Разработка системных приложений

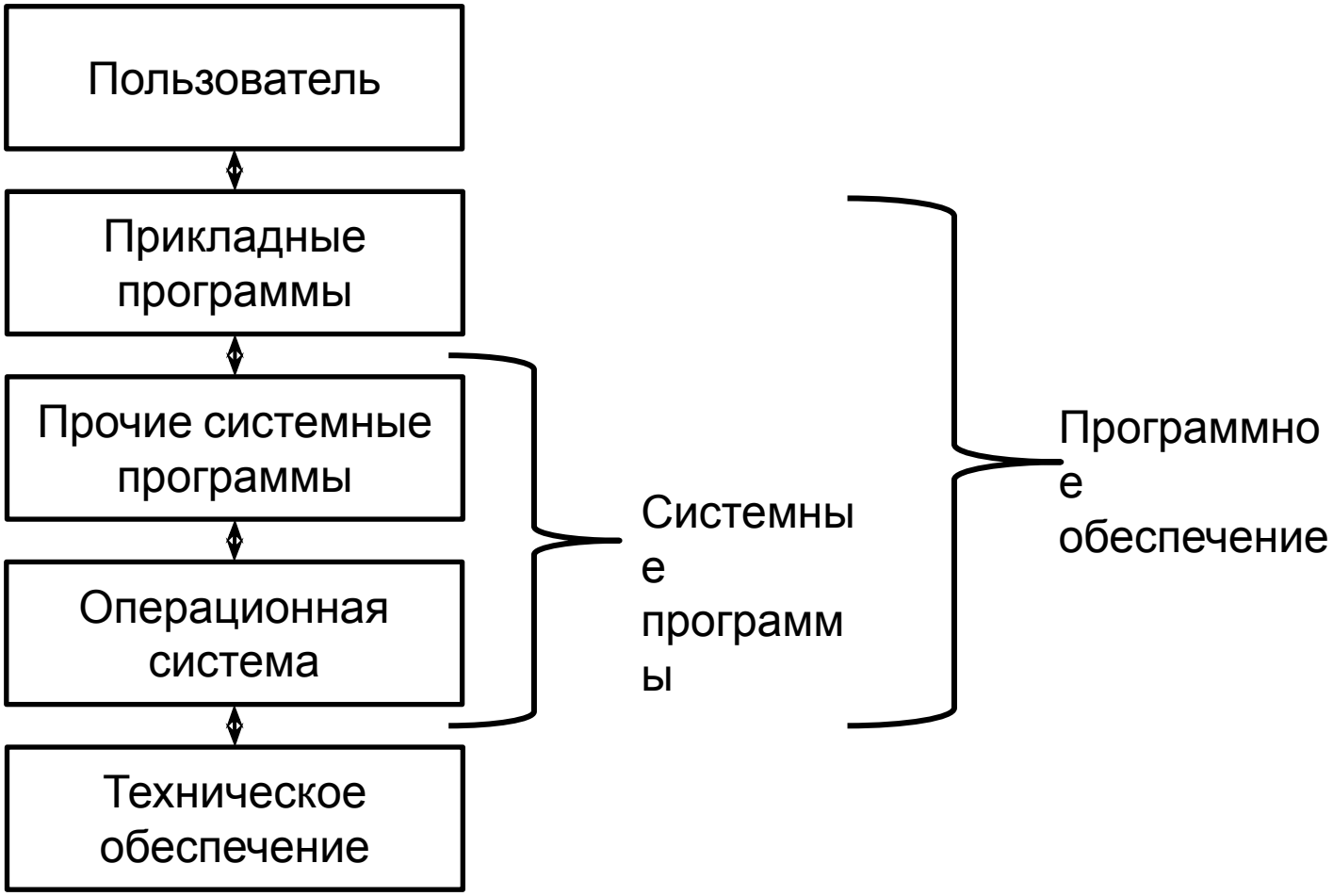
Основные понятия. Процессы.

Планирование процессов. Алгоритмы планирования.

Литература

- Таненбаум Э., Вудхалл А., Операционные системы. Разработка и реализация.
- Таненбаум Э. Современные операционные системы.
- Richard L. Halterman, Fundamentals of C++ Programming.

Структура вычислительной системы



Основные понятия.

- ***Системные вызовы*** (system calls) – это интерфейс между *операционной системой* и пользовательской программой. Они создают, удаляют и используют различные объекты, главные из которых – процессы и файлы. Пользовательская программа запрашивает сервис у *операционной системы*, осуществляя *системный вызов*
- существуют библиотеки процедур, которые загружают машинные регистры определенными параметрами и осуществляют *прерывание процессора*, после чего управление передается обработчику данного *вызова*, входящему в ядро *операционной системы*. Цель таких библиотек – сделать *системный вызов* похожим на обычный *вызов* подпрограммы

- при *системном вызове* задача переходит в привилегированный режим или режим ядра (kernel mode)
- в этом режиме работает код ядра *операционной системы*, причем исполняется он в адресном пространстве и в контексте вызвавшей его задачи. Таким образом, ядро *операционной системы* имеет полный доступ к памяти пользовательской программы, и при *системном вызове* достаточно передать адреса одной или нескольких областей памяти с параметрами *вызова* и адреса одной или нескольких областей памяти для результатов *вызова*.

