" в соответствующей строке) составляет 5 единиц времени, для процесса p_1 – 4 единицы времени, для процесса p_2 – 8 единиц времени. Таким образом, среднее время ожидания для этого алгоритма получается равным $(5 + 4 + 8)/3 = 5,6(6)$ единицы времени. Полное время выполнения для процесса p_0 (количество непустых столбцов в соответствующей строке) составляет 18 единиц времени, для процесса p_1 – 8 единиц, для процесса p_2 – 9 единиц. Среднее полное время выполнения оказывается равным $(18 + 8 + 9)/3 = 11,6(6)$ единицы времени.

Легко увидеть, что среднее время ожидания и среднее полное время выполнения для обратного порядка процессов не отличаются от соответствующих времен для *алгоритма FCFS* и составляют 2 и 8 единиц времени соответственно.

Алгоритмы планирования. Round Robin

Таблица 3.3.

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
p ₀	и	г	г	и	г	и	г	и	г	и	и	и	и	и	и	и	и	и
p ₁	г	и	г	г	и	г	и	г	и									
p ₂	г	г	и															

На производительность *алгоритма RR* сильно влияет величина *кванта времени*. Рассмотрим тот же самый пример с порядком процессов p_0 , p_1 , p_2 для величины *кванта времени*, равной 1 (табл. 3.3). Время ожидания для процесса p_0 составит 5 единиц времени, для процесса p_1 – тоже 5 единиц, для процесса p_2 – 2 единицы. В этом случае среднее время ожидания получается равным $(5 + 5 + 2)/3 = 4$ единицам времени. Среднее полное время исполнения составит $(18 + 9 + 3)/3 = 10$ единиц времени

При очень больших величинах *кванта времени*, когда каждый процесс успевает завершить свой *CPU burst* до возникновения прерывания по времени, *алгоритм RR* вырождается в *алгоритм FCFS*. При очень малых величинах создается иллюзия того, что каждый из n процессов работает на собственном виртуальном процессоре с производительностью $\sim 1/n$ от производительности реального процессора. Правда, это справедливо лишь при теоретическом анализе при условии пренебрежения временами переключения *контекста процессов*. В реальных условиях при слишком малой величине *кванта времени* и, соответственно, слишком частом переключении контекста накладные расходы на переключение резко снижают производительность системы.

Алгоритмы планирования. Shortest Job First (SJF)

SJF-алгоритм краткосрочного планирования может быть как вытесняющим, так и невытесняющим. При невытесняющем SJF - планировании процессор предоставляется избранному процессу на все необходимое 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.

Алгоритмы планирования. Не вытесняющий SJF

Таблица 3.4.

Процесс	p ₀	p ₁	p ₂	p ₃
Продолжительность очередного CPU burst	5	3	7	1

При использовании невытесняющего алгоритма SJF первым для исполнения будет выбран процесс p₃, имеющий наименьшее значение продолжительности очередного CPU burst. После его завершения для исполнения выбирается процесс p₁, затем p₀ и, наконец, p₂. Эта картина отражена в таблице 3.5.

Таблица 3.5.

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
p ₀	Г	Г	Г	Г	И	И	И	И	И							
p ₁	Г	И	И	И												
p ₂	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И	И
p ₃	И															

Среднее время ожидания для алгоритма SJF составляет $(4 + 1 + 9 + 0)/4 = 3,5$ единицы времени. Легко посчитать, что для алгоритма FCFS при порядке процессов p₀, p₁, p₂, p₃ эта величина будет равняться $(0 + 5 + 8 + 15)/4 = 7$ единицам времени, т. е. будет в два раза больше, чем для алгоритма SJF. Можно показать, что для заданного набора процессов (если в очереди не появляются новые процессы) алгоритм SJF является оптимальным с точки зрения минимизации среднего времени ожидания среди класса невытесняющих алгоритмов.

Алгоритмы планирования. Вытесняющий SJF

Таблица 3.6.

Процесс	Время появления в очереди очередного <i>CPU burst</i>	Продолжительность
P_0	0	6
P_1	2	2
P_2	6	7
P_3	0	5

Таблица 3.7.

[illegible]

Алгоритмы планирования. Гарантированное планирование

При интерактивной работе N пользователей в вычислительной системе можно применить алгоритм *планирования*, который гарантирует, что каждый из пользователей будет иметь в своем распоряжении $\sim 1/N$ часть процессорного времени. Пронумеруем всех пользователей от 1 до N . Для каждого пользователя с номером i введем две величины: T_i – время нахождения пользователя в системе или, другими словами, длительность сеанса его общения с машиной и τ_i – суммарное процессорное время уже, выделенное всем его процессам в течение сеанса. Справедливым для пользователя было бы получение T_i/N процессорного времени. Если

$$\tau_i \ll T_i/N$$

то i -й пользователь несправедливо обделен процессорным временем. Если же

$$\tau_i \gg T_i/N$$

то система явно благоволит к пользователю с номером i . Вычислим для процессов каждого пользователя значение коэффициента справедливости

$$\tau_i N / T_i$$

и будем предоставлять очередной *квант времени* готовому процессу с наименьшей величиной этого отношения. Предложенный алгоритм называют алгоритмом *гарантированного планирования*. К недостаткам этого алгоритма можно отнести невозможность предугадать поведение пользователей. Если некоторый пользователь отправится на пару часов пообедать и поспать, не прерывая сеанса работы, то по возвращении его процессы будут получать неоправданно много процессорного времени.

[illegible]

Многоуровневые очереди (Multilevel Queue)



Многоуровневые очереди с обратной связью (Multilevel Feedback Queue)

