1. ***Опишите алгоритм сборки программы с использованием CMake и Clang.***

**1. Установка необходимых инструментов**

Cmake и Clang можно установить через пакетный менеджер вот такой командой:

sudo apt install cmake clang

**2. Создание структуры проекта и создание CMakeLists.txt**

Создайте директорию вашего проекта и структуру файлов. Создать файл проекта, создать папку src где будет находиться код и файл CMakeLists.txt (конфигурационный файл, используемый системой сборки CMake, содержит инструкции и параметры, которые определяют, как должен быть собран проект)

**3. Написание кода**

Написать код в main.cpp и других файлах если необходимо

**4. Написание содержимого для CMakeLists.txt**

-------------------------------------------------------------------------------------------------------------------------

cmake\_minimum\_required(VERSION 3.10) #минимальная версия CMake

project(MyProject) #имя проекта

set(CMAKE\_CXX\_STANDARD 14) #установка стандарта C++

add\_executable(MyProject src/main.cpp) #указываем, где находится исходный код

-------------------------------------------------------------------------------------------------------------------------

**5. Генерация файлов сборки**

Открыть терминал, перейти в директорию проекта и выполнить следующие команды:

-- mkdir build – создание новой директории с именем build

-- cd build – переход в эту директорию

-- cmake .. – запуск CMake и указание использования CMakeLists.txt

**6. Сборка проекта**

После успешной генерации файлов сборки надо выполнить следующую команду:

cmake –build .

Эта команда скомпилирует проект, указывая настройки в CMakeLists.txt

1. ***Опишите алгоритм запуска программы через дочерний процесс в Windows.***
2. **Подготовка структуры для создания процесса**:
   * Используйте структуру STARTUPINFO для настройки параметров нового процесса.
3. **Создание процесса**:
   * Вызовите функцию CreateProcess, чтобы создать новый процесс.
4. **Ожидание завершения процесса**:
   * Используйте функцию WaitForSingleObject, чтобы дождаться завершения дочернего процесса.
5. **Закрытие дескрипторов**:
   * Закройте дескрипторы, связанные с процессом и его потоком.

**Подробное описание функций и параметров**

1. **STARTUPINFO**:

Структура, которая содержит информацию о процессе, который будет создан.

**Поля**:

1) cb: Размер структуры в байтах (устанавливается в sizeof(STARTUPINFO)).

2) dwFlags: Флаги, определяющие поведение нового процесса (например, STARTF\_USESHOWWINDOW для управления видимостью окна).

3) wShowWindow: Указывает, как должно отображаться окно (например, SW\_SHOW).

4) hStdInput, hStdOutput, hStdError: Дескрипторы для перенаправления стандартного ввода/вывода.

1. **CreateProcess**:

Функция для создания нового процесса и его первичного потока.

**Параметры**:

1) lpApplicationName: Указатель на строку, содержащую имя исполняемого файла. Если NULL, то используется lpCommandLine.

2) lpCommandLine: Указатель на строку с командой для запуска (например, "C:\\Path\\to\\program.exe").

3) lpProcessAttributes: Указатель на дескриптор безопасности для процесса (обычно NULL).

4) lpThreadAttributes: Указатель на дескриптор безопасности для потока (обычно NULL).

5) bInheritHandles: Логическое значение, указывающее, наследуются ли дескрипторы (обычно FALSE).

6) dwCreationFlags: Флаги создания (например, 0 для обычного процесса).

7) lpEnvironment: Указатель на новую область окружения (обычно NULL).

8) lpCurrentDirectory: Указатель на текущую директорию (обычно NULL).

9) lpStartupInfo: Указатель на структуру STARTUPINFO.

10) lpProcessInformation: Указатель на структуру PROCESS\_INFORMATION, где будут храниться идентификаторы процесса и потока.

1. **WaitForSingleObject**:
   * Функция, которая заставляет вызывающий поток ожидать, пока указанный объект не перейдет в состояние сигнала.
   * **Параметры**:
     + hHandle: Дескриптор объекта (например, дескриптор процесса).
     + dwMilliseconds: Время ожидания в миллисекундах (например, INFINITE для бесконечного ожидания).
2. **CloseHandle**:
   * Функция, которая закрывает дескриптор, освобождая связанные с ним ресурсы.
   * **Параметры**:
     + hObject: Дескриптор, который нужно закрыть (например, дескриптор процесса или потока).
3. ***Опишите алгоритм запуска программы через дочерний процесс в Linux.***

**1. Создание дочернего процесса**

Используется функция fork()

**2. Замена процесса**

В дочернем процессе вызвать одну из функций exec(), чтобы заменить его текущий образ на новый исполняемый файл

**3. Ожидание завершения дочернего процесса**

В родительском процессе использовать функцию wait() или waitpid(), чтобы дождаться завершения дочернего процесса.