- ***Прерывание*** (hardware interrupt) – это событие, генерируемое внешним (по отношению к *процессору*) устройством.
- Посредством аппаратных *прерываний* аппаратура либо информирует центральный *процессор* о том, что произошло какое-либо событие, требующее немедленной реакции (например, пользователь нажал клавишу), либо сообщает о завершении асинхронной операции ввода-вывода (например, закончено чтение данных с диска в основную память)

- Важный тип аппаратных *прерываний – прерывания* таймера, которые генерируются периодически через фиксированный промежуток времени.
- *Прерывания* таймера используются *операционной системой* при планировании процессов. Каждый тип аппаратных *прерываний* имеет собственный номер, однозначно определяющий источник *прерывания*.

- ***Исключительная ситуация*** (exception) – событие, возникающее в результате попытки выполнения программой команды, которая по каким-то причинам не может быть выполнена до конца.
- *Исключительные ситуации* можно разделить на исправимые и неисправимые

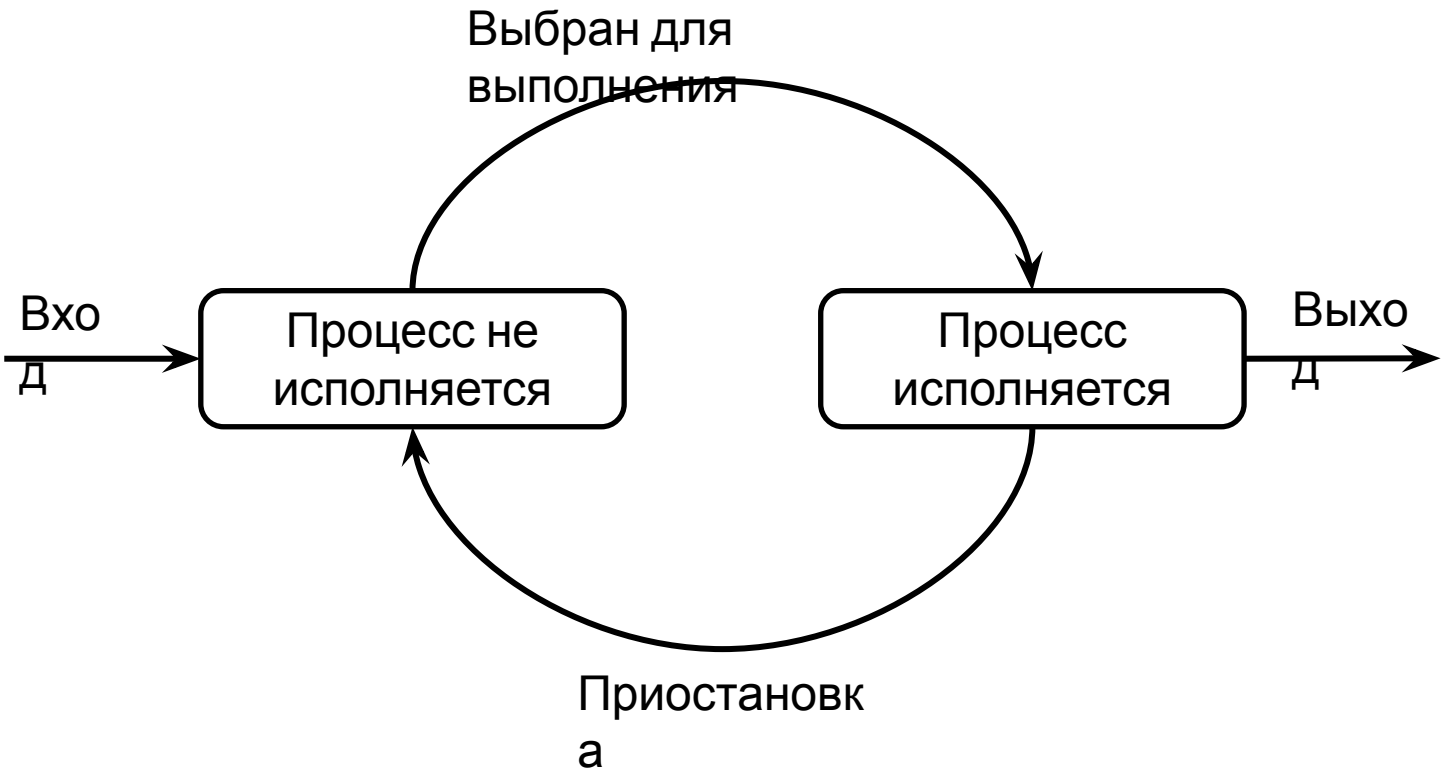
Процесс

- ***Процесс*** – некоторая совокупность набора исполняющихся команд, ассоциированных с ним ресурсов (выделенная для исполнения память или адресное пространство, стеки, используемые файлы и устройства ввода-вывода и т. д.) и текущего момента его выполнения (значения регистров, программного счетчика, состояние стека и значения переменных), находящуюся под управлением операционной системы

