Лекция №12

Управление процессами

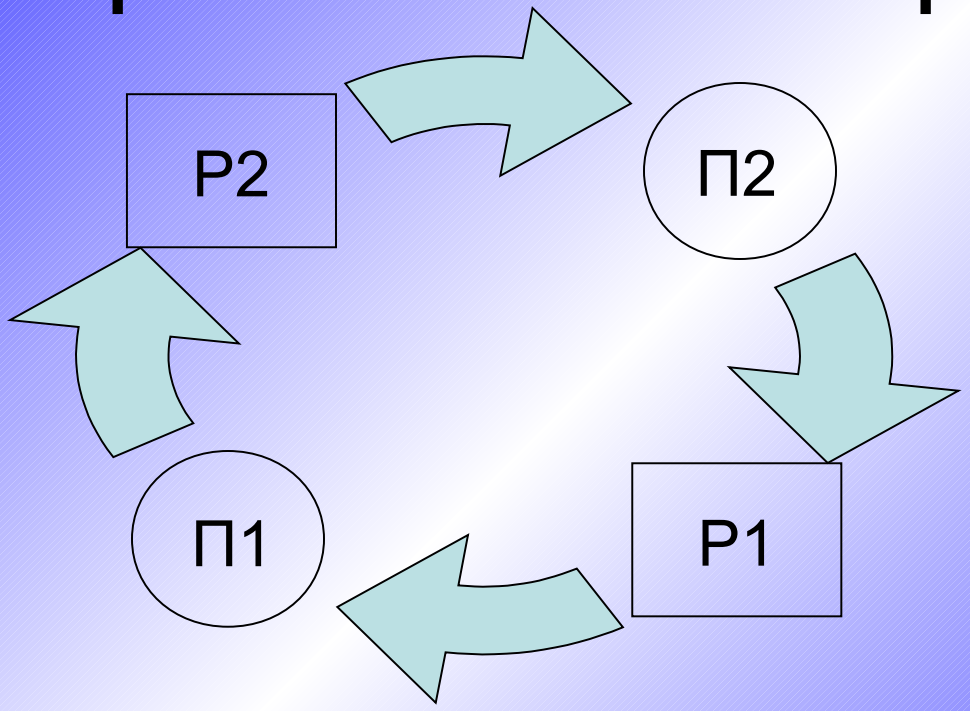
(часть 3)

Тупиковые ситуации

Определение

Взаимоблокировка (тупиковая ситуация) – состояние в операционной системе, когда процессы не могут получить доступ к какому-либо ресурсу из-за невозможности его освобождения одним из процессов.

Пример взаимоблокировки



Р1, Р2 – ресурсы; П1, П2 – процессы; ребро от процесса к ресурсу – запрос процесса на ресурс; ребро от ресурса к процессу – ресурс занят указанным процессом.

Условия взаимоблокировки

Для того, чтобы произошла взаимоблокировка, должны выполниться четыре условия.

- Условие взаимного исключения – каждый ресурс в данный момент или отдан ровно одному процессу или доступен.
- Условие удержания и ожидания – процессы, в данный момент удерживающие полученные ранее ресурсы могут запрашивать новые ресурсы.

Условия взаимоблокировки

- Условие отсутствия принудительной выгрузки ресурса – у процесса нельзя принудительным образом забрать ранее полученные ресурсы; процесс, владеющий ими должен сам освободить ресурсы.
- Условие циклического ожидания – должна существовать круговая последовательность из двух и более процессов, каждый из которых ждёт доступа к ресурсу, удерживаемому следующим членом последовательности.

Стратегии, используемые по отношению к взаимоблокировкам

- Игнорирование проблемы.
- Обнаружение взаимоблокировки и восстановление работоспособности.
- Динамическое избежание взаимоблокировок с помощью аккуратного распределения ресурсов.
- Предотвращение с помощью структурного опровержения одного из четырёх условий, необходимых для взаимоблокировки.

Игнорирование проблемы

- Часть взаимоблокировок не обнаруживаются.
- В случае невысокой частоты появления взаимоблокировок их можно игнорировать.
- Учёт взаимоблокировок приводит к наложению ограничений на процессы и к затрачиванию процессорного времени на проверку различных условий.

Обнаружение и устранение взаимоблокировок

Решаемые задачи:

- обнаружение взаимоблокировки при наличии одного ресурса каждого типа;
- обнаружение взаимоблокировки при наличии нескольких ресурсов каждого типа;
- выход из взаимоблокировки.

