Операционные среды и системное программирование
Л.р.1. Управление процессами, потоками, нитями

Вычислительный процесс

Это системный объект (объект типа Process), представляющий собой выполняющуюся программу. Он включает в себя выделенное адресное пространство, виртуальную память, дескрипторы системных ресурсов, а также один или несколько потоков выполнения, для выполнения инструкций кода. Процессы изолированы друг от друга и взаимодействуют посредством межпроцессного взаимодействия, контролируемого системой безопасности Windows.

Атрибуты процесса.

1. Адресное пространство — выделенное виртуальное пространство памяти, которое позволяет процессу изолированно хранить данные и код. Каждый процесс получает собственное пространство памяти, что обеспечивает безопасность и предотвращает конфликты с другими процессами.

2. Исполняемый код — двоичный код, загруженный из исполняемого файла (например, .exe), который выполняется в процессах. В этом коде находится основная логика программы.

3. Ресурсы и дескрипторы — каждый процесс имеет дескрипторы для различных ресурсов, таких как файлы, сетевые соединения, события, каналы и другие системные объекты. Эти ресурсы позволяют процессу взаимодействовать с внешним миром и другими компонентами системы.

4. Потоки выполнения (Threads) — каждый процесс содержит как минимум один поток (main thread), через который выполняется код. Потоки делят адресное пространство и системные ресурсы процесса, что позволяет нескольким потокам выполнять задачи одновременно и эффективно.

5. Информация об окружении — это данные, включая переменные среды, текущий рабочий каталог и командные аргументы, которые процесс использует во время выполнения.

6. Состояния и приоритеты — процессы могут находиться в различных состояниях (создание, выполнение, ожидание, завершение) и иметь

приоритеты, которые определяют порядок их выполнения. Операционная система использует эту информацию для управления выполнением процессов, распределения ресурсов и приоритизации задач.

7. Права доступа — каждый процесс имеет собственные права доступа и маркер безопасности, который определяет, какие действия он может выполнять в системе. Windows использует эти атрибуты безопасности для контроля доступа к системным ресурсам.

Образ процесса.

Образ процесса (Process Image) — это набор данных и кода, необходимых для выполнения процесса. Он создаётся операционной системой и загружается в память при запуске программы. Образ процесса включает в себя все ресурсы, которые нужны процессу для выполнения.

Содержит:

1. Исполняемый код программы
2. Глобальные и статические переменные, которые используются программой.
3. Сегмент кучи , то есть динамически выделяемая память, используемая во время выполнения процесса для хранения объектов и структур данных, размер которых изменяется.
4. Сегмент стека
5. Сегмент памяти под исполняемые модули и библиотеки
6. Таблицы дескрипторов файлов и ресурсов
7. Переменные окружения и параметры командной строки, передаваемые процессу при запуске.

Адресное пространство

Адресное пространство — это набор виртуальных адресов, которые операционная система выделяет каждому процессу. В среде Windows адресное пространство является изолированным и уникальным для каждого процесса, что способствует безопасности и стабильности системы. Процесс не имеет доступа к адресному пространству других процессов напрямую, что предотвращает случайные или намеренные вмешательства в работу других программ.

Виртуальная память: В Windows каждое 32-битное приложение получает 4 ГБ виртуального адресного пространства. 64-битные приложения могут адресовать гораздо больше — до терабайт памяти.

Отображение в физическую память: Виртуальная память не соответствует напрямую физической памяти (оперативной). С помощью механизма виртуальной памяти Windows подкачивает данные между физической памятью и файлом подкачки на диске, что позволяет программе работать с объёмом данных, превышающим объём физической памяти.

Файл подкачки (swap file) — это специальный файл на жёстком диске или SSD, который операционная система использует для расширения объёма доступной оперативной памяти (RAM). Когда физической памяти недостаточно для размещения всех активных процессов и данных, система временно перемещает менее часто используемые данные из оперативной памяти в файл подкачки. Этот процесс называется подкачкой.