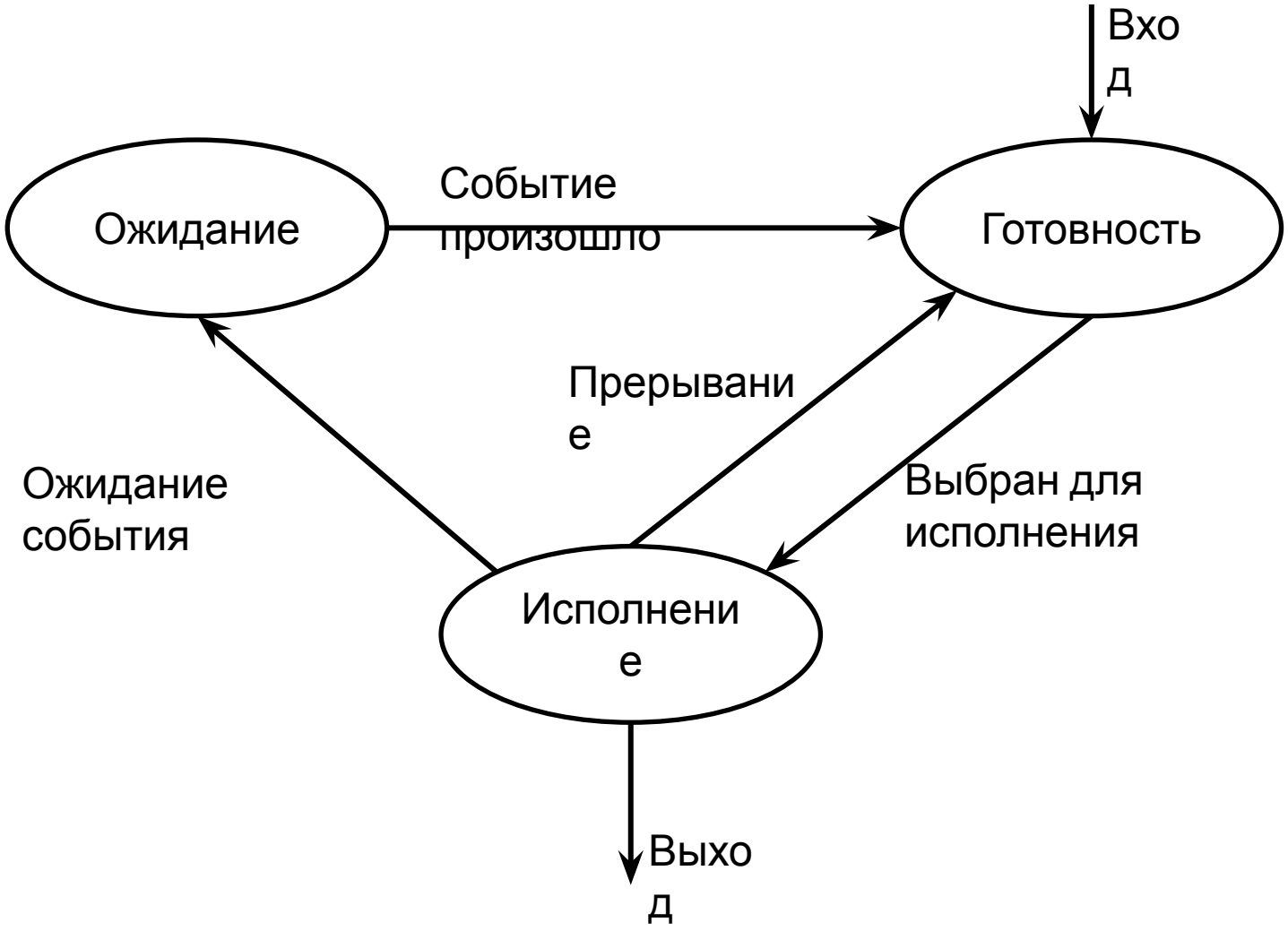
Процесс != программа

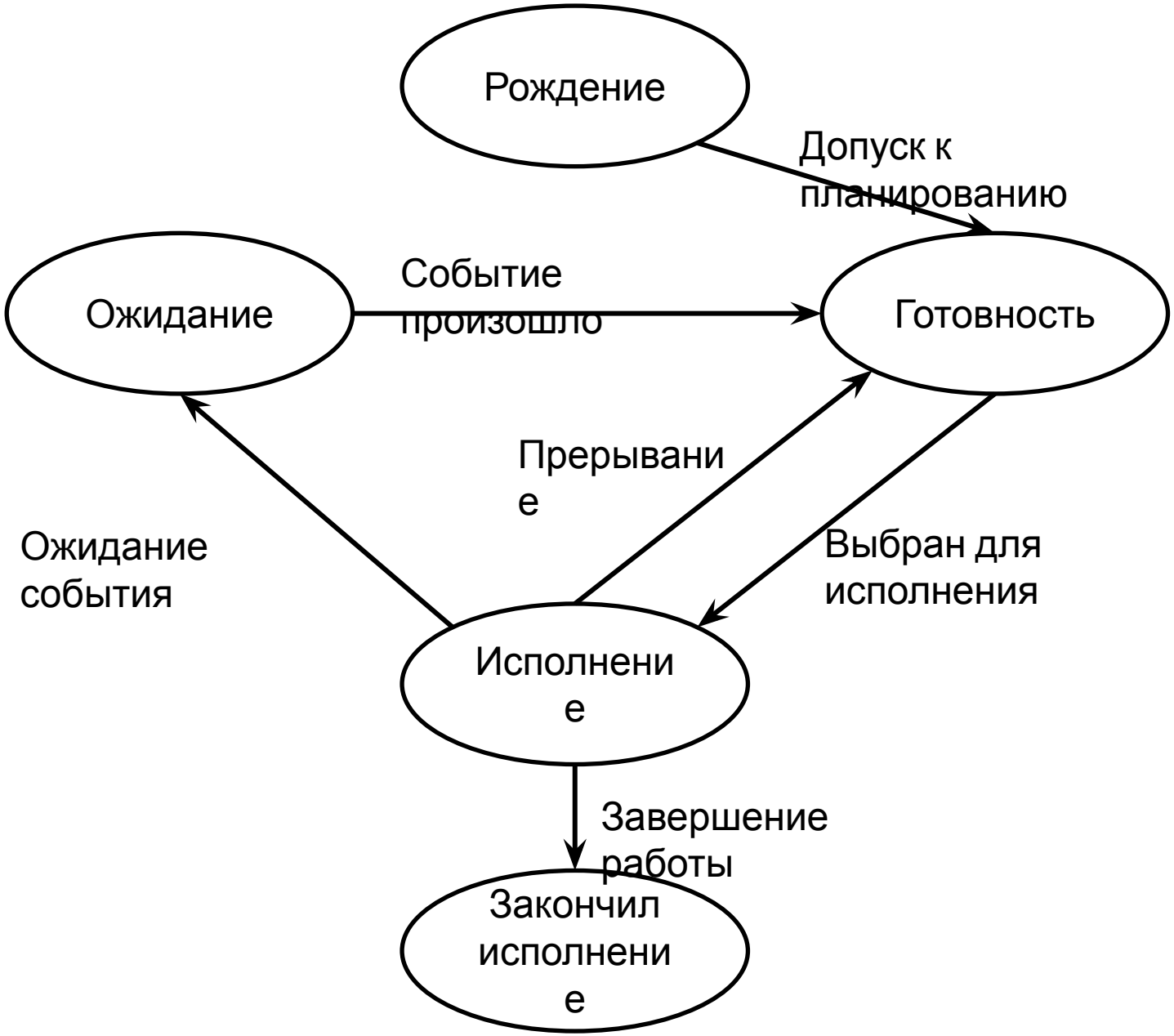
- в некоторых операционных системах для работы определенных программ может организовываться более одного *процесса* или один и тот же *процесс* может исполнять последовательно несколько различных программ
- даже в случае обработки только одной программы в рамках одного *процесса* нельзя считать, что *процесс* представляет собой просто динамическое описание кода исполняемого файла, данных и выделенных для них ресурсов

Состояния процесса



Состояния процесса




Рождение
Допуск к планированию
Ожидание
Событие произошло
Готовность
Прерывание
Ожидание события
Выбран для исполнения
Исполнение
Завершение работы
Закончил исполнение

Операции над процессами

- создание процесса – завершение процесса
- приостановка процесса – запуск процесса
- блокирование процесса – разблокирование процесса
- изменение приоритета процесса

Изменением *состояния процессов* занимается операционная система, совершая *операции* над ними

Блок управления процессом. Process Control Block(PCB)

- *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)

Контекст процесса.

- Содержимое всех регистров процессора (включая значение программного счетчика) называется *регистровым контекстом процесса,*
- а все остальное – *системным контекстом процесса*.