**Подробное описание функций и параметров:**

**1. fork()**

*Что за функция:* функция для создания нового процесса, который является копией вызывающего

*Возвращаемое значение:* 0 – в дочернем процессе, отрицательное значение – ошибка (не удалось создать процесс), положительное значение – в родительском процессе, это идентификатор дочернего процесса (PID).

**2. exec()**

*Что за функция:* семейство функций для замены текущего процесса новым исполняемым файлом

Функции:

execl()

excv()

execle()

execve()

execvp()

*Возвращаемое значение:* если функция успешно выполняется, функция не возвращается. Если возникает ошибка, возвращается -1.

Параметры для execvp:

1) file – имя исполняемого файла

2) argv – массив строк (аргументы командной строки), заканчиваю

**3. wait() или waitpid()**

*Что за функция:* функции для ожидания завершения дочернего процесса

*Параметры:*

1) pid: идентификатор процесса, который нужно ожидать (или -1 для ожидания любого дочернего процесса).

2) status: указатель на целочисленную переменную, в которую будет записан статус завершения процесса.

3) options: дополнительные опции (обычно 0).

*Возвращаемое значение:* PID завершившегося процесса или -1 в случае ошибки

1. ***Опишите алгоритм установки связи между дочерним и родительским процессами по средствам анонимного канала в Windows.***

***1) Создание анонимного канала:***

Используйте функцию CreatePipe для создания канала.

**CreatePipe**: Создает анонимный канал.

* **Параметры**:
  + lpReadPipe: Указатель на дескриптор для чтения.
  + lpWritePipe: Указатель на дескриптор для записи.
  + lpPipeAttributes: Атрибуты безопасности (обычно NULL).
  + nSize: Размер буфера (обычно 0 для стандартного размера).
* **Возвращает**: TRUE при успешном создании, иначе FALSE.

***2) Создание дочернего процесса:***

Используйте функцию CreateProcess, передав дескрипторы канала.

**CreateProcess**: Создает новый процесс и его основной поток.

* **Параметры**:
  + lpApplicationName: Имя исполняемого файла (обычно NULL).
  + lpCommandLine: Командная строка для запуска процесса.
  + lpProcessAttributes: Атрибуты безопасности процесса (обычно NULL).
  + lpThreadAttributes: Атрибуты безопасности потока (обычно NULL).
  + bInheritHandles: Указывает, должны ли дескрипторы наследоваться (обычно TRUE для передачи дескрипторов канала).
  + dwCreationFlags: Флаги создания (обычно 0).
  + lpEnvironment: Указатель на окружение (обычно NULL).
  + lpCurrentDirectory: Рабочая директория (обычно NULL).
  + lpStartupInfo: Указатель на структуру STARTUPINFO, содержащую информацию о запуске.
  + lpProcessInformation: Указатель на структуру PROCESS\_INFORMATION, содержащую информацию о созданном процессе.

***3) Настройка стандартных потоков для дочернего процесса:***

Используйте SetHandleInformation, чтобы указать, что дескрипторы канала не должны наследоваться.

***4) Чтение и запись данных между процессами:***

Используйте функции ReadFile и WriteFile для обмена данными.

**WriteFile**: Записывает данные в файл или устройство.

* **Параметры**:
  + hFile: Дескриптор канала для записи.
  + lpBuffer: Указатель на буфер с данными.
  + nNumberOfBytesToWrite: Количество байт для записи.
  + lpNumberOfBytesWritten: Указатель на переменную для записи количества записанных байт.
  + lpOverlapped: Указатель на структуру для асинхронного выполнения (обычно NULL).

**ReadFile**: Читает данные из файла или устройства.

* **Параметры**:
  + hFile: Дескриптор канала для чтения.
  + lpBuffer: Указатель на буфер для хранения прочитанных данных.
  + nNumberOfBytesToRead: Количество байт для чтения.
  + lpNumberOfBytesRead: Указатель на переменную для записи количества прочитанных байт.
  + lpOverlapped: Указатель на структуру для асинхронного выполнения (обычно NULL).

***5) Закрытие дескрипторов:***

Закройте неиспользуемые дескрипторы канала после завершения работы.

1. ***Опишите алгоритм установки связи между двумя процессами по средствам именованного канала в Windows.***

***1) Создание именованного канала:***

Используйте функцию CreateNamedPipe для создания канала.

**CreateNamedPipe**: Создает именованный канал.

* **Параметры**:
  + lpName: Имя канала (должно начинаться с \\.\pipe\).
  + dwOpenMode: Режим открытия (например, PIPE\_ACCESS\_DUPLEX).
  + dwPipeMode: Режим канала (например, PIPE\_TYPE\_MESSAGE).
  + nMaxInstances: Максимальное количество экземпляров.
  + nOutBufferSize: Размер выходного буфера.
  + nInBufferSize: Размер входного буфера.
  + dwTimeout: Время ожидания (обычно 0).
  + lpSecurityAttributes: Атрибуты безопасности (обычно NULL).
* **Возвращает**: Дескриптор канала, или INVALID\_HANDLE\_VALUE в случае ошибки.