Обнаружение взаимоблокировки при наличии одного ресурса каждого типа

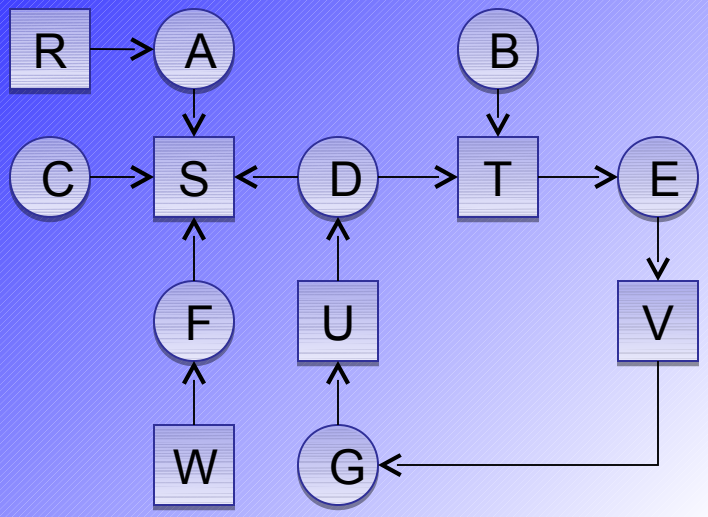
Метод обнаружения:

- построение графа использования ресурсов;
- проверка графа на наличие циклов.

Алгоритм обнаружения циклов в направленном графе

1. Для каждого узла N в графе выполняются следующие пять шагов, где N является начальным узлом.
2. Начальные условия: L – пустой список, все рёбра не маркированы.
3. Текущий узел добавляем в конец списка L и проверяем количество появлений узла в списке. Если узел присутствует в двух местах, то граф содержит цикл и работа алгоритма завершается.
4. Для заданного узла проверяем, выходит ли из него хотя бы одно немаркированное ребро. Если да, то переход к шагу 5, если нет, то переход к шагу 6.
5. Выбирается и отмечается любое немаркированное исходящее ребро. Затем по нему переходим к новому текущему узлу и возврат к шагу 3.
6. Удаление последнего узла из списка и возвращение к предыдущему узлу. Обозначаем его текущим узлом и возврат к шагу 3. Если этот узел первоначальный, то граф не содержит циклов – завершение алгоритма.

Пример обнаружения цикла



Условия:

- процесс A занимает ресурс R и пытается получить ресурс S;
- процесс B ничего не использует, но пытается получить ресурс T;
- процесс C ничего не использует, но пытается получить ресурс S;
- процесс D занимает ресурс U и пытается получить ресурсы S и T;
- процесс E занимает ресурс T и пытается получить ресурс V;
- процесс F занимает ресурс W и пытается получить ресурс S;
- процесс G занимает ресурс V и пытается получить ресурс U.

Обнаружение взаимоблокировки при наличии нескольких ресурсов каждого типа

Метод обнаружения:

- построение вектора существующих ресурсов;
- построение вектора доступных ресурсов;
- построение матрицы текущего распределения ресурсов;
- построение матрицы запросов на ресурсы;
- поиск заблокированных процессов путём сравнения векторов.

Обнаружение взаимоблокировки при наличии нескольких ресурсов каждого типа

- $P_1..P_n$ – процессы.
- $E_1..E_m$ – классы ресурсов (E_1 – количество ресурсов класса 1).

Существующие ресурсы
$(E_1, E_2, E_3, \ldots, E_m)$

Доступные ресурсы
$(A_1, A_2, A_3, \ldots, A_m)$

Матрица текущего распределения ресурсов

$$\begin{vmatrix} C_{11} & C_{12} & C_{13} & \ldots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \ldots & C_{2m} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ C_{n1} & C_{n2} & C_{n3} & \ldots & C_{nm} \end{vmatrix}$$

Матрица запросов

$$\begin{vmatrix} R_{11} & R_{12} & R_{13} & \ldots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \ldots & R_{2m} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ R_{n1} & R_{n2} & R_{n3} & \ldots & R_{nm} \end{vmatrix}$$

Таким образом, $\sum_{i=1}^{n} C_{ij} + A_j = E_j$, где *i* – текущий процесс, *j* – текущий класс ресурсов.

Обнаружение взаимоблокировки при наличии нескольких ресурсов каждого типа

- Начальное состояние – все процессы немаркированы.
- Маркируются процессы, которые могут завершиться и, следовательно, не находятся в тупике.

Алгоритм:

1. поиск немаркированного процесса P_i, для которого i-я строка матрицы R меньше вектора A или равна ему;
2. если такой процесс найден, то i-я строка матрицы C прибавляется к вектору A, процесс маркируется. Возврат к шагу 1;
3. если таких процессов нет, то алгоритм завершается.

Пример использования алгоритма

E=(4 2 3 1) – количество ресурсов каждого вида

A=(2 1 0 0) – количество нераспределённых ресурсов каждого вида

Матрица текущего распределения ресурсов

$$C=\begin{vmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{vmatrix}$$

Матрица запросов

$$R=\begin{vmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{vmatrix}$$

Порядок завершения процессов: 3-й – 2-й – 1-й.

Выход из взаимоблокировки

- Восстановление при помощи принудительной выгрузки ресурса (остановка процесса и дальнейшее его восстановление).
- Восстановление через откат (периодическое создание контрольных точек процессов).
- Восстановление путём уничтожения процессов.