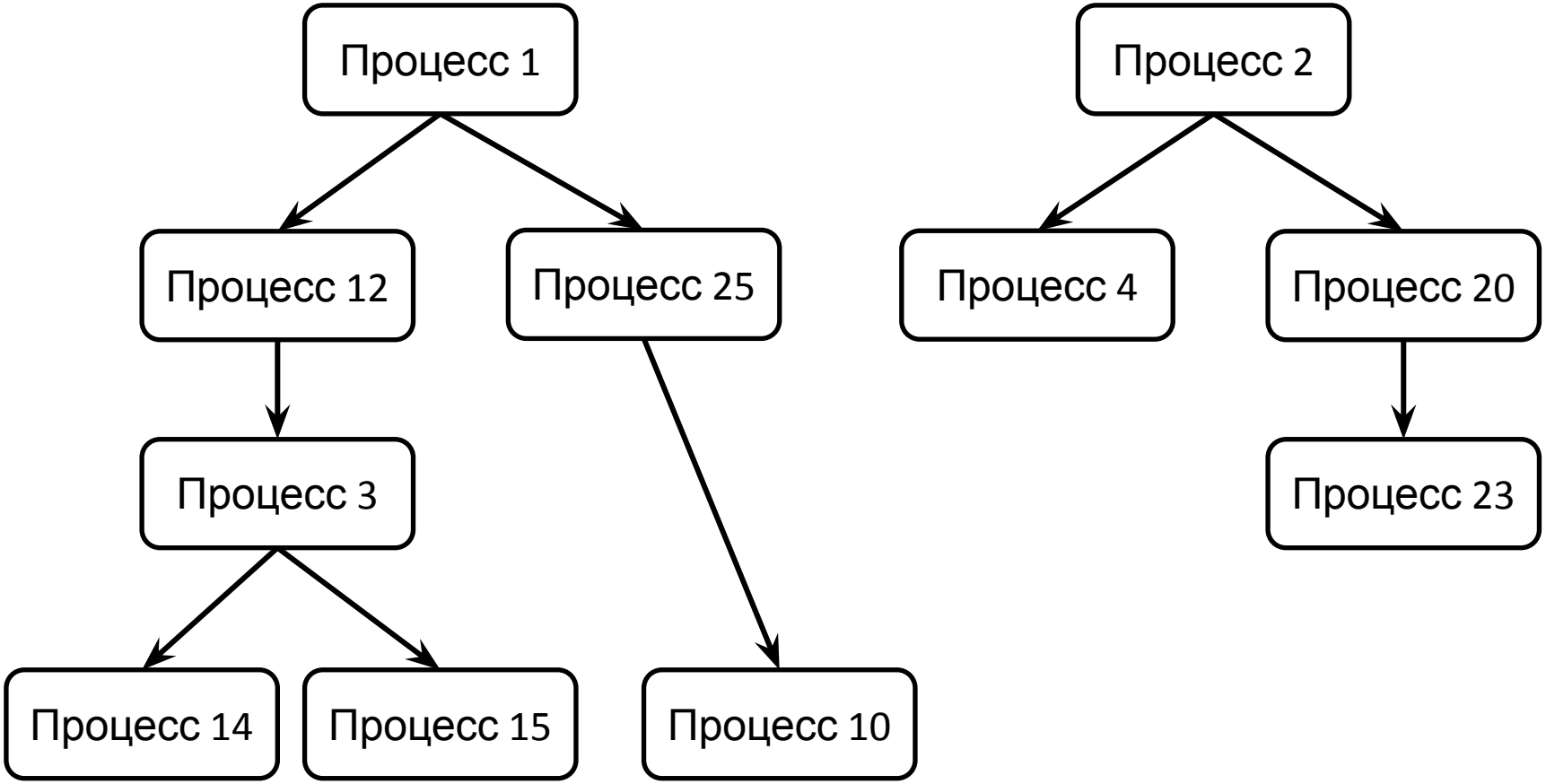
Контекст процесса.

- Код и данные, находящиеся в адресном пространстве *процесса*, называются *пользовательским контекстом*
- Совокупность *регистрового, системного* и *пользовательского контекстов процесса* для краткости принято называть просто ***контекстом процесса***

Одноразовые операции. Рождение процесса.

- Инициатором рождения нового *процесса* после старта операционной системы может выступить либо *процесс* пользователя, совершивший специальный системный вызов, либо сама операционная система, то есть, в конечном итоге, тоже некоторый *процесс*.
- *Процесс*, инициировавший *создание* нового *процесса*, принято называть процессом-родителем (parent process), а вновь созданный *процесс* – процессом-ребенком (child process).

Генеалогическое древо процессов



Этапы рождения процесса

1. система заводит новый *PCB* с *состоянием процесса* «*рождение*» и начинает его заполнять. Новый *процесс* получает уникальный идентификационный номер
2. процессу выделяются ресурсы(память, файлы, устройства ввода-вывода и т. д.)
3. в адресное пространство процесса вносятся программный код, значения данных и устанавливается программный счетчик
4. после того как *процесс* наделен содержанием, в *PCB* дописывается оставшаяся информация, и *состояние* нового *процесса* изменяется на «*готовность*»

Завершение жизненного цикла процесса

- После того как *процесс* завершил свою работу, операционная система переводит его в *состояние* «закончил исполнение» и освобождает все его ресурсы, делая соответствующие записи в *PCB*. При этом сам *PCB* не уничтожается, а остается в системе еще некоторое время
- В операционной системе Unix *процессы*, находящиеся в *состоянии* «закончил исполнение», принято называть *процессами-зомби*

Многоразовые операции. Запуск процесса.

1. Из числа *процессов*, находящихся в *состоянии* готовность, операционная система выбирает один *процесс* для последующего исполнения.
2. Для избранного *процесса* операционная система обеспечивает наличие в оперативной памяти информации, необходимой для его дальнейшего выполнения.
3. Далее *состояние процесса* изменяется на *исполнение*, восстанавливаются значения регистров для данного *процесса* и управление передается команде, на которую указывает счетчик команд *процесса*.
4. Все данные, необходимые для восстановления *контекста*, извлекаются из *PCB процесса*, над которым совершается *операция*.

Многоразовые операции. Приостановка процесса.