***2) Подключение к именованному каналу:***

Используйте функцию ConnectNamedPipe для ожидания соединения от клиента.

**ConnectNamedPipe**: Устанавливает соединение с клиентом.

* **Параметры**:
  + hNamedPipe: Дескриптор канала.
  + lpOverlapped: Указатель на структуру для асинхронного ввода-вывода (обычно NULL).
* **Возвращает**: TRUE при успешном подключении, или FALSE в случае ошибки.

***3) Открытие именованного канала в клиентском процессе:***

Используйте функцию CreateFile для открытия канала.

**CreateFile**: Открывает или создает именованный канал.

* **Параметры**:
  + lpFileName: Имя канала.
  + dwDesiredAccess: Желаемый доступ (например, GENERIC\_READ | GENERIC\_WRITE).
  + dwShareMode: Режим совместного доступа (обычно 0).
  + lpSecurityAttributes: Атрибуты безопасности (обычно NULL).
  + dwCreationDisposition: Диспозиция создания (например, OPEN\_EXISTING).
  + dwFlagsAndAttributes: Атрибуты файла (обычно 0).
  + hTemplateFile: Дескриптор шаблонного файла (обычно NULL).
* **Возвращает**: Дескриптор канала, или INVALID\_HANDLE\_VALUE в случае ошибки.

***4) Обмен данными через именованный канал:***

Используйте функции ReadFile и WriteFile для чтения и записи данных.

 **WriteFile**: Записывает данные в именованный канал.

* **Параметры**:
  + hFile: Дескриптор канала.
  + lpBuffer: Указатель на буфер с данными.
  + nNumberOfBytesToWrite: Количество байт для записи.
  + lpNumberOfBytesWritten: Указатель на переменную для хранения количества записанных байт.
  + lpOverlapped: Указатель на структуру для асинхронного ввода-вывода (обычно NULL).
* **Возвращает**: TRUE при успешной записи, или FALSE в случае ошибки.

 **ReadFile**: Читает данные из именованного канала.

* **Параметры**:
  + hFile: Дескриптор канала.
  + lpBuffer: Указатель на буфер для получения данных.
  + nNumberOfBytesToRead: Максимальное количество байт для чтения.
  + lpNumberOfBytesRead: Указатель на переменную для хранения количества прочитанных байт.
  + lpOverlapped: Указатель на структуру для асинхронного ввода-вывода (обычно NULL).
* **Возвращает**: TRUE при успешном чтении, или FALSE в случае ошибки.

***5) Закрытие дескрипторов:***

Используйте функцию CloseHandle для закрытия дескрипторов канала.

**CloseHandle**: Закрывает открытый дескриптор.

* **Параметры**:
  + hObject: Дескриптор, который нужно закрыть.
* **Возвращает**: TRUE при успешном закрытии, или FALSE в случае ошибки.

***6) Удаление именованного канала (по желанию):***

Используйте функцию DisconnectNamedPipe и DeleteFile для удаления канала.

 **DisconnectNamedPipe**: Отключает текущий экземпляр именованного канала.

* **Параметры**:
  + hNamedPipe: Дескриптор канала.
* **Возвращает**: TRUE при успешном отключении, или FALSE в случае ошибки.

 **DeleteFile**: Удаляет файл (в данном случае, именованный канал).

* **Параметры**:
  + lpFileName: Указатель на строку с именем файла.
* **Возвращает**: TRUE при успешном удалении, или FALSE в случае ошибки.

1. ***Опишите алгоритм установки связи между двумя процессами по средствам fifo-очереди в Linux.***

***1) Создание FIFO-очереди:***

Используйте функцию mkfifo для создания именованной очереди.

**mkfifo**: Создает именованную очередь (FIFO).

* **Параметры**:
  + pathname: Путь к FIFO.
  + mode: Режим доступа (например, 0666 для чтения и записи для всех).
* **Возвращает**: 0 при успешном создании, или -1 в случае ошибки.

***2) Открытие FIFO-очереди:***

Один процесс открывает FIFO на запись, а другой — на чтение, используя функцию open***.***

**open**: Открывает файл или устройство.

* **Параметры**:
  + pathname: Путь к FIFO.
  + flags: Флаги доступа (например, O\_WRONLY для записи или O\_RDONLY для чтения).
* **Возвращает**: Дескриптор файла при успешном открытии, или -1 в случае ошибки.

***3) Обмен данными через FIFO:***

Используйте функции write и read для записи и чтения данных.

 **write**: Записывает данные в файл или устройство.

* **Параметры**:
  + fd: Дескриптор файла.
  + buf: Указатель на буфер с данными.
  + count: Количество байтов для записи.
* **Возвращает**: Количество записанных байтов, или -1 в случае ошибки.