Выбор способа зависит от типа используемого ресурса и возможности приостановки процесса или его повторного запуска.

Избежание взаимоблокировок

- Безопасное состояние – состояние без взаимоблокировки с существующим порядком планирования, при котором каждый процесс может работать до завершения, даже если все процессы немедленно захотят получить все необходимые им ресурсы.
- В безопасном состоянии система может гарантировать, что все процессы завершат свою работу, и не будет взаимоблокировки.
- Необходимо заранее знать какие ресурсы и сколько их необходимо каждому процессу.

Алгоритм исключения тупиков для одного вида ресурсов

Имеется ресурсов

Максимально необходимо ресурсов

а

A	0	6
B	0	5
C	0	4
D	0	7

Свободно ресурсов – 10

б

A	1	6
B	1	5
C	2	4
D	4	7

Свободно ресурсов – 2

в

A	1	6
B	2	5
C	2	4
D	4	7

Свободно ресурсов – 1

Состояния (а) и (б) – безопасные, состояние (в) – небезопасное.

Алгоритм исключения взаимоблокировок состоит в проверке безопасности состояния при каждом запросе на ресурс и выдаче этого ресурса только в случае, если это приведёт к безопасному состоянию.

Алгоритм исключения тупиков для нескольких видов ресурсов

E=(6342) – количество ресурсов каждого вида

A=(1020) – количество нераспределённых ресурсов каждого вида

A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Распределённые ресурсы

A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Необходимые ресурсы

Алгоритм аналогичен алгоритму исключения тупиков для одного вида ресурсов (проверка безопасности состояния проходит по всем видам ресурсов). Текущее состояние является безопасным (порядок завершения процессов: D – E – A – B – C).

Предотвращение взаимоблокировок

Необходимо опровержение одного из четырёх условий возникновения взаимоблокировки.

- Атака условия взаимного исключения.
- Атака условия удержания и ожидания.
- Атака условия отсутствия принудительной выгрузки ресурса.
- Атака условия циклического ожидания.

Атака условия взаимного исключения

- Нельзя предоставить принтер в одновременное пользование двум процессам.
- Не все ресурсы могут поддерживать создание временного хранилища данных на диске.

Атака условия удержания и ожидания

- Не всегда можно заранее предусмотреть количество используемых процессом ресурсов.
- В случае выделения всех необходимых ресурсов на всё время выполнения процесса ресурсы могут долго оставаться занятыми.
- Вариант нарушения условия – требование к процессу о временном освобождении используемых в данный момент ресурсов и дальнейшее выделение сразу всех необходимых.

Атака условия отсутствия принудительной выгрузки ресурса

- Насильственное изъятие у процесса используемого ресурса не всегда возможно (печать на принтере, запись компакт-диска).

Атака условия циклического ожидания

Варианты устранения циклического ожидания:

- требование к использованию процессом одного ресурса в один момент времени; при запросе ко второму – освобождение первого;
- общая нумерация всех ресурсов; при этом процесс может запрашивать ресурс только с последующим номером.

Наследование ресурсов

Варианты наследования

- Создание потока в рамках самого процесса (каждый из потоков получает доступ к используемым ресурсам, определяемым программистом).
- Создание дочернего процесса (дочерний процесс получает доступ только на чтение ресурсов родительского).

Преимущества наследования ресурсов процессами

- При выполнении подпрограммы поток, написанный с ошибкой, может повлиять на корректность работы всего процесса.
- При выполнении подпрограммы создание нового потока приводит к необходимости реализации синхронизации между потоками программистом.

Использование дочерних процессов

- Родительский процесс может синхронизировать своё исполнение с завершением процесса (ожидать завершения дочернего процесса).
- Возможен обмен информацией между родительским и дочерним процессом при помощи средств межпроцессного взаимодействия (например, разделяемой памяти).

Использование дочерних процессов

- Дочерний процесс копирует адресное пространство родительского в своё виртуальное адресное пространство и продолжает пользоваться копией.
- В UNIX-системах существует чёткая иерархия процессов. Существует загрузочный процесс, являющийся корнем иерархии процессов.