- работа *процесса*, находящегося в *состоянии* «исполнение», приостанавливается в результате какого-либо прерывания.
- процессор автоматически сохраняет счетчик команд и, возможно, один или несколько регистров в стеке исполняемого *процесса*
- затем передает управление по специальному адресу обработки данного прерывания. На этом деятельность hardware по обработке прерывания завершается. По указанному адресу обычно располагается одна из частей операционной системы.
- она сохраняет динамическую часть *системного* и *регистрового контекстов процесса* в его *PCB*, переводит *процесс* в *состояние* «готовность» и приступает к обработке прерывания, то есть к выполнению определенных действий, связанных с возникшим прерыванием.

Многоразовые операции. Блокирование процесса.

- *процесс* блокируется, когда он не может продолжать работу, не дождавшись возникновения какого-либо события в вычислительной системе
- для этого он обращается к операционной системе с помощью определенного системного вызова
- операционная система обрабатывает системный вызов (инициализирует операцию ввода-вывода, добавляет *процесс* в очередь *процессов*, дожидающихся освобождения устройства или возникновения события, и т. д.) и переводит *процесс* из *состояния* «исполнение» в *состояние* «ожидание»

Многоразовые операции. Разблокирование процесса.

- после возникновения в системе какого-либо события операционной системе нужно точно определить, какое именно событие произошло.
- затем операционная система проверяет, находился ли некоторый *процесс* в *состоянии* ожидание для данного события, и если находился, переводит его в *состояние* готовность, выполняя необходимые действия, связанные с наступлением события (инициализация *операции* ввода-вывода для очередного ожидающего *процесса* и т. п.).

Переключение контекста

- Для корректного переключения процессора с одного *процесса* на другой необходимо сохранить *контекст* исполнявшегося *процесса* и восстановить *контекст процесса*, на который будет переключен процессор. Такая процедура сохранения/восстановления работоспособности *процессов* называется *переключением контекста*

Планирование процессов

- Всякий раз, когда нам приходится иметь дело с ограниченным количеством ресурсов и несколькими их потребителями мы вынуждены заниматься распределением наличных ресурсов между потребителями или, другими словами, планированием использования ресурсов
- Процесс *планирования* осуществляется частью операционной системы, называемой планировщиком

Уровни планирования процессов

- *Долгосрочное планирование процессов*
- *Краткосрочное планирование процессов*
- *Среднесрочное планирование процессов*

Долгосрочное планирование процессов

- отвечает за порождение новых процессов в системе, определяя ее ***степень мультипрограммирования***
- осуществляется достаточно редко, между появлением новых процессов могут проходить минуты и даже десятки минут
- если *степень мультипрограммирования* системы поддерживается постоянной, то новые процессы могут появляться только после завершения ранее загруженных
- решение о выборе для запуска того или иного процесса оказывает влияние на функционирование вычислительной системы на протяжении достаточно длительного времени

Краткосрочное планирование процессов

- проводится, к примеру, при обращении исполняющегося процесса к устройствам ввода-вывода или просто по завершении определенного интервала времени
- осуществляется, как правило, не реже одного раза в 100 миллисекунд
- выбор нового процесса для исполнения оказывает влияние на функционирование системы до наступления очередного аналогичного события, т. е. в течение короткого промежутка времени

Среднесрочное планирование процессов

- В некоторых вычислительных системах бывает выгодно для повышения производительности временно удалить какой-либо частично выполнившийся процесс из оперативной памяти на диск, а позже вернуть его обратно для дальнейшего выполнения. Такая процедура в англоязычной литературе получила название swapping (свопинг)

Критерии планирования.

- Справедливость – гарантировать каждому процессу определенную часть времени использования процессора в системе, стараясь не допустить возникновения ситуации, когда процесс одного пользователя постоянно занимает процессор, в то время как процесс другого пользователя фактически не начинал выполняться.
- Эффективность – постараться занять процессор на все 100% рабочего времени, не позволяя ему простаивать в ожидании процессов, готовых к исполнению. В реальных вычислительных системах загрузка процессора колеблется от 40 до 90%.
- Сокращение полного времени выполнения (***turnaround time***) – обеспечить минимальное время между стартом процесса или постановкой задания в очередь для загрузки и его завершением.
- Сокращение времени ожидания (***waiting time***) – сократить время, которое проводят процессы в состоянии готовность и задания в очереди для загрузки.
- Сокращение времени отклика (***response time***) – минимизировать время, которое требуется процессу в

Требования к алгоритмам планирования

- **Предсказуемость**. Одно и то же задание должно выполняться приблизительно за одно и то же время. Применение алгоритма *планирования* не должно приводить, к примеру, к извлечению квадратного корня из 4 за сотые доли секунды при одном запуске и за несколько суток – при втором запуске.
- **Минимальные накладные расходы**. Если на каждые 100 миллисекунд, выделенные процессу для использования процессора, будет приходиться 200 миллисекунд на определение того, какой именно процесс получит процессор в свое распоряжение, и на переключение контекста, то такой алгоритм, очевидно, применять не стоит.
- **Равномерная загрузка ресурсов вычислительной системы**, отдавая предпочтение тем процессам, которые будут занимать малоиспользуемые ресурсы.
- **Масштабируемость**, т. е. алгоритм не сразу теряет работоспособность при увеличении нагрузки. Например, рост количества процессов в системе в два раза не должен приводить к увеличению полного времени выполнения процессов на порядок.