Пространство дескрипторов.

Пространство дескрипторов — это набор уникальных идентификаторов, которые процесс использует для доступа к системным объектам и ресурсам. Дескрипторы — это маркеры, с помощью которых процесс взаимодействует с системными ресурсами, такими как файлы, потоки, события, мьютексы и другие объекты ядра.

Каждый процесс имеет своё собственное пространство дескрипторов. Это означает, что дескрипторы, доступные одному процессу, недоступны другим процессам, если они не переданы специально.

Состояния процесса.

1. New (Создание)

В этом состоянии процесс только что создан и подготавливается к запуску. В процессе создания операционная система выделяет необходимые ресурсы, такие как адресное пространство и место для хранения информации о состоянии процесса.

2. Ready (Готовность)

Процесс в состоянии готовности полностью подготовлен к исполнению, но не выполняется, так как процессор занят другими задачами. Процесс может быть перемещён из этого состояния в состояние выполнения, когда процессор освободится.

3. Running (Выполнение)

Это состояние, когда процесс исполняется на процессоре. Только один процесс на каждом процессоре может находиться в этом состоянии. Если процесс завершит свои задачи или будет временно прерван для выполнения другого, он перейдёт в другое состояние.

4. Blocked/Waiting (Ожидание)

Процесс может перейти в состояние ожидания, если ему требуется ресурс, который в данный момент недоступен, например, ввод-вывод или завершение другого процесса. Как только нужный ресурс освобождается, процесс возвращается в состояние готовности.

5. Terminated (Завершение)

Процесс переходит в состояние завершения, когда он успешно выполнил все свои задачи или был принудительно остановлен. На этом этапе система освобождает все ресурсы, связанные с процессом, и завершает его выполнение.

6. Suspended (Приостановка): В этом состоянии процесс временно приостановлен и не может быть запущен до тех пор, пока не возобновится. Это состояние используется для экономии ресурсов.

7. Transition (Переход): Процесс ожидает завершения некоторых условий, прежде чем перейти в состояние готовности. Например, процесс может находиться в переходном состоянии, если ожидает загрузки нужных страниц в память.

Вычислительный поток

В Windows вычислительный поток, или просто поток, представляет собой основную легковесную единицу выполнения в процессе и является системным объектом, который управляет выполнением кода. Каждый поток обладает своим собственным контекстом выполнения, включая состояние, регистры, стек и пространство адресов, что позволяет выполнять несколько задач параллельно в рамках одного процесса

Атрибуты потока

1. Идентификатор потока — Уникальный идентификатор, присваиваемый каждому потоку в процессе.

2. Приоритет потока –- Значение, которое определяет относительную важность потока по сравнению с другими потоками в системе. Приоритеты могут варьироваться от низкого до высокого.

3. Состояние потока — Поток может находиться в различных состояниях, таких как выполняющийся, приостановленный, готовый или заблокированный.

4. Контекст выполнения — Содержит информацию о текущем состоянии выполнения потока, включая:

- указатель на стек
- регистры процессора
- данные, необходимые для продолжения выполнения после приостановки.

5. Обработка исключений — Потоки могут иметь свои собственные механизмы обработки исключений, позволяющие перехватывать и обрабатывать ошибки, возникающие в процессе выполнения.

Приоритеты потока

Уровни приоритета: В Windows приоритеты потоков делятся на несколько уровней. Эти уровни включают:

Динамические приоритеты — они могут изменяться в зависимости от загрузки системы и поведения потока. Это позволяет системе оптимизировать выполнение потоков в реальном времени.

Статические приоритеты — устанавливаются при создании потока и остаются фиксированными, если их не изменить программно.
Нормальные и специальные приоритеты: В Windows существует несколько категорий приоритетов, таких как:

Idle (0): Самый низкий приоритет. Потоки с этим приоритетом выполняются только тогда, когда нет других потоков, ожидающих выполнения.