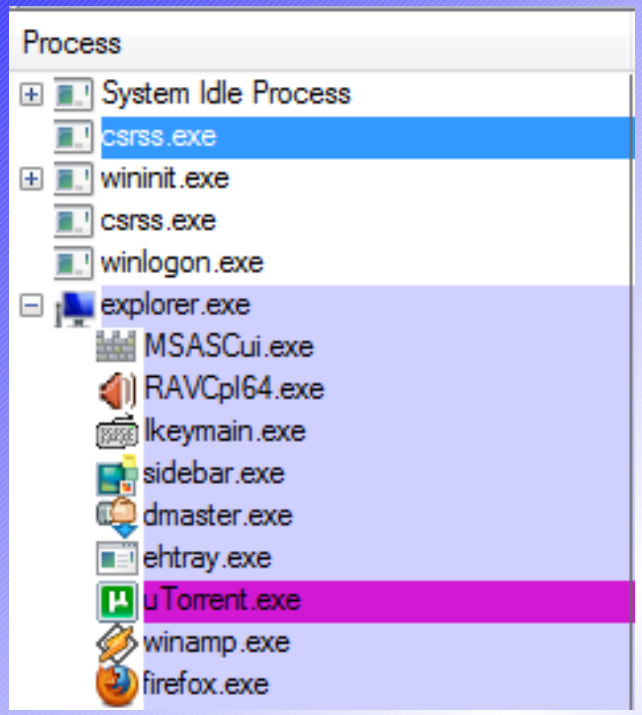
Обособленные дочерние процессы

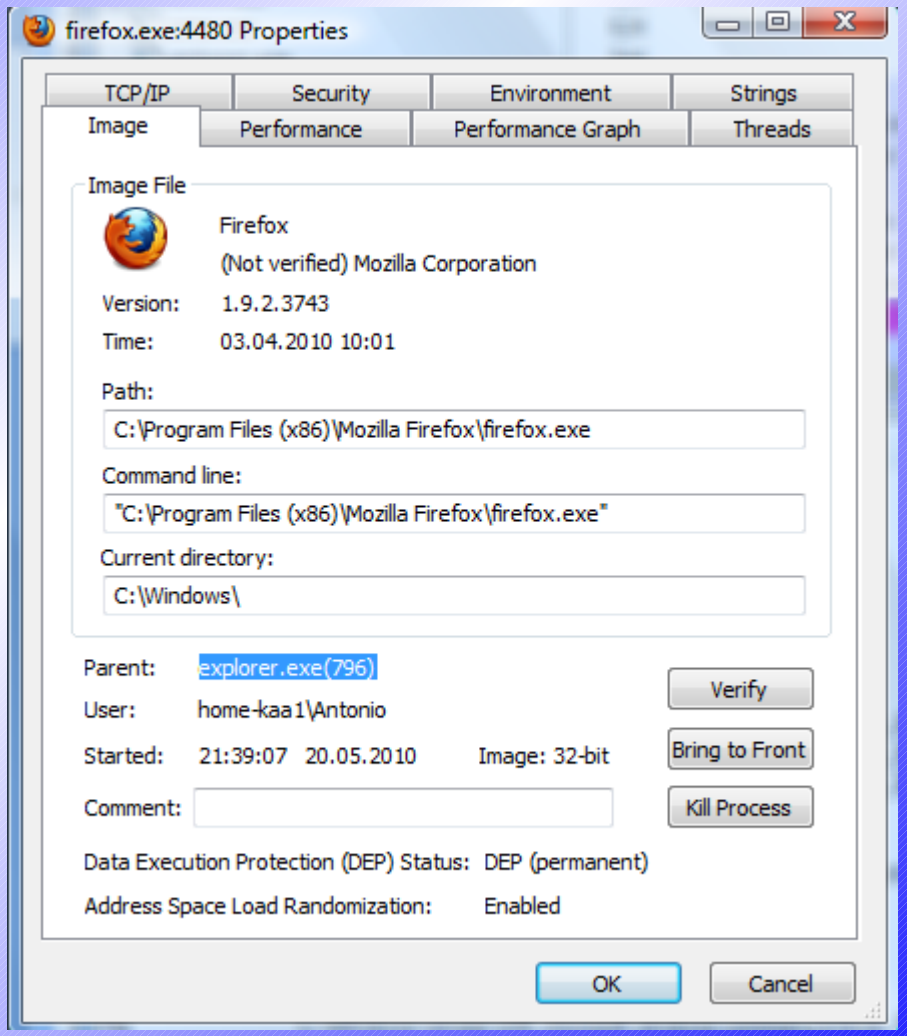
- Обособленный дочерний процесс – процесс, не передающий данные родительскому процессу, а только использующий предоставленные ресурсы.
- Адресные пространства родительского и дочернего процессов изначально различны.

Создание новых процессов

- Новые процессы ответвляются от уже существующих в системе процессов.
- Текущий процесс выполняет системный вызов, приводящий к созданию нового процесса. При этом созданный процесс представляет собой копию исходного процесса и его контекста.
- У нового процесса (дочернего) свой идентификатор, а родителем для него является запустивший процесс.

Создание новых процессов





Наследуемые свойства

- Дескрипторы к файлам, устройствам, средствам межпроцессного взаимодействия.
- Открытые дескрипторы процесса, потока, мьютекса, события, семафора, именованного канала, неименованного канала и отображаемых в памяти объектов.
- Переменные окружения.
- Текущий каталог и др.

Обмен сообщениями

Причины наличия в ОС обмена сообщениями

- Совместное использование данных (различные процессы могут работать с одной базой данных или с разделяемым файлом, совместно изменяя их содержимое).
- Модульная конструкция системы (процессы в микроядерной системе взаимодействуют друг с другом за счёт передачи сообщений через микроядро.
- Вычисления, при которых выходная информация одного процесса является входной для другого.