 **read**: Читает данные из файла или устройства.

* **Параметры**:
  + fd: Дескриптор файла.
  + buf: Указатель на буфер для получения данных.
  + count: Максимальное количество байтов для чтения.
* **Возвращает**: Количество прочитанных байтов, или -1 в случае ошибки.

***4) Закрытие FIFO-очереди:***

Закройте дескрипторы, используя функцию close.

**close**: Закрывает открытый дескриптор файла.

* **Параметры**:
  + fd: Дескриптор, который нужно закрыть.
* **Возвращает**: 0 при успешном закрытии, или -1 в случае ошибки.

***5) Удаление FIFO-очереди (по желанию):***

Используйте функцию unlink для удаления FIFO.

**unlink**: Удаляет файл или устройство.

* **Параметры**:
  + pathname: Путь к FIFO.
* **Возвращает**: 0 при успешном удалении, или -1 в случае ошибки.

**unlink**: Удаляет файл или устройство.

* **Параметры**:
  + pathname: Путь к FIFO.
* **Возвращает**: 0 при успешном удалении, или -1 в случае ошибки.

1. ***Опишите алгоритм работы с TLS в потоках средствами API в Windows.***
2. ***Определение области TLS:***

Используйте функцию TlsAlloc для выделения индекса TLS.

**TlsAlloc**: Выделяет индекс для TLS.

* **Возвращает**: Индекс TLS при успешном выделении, или TLS\_OUT\_OF\_INDEXES в случае ошибки.

1. ***Инициализация данных TLS:***

Используйте TlsSetValue для установки значения для текущего потока.

**TlsSetValue**: Устанавливает значение для текущего потока.

* **Параметры**:
  + dwTlsIndex: Индекс TLS, полученный от TlsAlloc.
  + lpTlsValue: Указатель на значение, которое нужно установить.
* **Возвращает**: Ненулевое значение при успешном выполнении, или 0 в случае ошибки.

1. ***Получение данных TLS:***

Используйте TlsGetValue для получения значения из области TLS.

**TlsGetValue**: Получает значение для текущего потока.

* **Параметры**:
  + dwTlsIndex: Индекс TLS, полученный от TlsAlloc.
* **Возвращает**: Указатель на значение, установленное для текущего потока, или NULL в случае ошибки.

1. ***Освобождение области TLS:***

Используйте TlsFree для освобождения выделенного индекса TLS.

**TlsFree**: Освобождает индекс TLS.

* **Параметры**:
  + dwTlsIndex: Индекс TLS, который нужно освободить.
* **Возвращает**: Ненулевое значение при успешном выполнении, или 0 в случае ошибки.

1. ***Создание потоков:***

Используйте CreateThread для создания потоков, которые будут использовать TLS.

**CreateThread**: Создает новый поток.

* **Параметры**:
  + lpThreadAttributes: Атрибуты безопасности (обычно NULL).
  + dwStackSize: Размер стека (обычно 0 для стандартного).
  + lpStartAddress: Указатель на функцию, выполняемую потоком.
  + lpParameter: Параметры для потока (обычно NULL).
  + dwCreationFlags: Флаги создания (обычно 0).
  + lpThreadId: Указатель на переменную для хранения идентификатора потока (обычно NULL).
* **Возвращает**: Дескриптор потока, или NULL в случае ошибки.

1. ***Синхронизация и завершение потоков:***

Используйте WaitForSingleObject и CloseHandle для ожидания завершения потоков и освобождения ресурсов.

 **WaitForSingleObject**: Ожидает завершения потока.

* **Параметры**:
  + hHandle: Дескриптор объекта (в данном случае, потока).
  + dwMilliseconds: Время ожидания в миллисекундах (обычно INFINITE для бесконечного ожидания).
* **Возвращает**: WAIT\_OBJECT\_0 при успешном завершении, или другой код в случае ошибки.

 **CloseHandle**: Закрывает дескриптор объекта.

* **Параметры**:
  + hObject: Дескриптор, который нужно закрыть.
* **Возвращает**: TRUE при успешном закрытии, или FALSE в случае ошибки.

1. ***Опишите алгоритм работы с TLS в потоках средствами POSIX API в Linux.***
2. ***Определение области TLS:***

Используйте pthread\_key\_create для создания ключа TLS.

**pthread\_key\_create**: Создает ключ для TLS.

* **Параметры**:
  + key: Указатель на переменную для хранения ключа.
  + destructor: Функция очистки, вызываемая при удалении ключа (обычно NULL).
* **Возвращает**: 0 при успешном создании, или номер ошибки в случае сбоя.

1. ***Инициализация данных TLS:***

Используйте pthread\_setspecific для установки значения для текущего потока.

**pthread\_setspecific**: Устанавливает значение для текущего потока.

* **Параметры**:
  + key: Ключ TLS, полученный от pthread\_key\_create.
  + value: Указатель на значение, которое нужно установить.
* **Возвращает**: 0 при успешном выполнении, или номер ошибки в случае сбоя.