Lowest, Below Normal, Normal, Above Normal, Highest: Промежуточные уровни приоритета.

Real-time: Самый высокий приоритет, который гарантирует, что поток будет выполняться до завершения, если не возникнут другие потоки с таким же приоритетом.

Состояния потока

1. *Ready* (Готовый)

В этом состоянии поток готов к выполнению, но в данный момент не выполняется, так как операционная система не выделила ему процессорное время. Поток находится в очереди на выполнение и ожидает, когда система предоставит ему ресурсы.

2. *Running* (Выполняющийся)
Когда поток получает управление процессором, он переходит в состояние выполнения. В этом состоянии поток выполняет свою задачу, пока не будет прерван или не завершит выполнение.

3. *Blocked / Waiting* (Заблокированный / Ожидающий)
Поток находится в этом состоянии, когда он ожидает, что произойдет какое-то событие, например, завершение операции ввода-вывода, получение данных из другого потока или освобождение ресурса (например, мьютекса). Поток не может продолжать выполнение до тех пор, пока не будет выполнено условие, на которое он ожидает.

4. *Suspended* (Приостановленный)
Это состояние означает, что поток был временно остановлен. В отличие от заблокированного состояния, поток может быть возобновлён по желанию приложения. Потоки могут быть приостановлены, например, для управления ресурсами или во время отладки.

5. *Terminated* (Завершённый)
Когда поток завершает свою работу, он переходит в состояние завершения. Поток может завершиться по естественным причинам (например, выполнение кода завершилось) или из-за ошибки. После завершения поток освобождает все ресурсы, связанные с его работой.

6. *Zombie* (Зомби)
Хотя это состояние не является официальным в Windows, можно упомянуть о нем в контексте потоков. Когда поток завершает выполнение, но информация о его состоянии еще хранится в системе (например, для получения кода завершения), он может считаться «зомби». Обычно эта информация удаляется, когда родительский процесс считывает статус завершенного потока.

Разделяемые адресное пространство и пространство дескрипторов.

Все потоки, принадлежащие одному и тому же процессу, разделяют одно и то же адресное пространство. Это означает, что они могут непосредственно обмениваться данными и обращаться к общим ресурсам без необходимости использования механизма межпроцессного взаимодействия.

Пространство дескрипторов в Windows используется для управления ресурсами, такими как файлы, семафоры и другие объекты синхронизации. Каждый поток в процессе имеет доступ к общему пространству дескрипторов.

Создание (порождение) процессов.

Создание нового процесса в Windows начинается с вызова функции API, такой как CreateProcess. Эта функция принимает несколько параметров, включая имя исполняемого файла, параметры командной строки и атрибуты безопасности.

При создании процесса операционная система создает копию образа процесса, который включает код и данные приложения, а также необходимые ресурсы, такие как дескрипторы файлов и системные ресурсы.

Операционная система выделяет ресурсы, необходимые для нового процесса, включая память, дескрипторы и системные объекты.

Windows создает новый контекст процесса, который включает информацию о состоянии процесса, таких как его идентификатор, приоритет, состояние и адресное пространство. Новый процесс начинает выполнение в своем собственном адресном пространстве.

После создания процесса Windows автоматически создает основной поток, который начинает выполнение кода процесса. Этот поток управляет выполнением и жизненным циклом процесса, а также может создавать дополнительные потоки по мере необходимости.

Когда процесс завершает свою работу, он вызывает функцию *ExitProcess*, которая освобождает все ресурсы, связанные с ним, и уведомляет операционную систему о завершении

Иерархия процессов: родительские (parent) и дочерние (child) процессы.

Родительский процесс — это процесс, который создает один или несколько дочерних процессов. Дочерний процесс — это процесс, который был создан родительским процессом. Каждый процесс в Windows имеет уникальный идентификатор (PID) и идентификатор родительского процесса (PPID), который указывает на процесс, его создавший. При создании дочернего процесса родительский процесс использует функцию API, такую как *CreateProcess*.