Реализация обмена сообщениями

- В рамках процесса: потоки разделяют адресное пространство процесса, поэтому для обмена сообщениями достаточно сохранить информацию в место, определённое программистом.
- Между различными процессами: использование возможностей и средств межпроцессного взаимодействия, предоставленных операционной системой.

Способы межпроцессного обмена сообщениями

- Переменные окружения.
- Разделяемая память.
- Сигналы.
- Каналы – именованные и неименованные.
- Сокеты – предназначены для передачи информации между процессами по сети.

Характеристики способов обмена сообщениями

- Направление связи. Однонаправленная (симплексная); двунаправленная с поочередной передачей в разных направлениях (полудуплексная); двунаправленная с возможностью одновременной передачи в разных направлениях (дуплексная).
- Синхронность обмена данными. Синхронный – отправитель блокируется до получения этого сообщения адресатом; асинхронный – отправитель не блокируется.

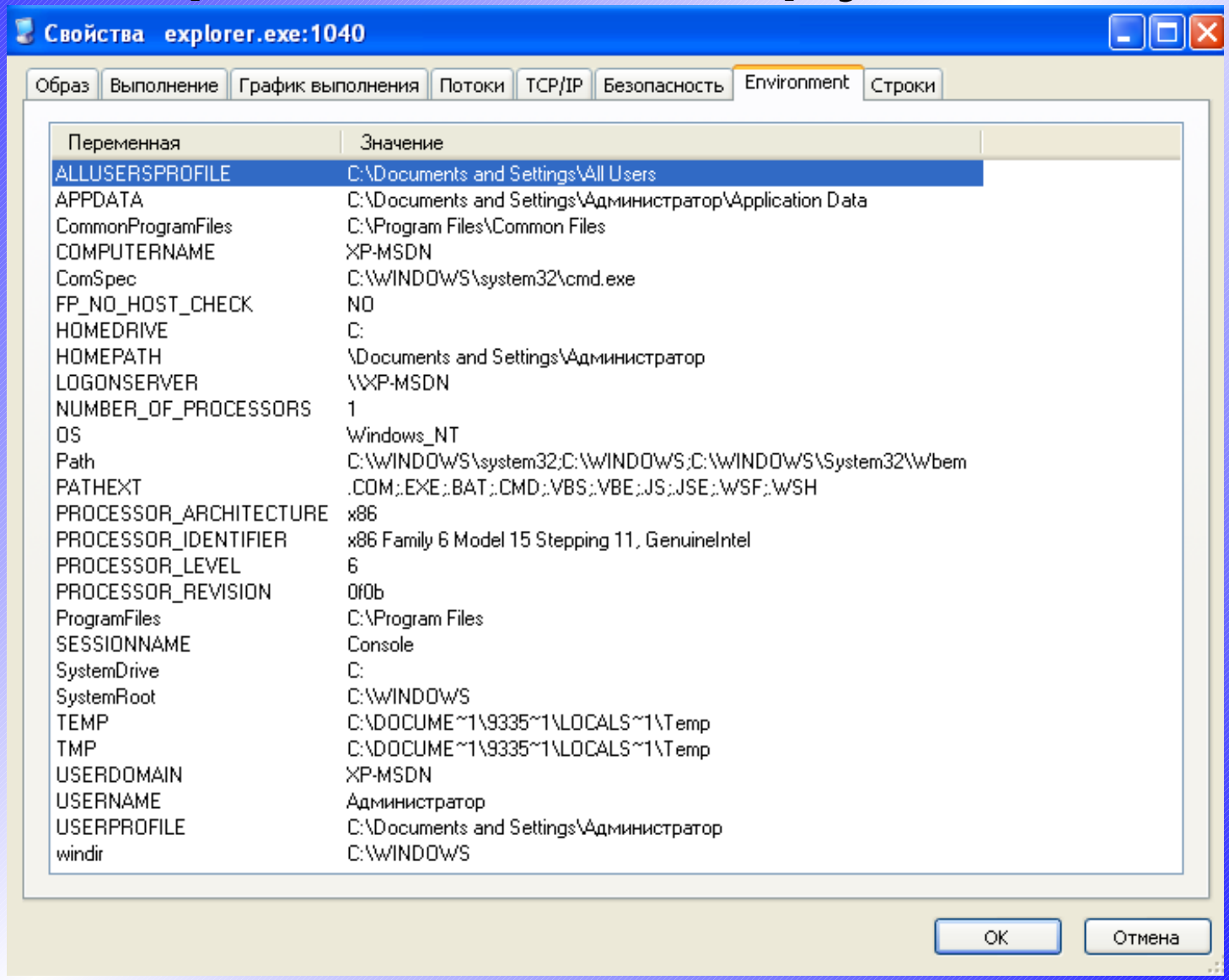
Характеристики способов обмена сообщениями

- Задержка при передаче информации. Отправителю необходимо учитывать: объём передаваемой информации и сведения о размере буфера (нулевой или отсутствует; ограниченный размер, неограниченный размер).
- Тип адресации (указание адресата). Прямая адресация – отправка информации получателю (с явным указанием имени получателя); непрямая адресация – информация помещается в промежуточный объект с собственным адресом (получатель - произвольный).