***3)*** ***Получение данных TLS:***

Используйте pthread\_getspecific для получения значения из области TLS.

**pthread\_getspecific**: Получает значение для текущего потока.

* **Параметры**:
  + key: Ключ TLS, полученный от pthread\_key\_create.
* **Возвращает**: Указатель на значение, установленное для текущего потока, или NULL в случае ошибки.

***4)*** ***Освобождение ключа TLS:***

Используйте pthread\_key\_delete для удаления ключа TLS.

**pthread\_key\_delete**: Удаляет ключ TLS.

* **Параметры**:
  + key: Ключ TLS, который нужно удалить.
* **Возвращает**: 0 при успешном выполнении, или номер ошибки в случае сбоя.

***5) Создание потоков:*** Используйте pthread\_create для создания потоков, которые будут использовать TLS.

**pthread\_create**: Создает новый поток.

* **Параметры**:
  + thread: Указатель на переменную для хранения идентификатора потока.
  + attr: Атрибуты потока (обычно NULL для стандартных).
  + start\_routine: Указатель на функцию, выполняемую потоком.
  + arg: Параметры для передаваемого потоку (обычно NULL).
* **Возвращает**: 0 при успешном создании, или номер ошибки в случае сбоя.

***6) Синхронизация и завершение потоков:***

Используйте pthread\_join для ожидания завершения потоков.

**pthread\_join**: Ожидает завершения потока.

* **Параметры**:
  + thread: Идентификатор потока, который нужно дождаться.
  + retval: Указатель на переменную для хранения возвращаемого значения потока (обычно NULL).
* **Возвращает**: 0 при успешном завершении, или номер ошибки в случае сбоя.

1. ***Опишите алгоритм работы с виртуальной памятью в Windows при условии, что сначала память резервируется и только потом появляется необходимость её использования.***

***1) Резервирование памяти:***

Используйте функцию VirtualAlloc для резервирования области виртуальной памяти.

**VirtualAlloc**: Резервирует или зарезервирует и выделяет область виртуальной памяти.

* **Параметры**:
  + lpAddress: Адрес, по которому нужно зарезервировать память (обычно NULL для автоматического выбора).
  + dwSize: Размер области памяти в байтах.
  + flAllocationType: Тип выделения (например, MEM\_RESERVE для резервирования памяти).
  + flProtect: Защита памяти (например, PAGE\_READWRITE для чтения и записи).
* **Возвращает**: Указатель на зарезервированную область памяти, или NULL в случае ошибки.

***2) Использование зарезервированной памяти:***

Запишите данные в зарезервированную память или прочитайте из нее.

***3) Изменение прав доступа к памяти (если необходимо):***

Используйте функцию VirtualProtect для изменения прав доступа к зарезервированной области памяти.

**VirtualProtect**: Изменяет защиту страницы памяти.

* **Параметры**:
  + lpAddress: Адрес начала области памяти.
  + dwSize: Размер области памяти в байтах.
  + flNewProtect: Новая защита (например, PAGE\_READONLY для только чтения).
  + lpflOldProtect: Указатель на переменную для хранения старой защиты.
* **Возвращает**: TRUE при успешном изменении защиты, или FALSE в случае ошибки.

***4) Освобождение памяти:***

Используйте функцию VirtualFree для освобождения зарезервированной памяти.

**VirtualFree**: Освобождает или освобождает и освобождает зарезервированную область памяти.

* **Параметры**:
  + lpAddress: Адрес, который нужно освободить.
  + dwSize: Размер области памяти (обычно 0 для освобождения всей страницы).
  + dwFreeType: Тип освобождения (например, MEM\_RELEASE для освобождения всей зарезервированной области).
* **Возвращает**: TRUE при успешном освобождении, или FALSE в случае ошибки.

1. ***Опишите алгоритм работы с виртуальной памятью в Linux при условии, что сначала память резервируется и только потом появляется необходимость её использования.***

***1) Резервирование памяти:***

Используйте функцию mmap для резервирования области виртуальной памяти.

**mmap**: Резервирует область виртуальной памяти и отображает ее в адресное пространство процесса.

* **Параметры**:
  + addr: Адрес, по которому нужно зарезервировать память (обычно NULL для автоматического выбора).
  + length: Размер области памяти в байтах.
  + prot: Права доступа (например, PROT\_READ | PROT\_WRITE для чтения и записи).
  + flags: Флаги (например, MAP\_PRIVATE | MAP\_ANONYMOUS для анонимной памяти).
  + fd: Дескриптор файла (обычно -1 для анонимной памяти).
  + offset: Смещение в файле (обычно 0 для анонимной памяти).
* **Возвращает**: Указатель на зарезервированную область памяти, или MAP\_FAILED в случае ошибки.