Дочерние процессы могут наследовать ресурсы и настройки от родительского процесса. Это упрощает взаимодействие между процессами и позволяет им делиться данными. Например, если родительский процесс открывает файл, дочерний процесс может получить доступ к этому файлу, если дескриптор был передан ему при создании.

Когда родительский процесс завершается, его дочерние процессы могут продолжать работать, но они становятся «сиротами» и могут быть присвоены системе (обычно процессу с идентификатором 1, называемому *init*. Дочерние процессы могут завершаться вместе с родительским процессом, если они настроены на это.

Операционная система учитывает иерархию процессов при диспетчеризации ресурсов и управлении их жизненным циклом.

Завершение процессов.

1. *Нормальное завершение* — Процесс завершает свою работу самостоятельно, когда выполняет все свои задачи. Это может произойти через команду *exit*, возврат значения из главной функции или вызов функции завершения, такой как *TerminateProcess*.

2. *Ошибка* — Процесс может завершиться из-за ошибки, такой как исключение, сбой при выполнении операций ввода-вывода или нехватка ресурсов.

3. *Завершение родительским процессом* — если родительский процесс завершает свою работу, дочерние процессы могут быть также завершены, в зависимости от настроек и управления их состоянием.

Код завершения.

Код 0: Указывает на успешное завершение процесса. Это означает, что процесс выполнил все свои задачи без ошибок.

Ненулевые коды: Означают, что процесс завершился с ошибкой. Значение кода может варьироваться, и каждое значение может представлять определенную ошибку или состояние, которое привело к завершению. Например, код 1 может указывать на общую ошибку, а другие коды могут отражать специфические проблемы, такие как ошибки ввода-вывода, нехватка памяти и т. д.

```
HANDLE hProcess = OpenProcess (PROCESS_QUERY_INFORMATION, FALSE,
processId);
DWORD exitCode;
if (GetExitCodeProcess (hProcess, &exitCode)) {}
CloseHandle(hProcess);
```

Создание (порождение) потоков, главный поток.

Для создания нового потока в Windows используется функция *CreateThread.*

Каждое приложение в Windows начинает выполнение с главного потока, который является первым потоком, созданным операционной системой для данного процесса. Главный поток управляет началом выполнения программы и отвечает за инициализацию и выполнение основного кода. Он также может создавать дополнительные потоки по мере необходимости.

Главный поток может управлять другими потоками, включая их создание, приостановку, возобновление и завершение. Например, главный поток может создать рабочие потоки для выполнения длительных операций, таких как обработка данных или взаимодействие с сетью, позволяя интерфейсу оставаться отзывчивым.

Диспетчирование потоков.

Диспетчирование потоков — это процесс, с помощью которого операционная система управляет выполнением потоков и распределяет ресурсы CPU между ними, обеспечивая многозадачность. Windows использует алгоритм диспетчеризации для определения, какой поток должен выполняться в какой момент времени, учитывая приоритеты, состояние и другие факторы

Приоритеты являются ключевыми для диспетчеризации потоков. Потоки с более высоким приоритетом будут выполняться раньше, чем потоки с более низким приоритетом. В Windows приоритеты могут быть отрегулированы с помощью функции *SetThreadPriority.*

Для безопасного взаимодействия между потоками и избежания гонок данных Windows предоставляет механизмы синхронизации, такие как мьютексы и семафоры.

Приостановка и возобновление потоков.

Приостановка потока — это процесс, при котором поток временно останавливает выполнение своих задач. В Windows это достигается с помощью функции *SuspendThread*

Возобновление потока — это процесс восстановления потока из приостановленного состояния. Для этого используется функция *ResumeThread*

Завершение потоков.