Параметры планирования

- **Статические параметры** – не изменяются в ходе функционирования вычислительной системы. Для *вычислительной системы* это предельные значения ее ресурсов (размер оперативной памяти, максимальное количество памяти на диске для осуществления свопинга, количество подключенных устройств ввода-вывода и т. п.)
- **Динамические параметры** – подвержены постоянным изменениям. Для *вычислительной системы* это количество свободных ресурсов на данный момент.

Статические параметры процесса

- Каким пользователем запущен процесс или сформировано задание.
- Насколько важной является поставленная задача, т. е. каков *приоритет* ее выполнения.
- Сколько процессорного времени запрошено пользователем для решения задачи.
- Каково соотношение процессорного времени и времени, необходимого для осуществления операций ввода-вывода.
- Какие ресурсы вычислительной системы (оперативная память, устройства ввода-вывода, специальные библиотеки и системные программы и т. д.) и в каком количестве необходимы заданию

Планирование процессов

Четыре случая, при которых происходит выбор процесса для исполнения из числа готовых к исполнению:

1. Когда процесс переводится из состояния исполнение в состояние закончил исполнение.
2. Когда процесс переводится из состояния исполнение в состояние ожидание.
3. Когда процесс переводится из состояния исполнение в состояние готовность (например, после прерывания от таймера).
4. Когда процесс переводится из состояния ожидание в состояние готовность

Вытесняющее и невытесняющее планирование

- Если в операционной системе планирование осуществляется только в вынужденных ситуациях, говорят, что имеет место *невытесняющее (nonpreemptive) планирование*.
- Если планировщик принимает и вынужденные, и невынужденные решения, говорят о *вытесняющем (preemptive) планировании*.

Невытесняющее планирование

- При таком режиме *планирования* процесс занимает столько процессорного времени, сколько ему необходимо. При этом переключение процессов возникает только при желании самого исполняющегося процесса передать управление (для ожидания завершения операции ввода-вывода или по окончании работы)
- Однако при *невытесняющем планировании* возникает проблема возможности полного захвата процессора одним процессом, который вследствие каких-либо причин (например, из-за ошибки в программе) зацикливается и не может передать управление другому процессу

Вытесняющее планирование

- В этом режиме *планирования* процесс может быть приостановлен в любой момент исполнения. Операционная система устанавливает специальный таймер для генерации сигнала прерывания по истечении некоторого интервала времени – *кванта*.
- После прерывания процессор передается в распоряжение следующего процесса. Временные прерывания помогают гарантировать приемлемое время отклика процессов для пользователей, работающих в диалоговом режиме, и предотвращают "зависание" компьютерной системы из-за зацикливания какой-либо программы.

Алгоритмы планирования

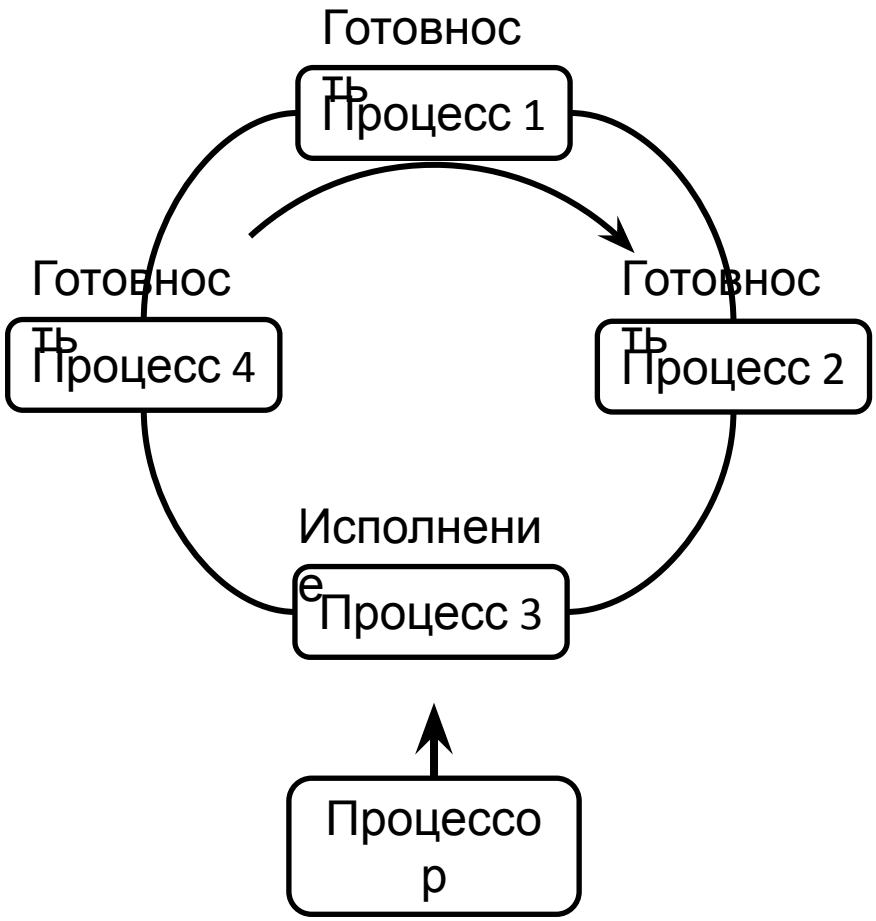
- First-Come, First-Served (FCFS)
- Round Robin (RR)
- Shortest-Job-First (SJF)
- Гарантированное планирование
- Приоритетное планирование
- Многоуровневые очереди (Multilevel Queue)
- Многоуровневые очереди с обратной связью (Multilevel Feedback Queue)