***2) Использование зарезервированной памяти:***

Запишите данные в зарезервированную память или прочитайте из нее.

***3) Изменение прав доступа к памяти (если необходимо):***

Используйте функцию mprotect для изменения прав доступа к зарезервированной области памяти.

**mprotect**: Изменяет права доступа к зарезервированной области памяти.

* **Параметры**:
  + addr: Адрес начала области памяти.
  + length: Размер области памяти в байтах.
  + prot: Новые права доступа (например, PROT\_READ для только чтения).
* **Возвращает**: 0 при успешном изменении прав доступа, или -1 в случае ошибки.

***4) Освобождение памяти:***

Используйте функцию munmap для освобождения зарезервированной памяти.

**munmap**: Освобождает ранее зарезервированную область памяти.

* **Параметры**:
  + addr: Адрес начала области памяти.
  + length: Размер области памяти в байтах.
* **Возвращает**: 0 при успешном освобождении, или -1 в случае ошибки.

1. ***Опишите алгоритм работы с пользовательской кучей в Windows.***

***1) Создание кучи:***

Используйте функцию HeapCreate для создания новой кучи.

**HeapCreate**: Создает новую кучу.

* **Параметры**:
  + flOptions: Флаги, определяющие поведение кучи (например, HEAP\_GENERATE\_EXCEPTIONS для генерации исключений при ошибках).
  + dwInitialSize: Начальный размер кучи в байтах (обычно 0).
  + dwMaximumSize: Максимальный размер кучи в байтах (обычно 0 для неограниченного размера).
* **Возвращает**: Дескриптор кучи, или NULL в случае ошибки.

***2) Выделение памяти из кучи:***

Используйте функцию HeapAlloc для выделения блока памяти.

**HeapAlloc**: Выделяет блок памяти из кучи.

* **Параметры**:
  + hHeap: Дескриптор кучи.
  + dwFlags: Флаги (например, HEAP\_ZERO\_MEMORY для инициализации памяти нулями).
  + dwBytes: Количество байт для выделения.
* **Возвращает**: Указатель на выделенный блок памяти, или NULL в случае ошибки.

***3) Работа с выделенной памятью:***

Выполните необходимые операции с выделенной памятью.

***4) Освобождение памяти:***

Используйте функцию HeapFree для освобождения выделенной памяти.

**HeapFree**: Освобождает блок памяти, выделенный из кучи.

* **Параметры**:
  + hHeap: Дескриптор кучи.
  + dwFlags: Флаги (обычно 0).
  + lpMem: Указатель на блок памяти, который нужно освободить.
* **Возвращает**: TRUE при успешном освобождении, или FALSE в случае ошибки.

***5) Уничтожение кучи:***

Используйте функцию HeapDestroy для уничтожения кучи после завершения работы.

**HeapDestroy**: Уничтожает кучу и освобождает все ресурсы, связанные с ней.

* **Параметры**:
  + hHeap: Дескриптор кучи.
* **Возвращает**: TRUE при успешном уничтожении, или FALSE в случае ошибки.

1. ***Опишите алгоритм получения информации о файле в Windows.***

***1) Получение атрибутов файла:***

Используйте функцию GetFileAttributes для получения атрибутов файла.

**GetFileAttributes**: Получает атрибуты файла или директории.

* **Параметры**:
  + lpFileName: Указатель на строку, содержащую имя файла или директории.
* **Возвращает**: Атрибуты файла (например, FILE\_ATTRIBUTE\_NORMAL), или INVALID\_FILE\_ATTRIBUTES в случае ошибки.

***2) Получение размера файла:***

Используйте функцию GetFileSize для получения размера файла.

**CreateFile**: Открывает или создает файл.

* **Параметры**:
  + lpFileName: Имя файла для открытия.
  + dwDesiredAccess: Желаемый доступ (например, GENERIC\_READ).
  + dwShareMode: Режим совместного доступа (обычно 0 для исключительного доступа).
  + lpSecurityAttributes: Атрибуты безопасности (обычно NULL).
  + dwCreationDisposition: Диспозиция создания (например, OPEN\_EXISTING).
  + dwFlagsAndAttributes: Атрибуты файла (например, FILE\_ATTRIBUTE\_NORMAL).
  + hTemplateFile: Дескриптор шаблонного файла (обычно NULL).
* **Возвращает**: Дескриптор файла, или INVALID\_HANDLE\_VALUE в случае ошибки.

**GetFileSize**: Получает размер указанного файла.

* **Параметры**:
  + hFile: Дескриптор файла.
  + lpFileSizeHigh: Указатель на переменную для хранения верхних 32 бит (обычно NULL).
* **Возвращает**: Размер файла в байтах, или INVALID\_FILE\_SIZE в случае ошибки.

***3)Получение времени создания, последнего доступа и изменения файла:*** Используйте функцию GetFileTime для получения временных меток файла.