Нормальное завершение — поток может завершиться нормально, когда он достигает конца своей функции. В этом случае все ресурсы, связанные с потоком, автоматически освобождаются. Например, если поток выполняет задачу, и по завершении функции *ThreadProc* (функция, выполняемая потоком) поток выходит, он завершается нормально.

Принудительное завершение — если поток не может быть завершен нормально (например, из-за зависания), его можно принудительно завершить с помощью функции *TerminateThread*. Этот подход менее предпочтителен, так как он не позволяет потоку выполнить операции очистки и может привести к утечкам памяти или повреждению данных.

Код завершения.

Коды завершения: Эти коды могут быть определены пользователем или стандартными значениями (например, 0 для нормального завершения или 1 для ошибки). Код завершения можно установить с помощью функции *ExitThread*, которая принимает целочисленный аргумент, представляющий код завершения.

Получение кода завершения: Родительский процесс или другие потоки могут получить код завершения завершенного потока, используя функцию *GetExitCodeThread*. Эта функция позволяет определить, как завершился поток, и в дальнейшем принять необходимые меры.

Нити (Fiber), особенности их выполнения.

Фибры — это пользовательские потоки, которые работают внутри потока операционной системы. Они реализуют кооперативный подход к многозадачности. Позволяют многократное переключение контекста выполнения без необходимости системные функции для управления потоками. Это позволяет разработчикам создавать высокоэффективные приложения с низкими накладными расходами на создание и переключение контекста.

Переключение между нитями осуществляется только явно, по инициативе текущей нити, и только в пределах одного потока. Отдав управление, нить не «знает», когда оно будет ей возвращено. Поток – «контейнер» нитей – продолжает диспетчиризироваться наравне с другими потоками, по общим правилам. Для планировщика потоков наличие нитей не существенно.

Взаимосвязь нитей и потоков

Каждый поток может содержать одну или несколько нитей. В этом контексте нити представляют собой более легковесные единицы выполнения, которые существуют внутри потока. То есть, когда поток создается, он может порождать нити для выполнения дополнительных задач, что позволяет оптимизировать выполнение приложения.

Нити, работающие внутри одного потока, используют одно и то же адресное пространство. Это означает, что все нити потока могут обращаться к одним и тем же переменным и ресурсам, что облегчает обмен данными между ними. Однако это также требует внимательного управления доступом к этим ресурсам, чтобы избежать конфликтов.

Потоки обеспечивают параллельное выполнение, позволяя нескольким потокам выполняться одновременно на разных ядрах процессора. В то же время нити обеспечивают кооперативную многозадачность внутри потока, позволяя различным задачам эффективно выполнять свои работы, когда это необходимо.

Потоки управляются ядром ОС. Планировщик отвечает за переключение между потоками, что позволяет достичь параллелизма на уровне процессоров.

Нити управляются приложением. Разработчик сам решает, когда переключать контекст выполнения между нитями с помощью вызовов, таких как *SwitchToFiber*. Это даёт разработчикам больше контроля, но требует внимательного управления для предотвращения зависания.

Создание, завершение, переключение нитей

CreateFiber — используется для создания новой нити. Она принимает следующие параметры:

1. Размер стека для новой нити.
2. Указатель на функцию, которую будет выполнять нить.
3. Указатель на данные, передаваемые в функцию

Завершение нитей также находится под контролем приложения. Нить завершается автоматически, когда функция, ассоциированная с ней, завершает выполнение. Кроме того, нить может быть завершена вручную с помощью функции *DeleteFiber*

Переключение между нитями осуществляется с помощью функции *SwitchToFiber*. Эта функция позволяет временно прервать выполнение текущей нити и передать управление другой нити.

При переключении контекста сохраняются состояния регистров, а затем происходит переключение на новую нить. После завершения выполнения новой нити управление возвращается к предыдущей нити.

Программный интерфейс (API) для «базового» управления процессами, потоками и нитями.

Изложен выше

Задание, Job – группировка и совместное управление несколькими процессами.