First-Come, First-Served (FCFS)

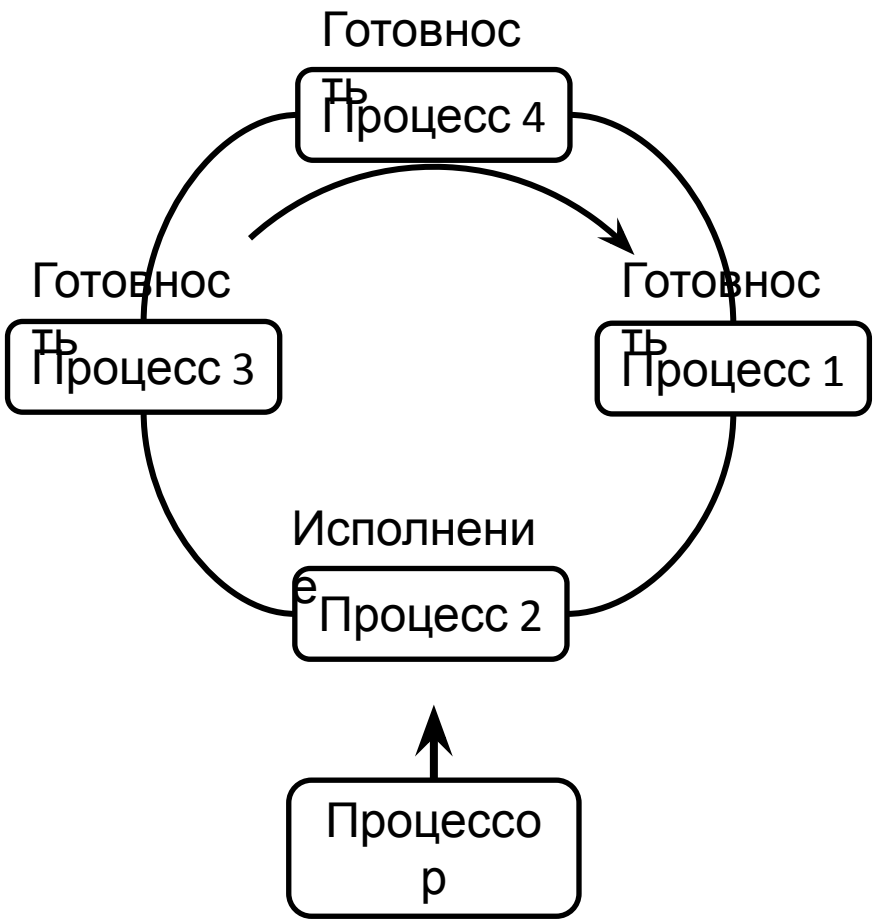
Представим себе, что процессы, находящиеся в состоянии «готовность», выстроены в очередь FIFO.

1. Когда процесс переходит в состояние «готовность», он, помещается в конец этой очереди.
2. Выбор нового процесса для исполнения осуществляется из начала очереди с удалением оттуда ссылки на него.
3. Процесс, получивший в свое распоряжение процессор, занимает его до перехода в состояние «ожидание» или «закончил исполнение».
4. После этого для выполнения выбирается новый процесс из начала очереди.

Round Robin (RR)



Начальный момент времени



По прошествии кванта времени

Round Robin (RR)

- Реализуется такой алгоритм так же, как и предыдущий, с помощью организации процессов, находящихся в состоянии «готовность», в очередь FIFO. Планировщик выбирает для очередного исполнения процесс, расположенный в начале очереди, и устанавливает таймер для генерации прерывания по истечении определенного *кванта времени*.

RR. При выполнении процесса возможны два варианта:

- Время непрерывного использования процессора, необходимое процессу, меньше или равно продолжительности *кванта времени*. Тогда процесс по своей воле освобождает процессор до истечения *кванта времени*, на исполнение поступает новый процесс из начала очереди, и таймер начинает отсчет *кванта* заново.
- Время необходимое процессу больше, чем *квант времени*. Тогда по истечении этого *кванта* процесс прерывается таймером и помещается в конец очереди процессов, готовых к исполнению, а процессор выделяется для использования процессу, находящемуся в ее начале.

Shortest-Job-First (SJF)

- Если короткие задачи расположены в очереди ближе к ее началу, то общая производительность этих алгоритмов значительно возрастает.
- Если бы мы знали время выполнения для процессов, находящихся в состоянии готовность, то могли бы выбрать для исполнения не процесс из начала очереди, а процесс с минимальной длительностью времени выполнения