**GetFileTime**: Получает временные метки для указанного файла.

* **Параметры**:
  + hFile: Дескриптор файла.
  + lpCreationTime: Указатель на переменную для хранения времени создания.
  + lpLastAccessTime: Указатель на переменную для хранения времени последнего доступа.
  + lpLastWriteTime: Указатель на переменную для хранения времени последней записи.
* **Возвращает**: TRUE при успешном получении времени, или FALSE в случае ошибки.

**FileTimeToSystemTime**: Преобразует FILETIME в структуру SYSTEMTIME.

* **Параметры**:
  + lpFileTime: Указатель на структуру FILETIME.
  + lpSystemTime: Указатель на структуру SYSTEMTIME, куда будет записано результат.
* **Возвращает**: TRUE при успешном преобразовании, или FALSE в случае ошибки.

***4) Закрытие дескриптора файла (если был открыт).***

**CloseHandle**: Закрывает открытый дескриптор.

* **Параметры**:
  + hObject: Дескриптор, который нужно закрыть.
* **Возвращает**: TRUE при успешном закрытии, или FALSE в случае ошибки.

1. ***Опишите алгоритм добавления строки в файл через WinAPI.***

***1) Открытие файла:***

Используйте функцию CreateFile для открытия файла в режиме добавления.

**CreateFile**: Открывает или создает файл.

* **Параметры**:
  + lpFileName: Имя файла для открытия.
  + dwDesiredAccess: Желаемый доступ (например, GENERIC\_WRITE для записи).
  + dwShareMode: Режим совместного доступа (обычно 0 для исключительного доступа).
  + lpSecurityAttributes: Атрибуты безопасности (обычно NULL).
  + dwCreationDisposition: Диспозиция создания (например, OPEN\_ALWAYS для открытия файла, если он существует, или создания нового).
  + dwFlagsAndAttributes: Атрибуты файла (например, FILE\_ATTRIBUTE\_NORMAL).
  + hTemplateFile: Дескриптор шаблонного файла (обычно NULL).
* **Возвращает**: Дескриптор файла, или INVALID\_HANDLE\_VALUE в случае ошибки.

***2) Перемещение указателя на конец файла:***

Если файл открыт в режиме добавления, указатель автоматически перемещается на конец файла.

**SetFilePointer**: Устанавливает текущую позицию указателя в файле.

* **Параметры**:
  + hFile: Дескриптор файла.
  + lDistanceToMove: Смещение (0 для конца файла).
  + lpDistanceToMoveHigh: Указатель на верхние 32 бита (обычно NULL).
  + dwMoveMethod: Метод перемещения (например, FILE\_END для конца файла).
* **Возвращает**: Новую позицию указателя в файле, или INVALID\_SET\_FILE\_POINTER в случае ошибки.

***3) Запись строки в файл:***

Используйте функцию WriteFile для записи новой строки.

**WriteFile**: Записывает данные в файл.

* **Параметры**:
  + hFile: Дескриптор файла.
  + lpBuffer: Указатель на буфер с данными.
  + nNumberOfBytesToWrite: Количество байт для записи.
  + lpNumberOfBytesWritten: Указатель на переменную для хранения количества записанных байт.
  + lpOverlapped: Указатель на структуру для асинхронного ввода-вывода (обычно NULL).
* **Возвращает**: TRUE при успешной записи, или FALSE в случае ошибки.

***4) Закрытие файла:***

Используйте функцию CloseHandle для закрытия дескриптора файла.

**CloseHandle**: Закрывает открытый дескриптор.

* **Параметры**:
  + hObject: Дескриптор, который нужно закрыть.
* **Возвращает**: TRUE при успешном закрытии, или FALSE в случае ошибки.

1. ***Опишите алгоритм удаления строки в файл через WinAPI.***

***1) Открытие файла:***

Используйте функцию CreateFile для открытия файла.

**CreateFile**: Открывает или создает файл.

* **Параметры**:
  + lpFileName: Имя файла для открытия.
  + dwDesiredAccess: Желаемый доступ (например, GENERIC\_READ | GENERIC\_WRITE).
  + dwShareMode: Режим совместного доступа (обычно 0 для исключительного доступа).
  + lpSecurityAttributes: Атрибуты безопасности (обычно NULL).
  + dwCreationDisposition: Диспозиция создания (например, OPEN\_EXISTING для открытия существующего файла).
  + dwFlagsAndAttributes: Атрибуты файла (например, FILE\_ATTRIBUTE\_NORMAL).
  + hTemplateFile: Дескриптор шаблонного файла (обычно NULL).
* **Возвращает**: Дескриптор файла, или INVALID\_HANDLE\_VALUE в случае ошибки.

***2) Получение размера файла:***

Используйте функцию GetFileSize для получения размера файла.