Переменные окружения

- Переменные окружения – именованные хранилища данных, в которые можно записывать любую текстовую информацию (присваивать значение переменной окружения), а впоследствии эту информацию считывать.
- Переменные окружения наследуются дочерним процессом от родительского.

Переменные окружения



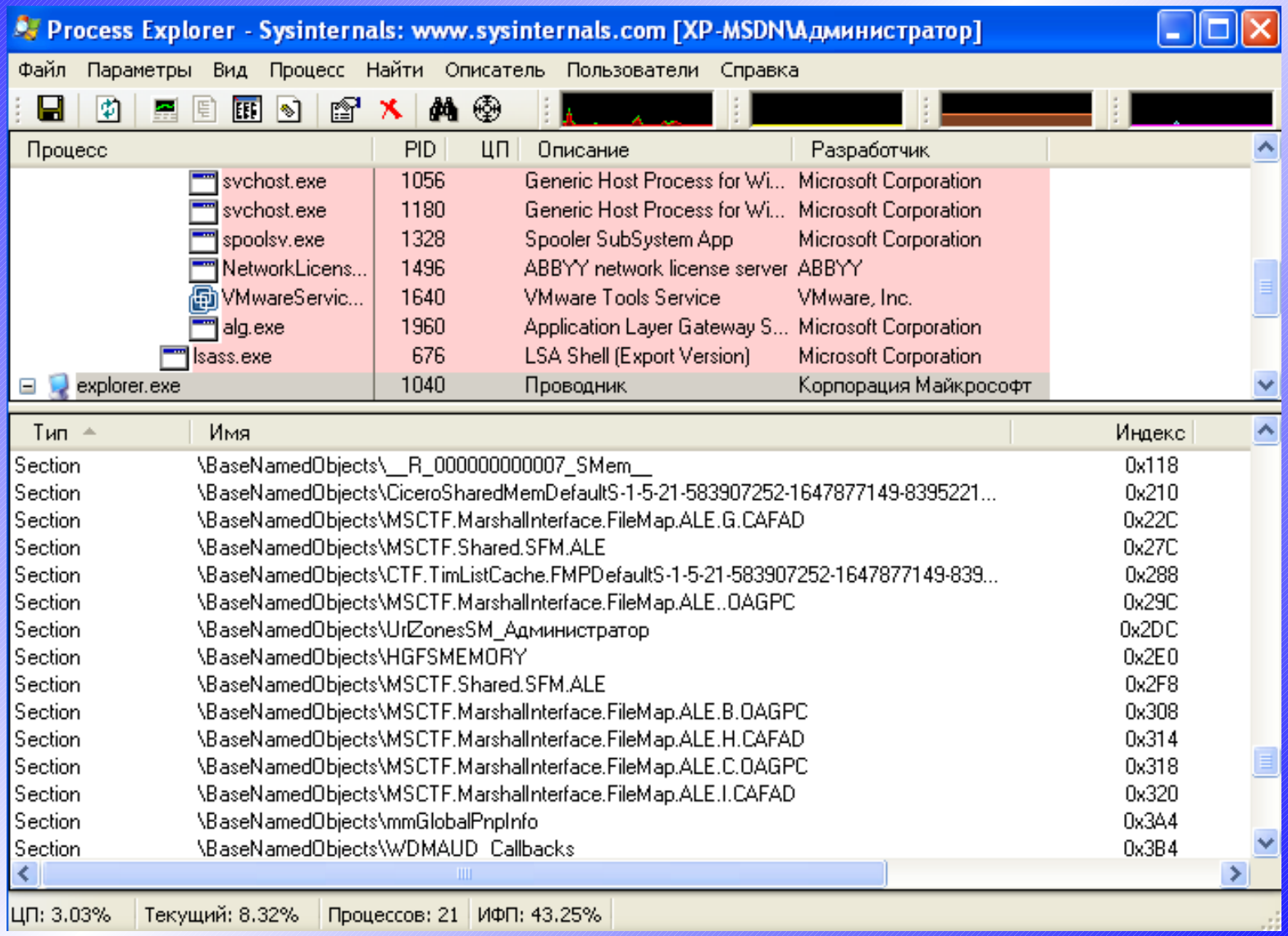
Недостатки переменных окружения

- Симплексная связь – дочерний процесс не может изменить окружение родительского процесса, а родительский не может воспользоваться окружением дочернего.
- Окружение уже запущенного процесса изменить извне нельзя.
- Через переменные окружения можно передавать только текстовые данные, обычно небольшого объёма.

Разделяемая память

- Два или более процессов могут совместно использовать некоторую область физической памяти каждый через своё адресное пространство.
- Созданием разделяемой памяти занимается операционная система.
- Возможность обмена информацией и влияние на другой процесс максимальны.
- Разделяемая память представляет собой наиболее быстрый способ взаимодействия процессов в одной вычислительной системе.

Разделяемая память



Недостатки разделяемой памяти

- Программы должны изначально содержать необходимый код (оба процесса должны обозначить участки своих адресных пространств как разделяемые.
- Разделяемая память не содержит средств синхронизации (за согласованную работу с разделямой памятью отвечает программист).

Каналы

- Каналы предоставляются операционной системой и позволяют передавать данные между процессами в порядке поступления ("первым пришел - первым вышел"), а также синхронизировать выполнение процессов.
- Каналы делают возможным взаимодействие с неизвестными отправителю процессами.
- Традиционная реализация каналов использует файловую систему для хранения данных. Операции с каналами – открытие/закрытие, чтение/запись и др.

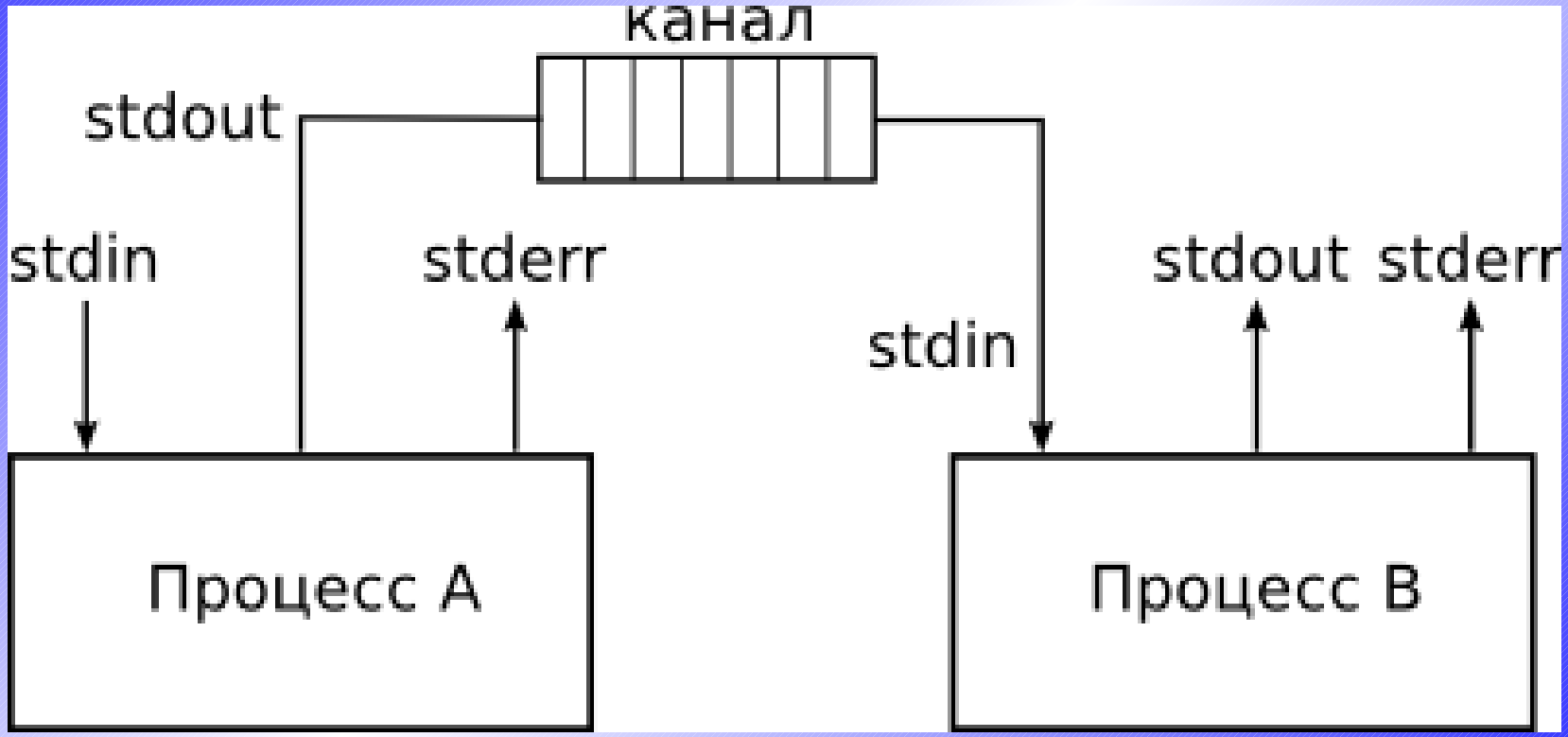
Способы передачи данных по каналам связи

- Каналы связи между процессами можно представить в виде специализированных файлов, которые не хранят информацию, а накапливают её до следующей операции чтения из канала другим процессом, образуя очередь.
- Потоковый ввод-вывод – данные представляют собой поток байтов без интерпретации со стороны системы.
- Обмен типизированными сообщениями – процессы придают передаваемым данным определённую структуру (границы между сообщениями, информацию об отправителе и др.)

Неименованные каналы

- Неименованные каналы – полудуплексное средство потоковой передачи байтов между родственными процессами.
- Функционируют в пределах локальной вычислительной системы и используются для перенаправления выходного потока одной программы на вход другой (конвейер).
- Использовать неименованные каналы могут только потомки процесса, создавшего неименованный канал.
- Неименованные каналы – временные.

Работа с неименованными каналами



Именованные каналы

- Именованные каналы – объекты ядра, позволяющие организовать межпроцессный обмен не только в изолированной вычислительной системе, но и в локальной сети.
- Обеспечивают дуплексную связь и позволяют использовать как потоковый способ, так и способ обмена сообщениями.
- Обмен данными может быть синхронным и асинхронным.
- Именованные каналы имеют собственное имя в системе и существуют постоянно.

Сигналы

- Сигналы используются в UNIX-подобных системах, как правило, для уведомления процесса о наступлении какого-либо события (не могут использоваться для передачи информации).
- Адресат должен знать, что означает полученный сигнал, надо ли на него реагировать и каким образом.

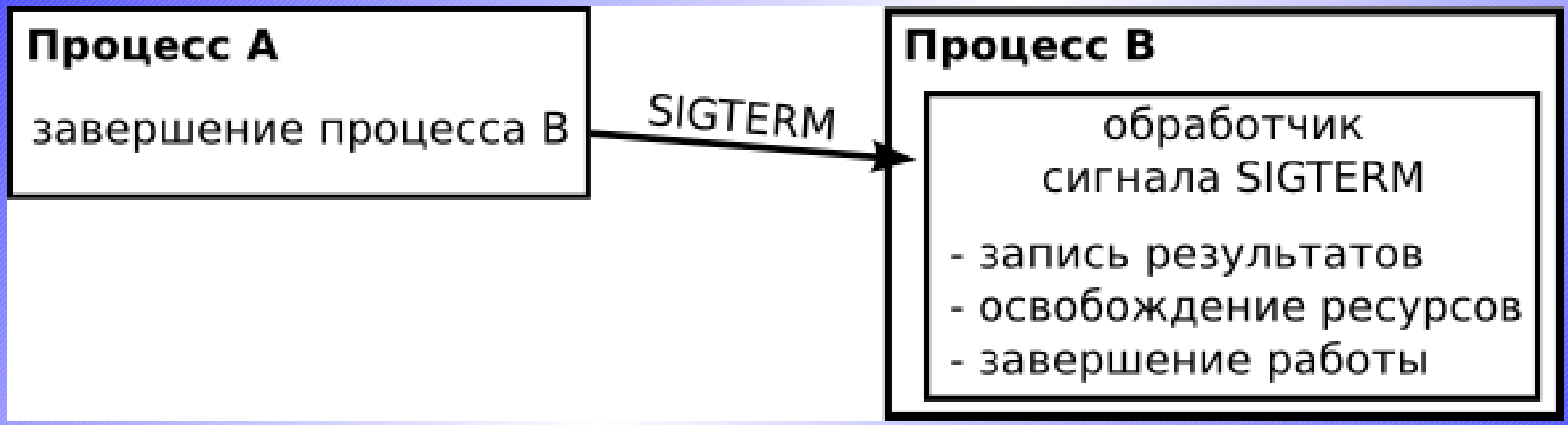
Сигналы

- При получении сигнала исполнение процесса прерывается и запускается специальная подпрограмма – обработчик сигнала (может быть определён ОС).
- У сигнала есть только одна характеристика, несущая информацию – его номер (целое число). Сигналы – это заранее определённый и пронумерованный список сообщений.
- Список сигналов и их имён стандартизован.

Примеры сигналов

- SIGHUP (1) – сигнал закрытия терминала, к которому привязан данный процесс. Обычно отправляется операционной системой всем процессам, запущенным из командной строки при завершении сеанса пользователя.
- SIGFPE (8) – сигнал ошибки в вычислениях с плавающей точкой. Отправляется операционной системой при некорректном исполнении программы.
- SIGCHLD – сигнал отправляется родительскому процессу в случае завершения его дочернего процесса.

Использование сигналов



SIGTERM (15) – сигнал завершения процесса. Как правило используется для корректного завершения его работы.

Рассмотренные вопросы

- Тупиковые ситуации.
- Алгоритмы обнаружения и исключения тупиков.
- Наследование ресурсов при создании процессов.
- Обмен сообщениями – способы межпроцессного взаимодействия.

Всем спасибо – все свободны,

если нет вопросов