**GetFileSize**: Получает размер указанного файла.

* **Параметры**:
  + hFile: Дескриптор файла.
  + lpFileSizeHigh: Указатель на переменную для хранения верхних 32 бит (обычно NULL).
* **Возвращает**: Размер файла в байтах, или INVALID\_FILE\_SIZE в случае ошибки.

***3) Чтение содержимого файла:***

Используйте функцию ReadFile для загрузки данных из файла.

**ReadFile**: Читает данные из файла.

* **Параметры**:
  + hFile: Дескриптор файла.
  + lpBuffer: Буфер, в который будут записаны данные.
  + nNumberOfBytesToRead: Количество байт для чтения.
  + lpNumberOfBytesRead: Указатель на переменную для хранения количества прочитанных байт.
  + lpOverlapped: Указатель на структуру для асинхронного ввода-вывода (обычно NULL).
* **Возвращает**: TRUE при успешном чтении, или FALSE в случае ошибки.

***4) Поиск и удаление строки:***

Модифицируйте содержимое, удаляя нужную строку.

***5) Запись обратно в файл:***

Используйте функцию WriteFile для записи измененного содержимого обратно в файл.

S**etFilePointer**: Устанавливает текущую позицию указателя в файле.

* **Параметры**:
  + hFile: Дескриптор файла.
  + lDistanceToMove: Смещение (0 для начала файла).
  + lpDistanceToMoveHigh: Указатель на верхние 32 бита (обычно NULL).
  + dwMoveMethod: Метод перемещения (например, FILE\_BEGIN для начала файла).
* **Возвращает**: Новую позицию указателя в файле, или INVALID\_SET\_FILE\_POINTER в случае ошибки.

**WriteFile**: Записывает данные в файл.

* **Параметры**:
  + hFile: Дескриптор файла.
  + lpBuffer: Указатель на буфер с данными.
  + nNumberOfBytesToWrite: Количество байт для записи.
  + lpNumberOfBytesWritten: Указатель на переменную для хранения количества записанных байт.
  + lpOverlapped: Указатель на структуру для асинхронного ввода-вывода (обычно NULL).
* **Возвращает**: TRUE при успешной записи, или FALSE в случае ошибки.

**SetEndOfFile**: Устанавливает конец файла.

* **Параметры**:
  + hFile: Дескриптор файла.
* **Возвращает**: TRUE при успешном завершении, или FALSE в случае ошибки.

***6) Закрытие файла:***

Используйте функцию CloseHandle для закрытия дескриптора файла.

**CloseHandle**: Закрывает открытый дескриптор.

* **Параметры**:
  + hObject: Дескриптор, который нужно закрыть.
* **Возвращает**: TRUE при успешном закрытии, или FALSE в случае ошибки.

1. ***Опишите алгоритм добавления строки в файл через POSIX API.***

***1) Открытие файла:***

Используйте функцию open для открытия файла в режиме добавления.

**open**: Открывает файл.

* **Параметры**:
  + pathname: Имя файла для открытия.
  + flags: Флаги доступа (например, O\_WRONLY для записи, O\_APPEND для добавления данных в конец файла, O\_CREAT для создания файла, если он не существует).
  + mode: Режим доступа (например, S\_IRUSR | S\_IWUSR для разрешения на чтение и запись владельцу файла).
* **Возвращает**: Дескриптор файла, или -1 в случае ошибки.

***2)*** ***Перемещение указателя на конец файла:***

Если файл открыт в режиме добавления, указатель автоматически перемещается на конец файла.

**lseek**: Изменяет текущую позицию чтения/записи.

* **Параметры**:
  + fd: Дескриптор файла.
  + offset: Смещение (0 для конца файла).
  + whence: Откуда начинать (например, SEEK\_END для конца файла).
* **Возвращает**: Новую позицию в байтах, или -1 в случае ошибки.

***3) Запись строки в файл:***

Используйте функцию write для записи новой строки.

**write**: Записывает данные в файл.

* **Параметры**:
  + fd: Дескриптор файла.
  + buf: Указатель на буфер с данными.
  + count: Количество байт для записи.
* **Возвращает**: Количество записанных байт, или -1 в случае ошибки.

***4) Закрытие файла:***

Используйте функцию close для закрытия дескриптора файла.

**close**: Закрывает открытый дескриптор файла.

* **Параметры**:
  + fd: Дескриптор файла.
* **Возвращает**: 0 при успешном закрытии, или -1 в случае ошибки.

1. ***Опишите алгоритм удаления строки в файл через POSIX API.***

***1) Открытие файла:***

Используйте функцию open для открытия файла.

**open**: Открывает файл.

* **Параметры**:
  + pathname: Имя файла для открытия.
  + flags: Флаги доступа (например, O\_RDWR для чтения и записи).
* **Возвращает**: Дескриптор файла, или -1 в случае ошибки.

***2) Чтение содержимого файла:***

Используйте функцию read для загрузки данных из файла.

**read**: Читает данные из файла.

* **Параметры**:
  + fd: Дескриптор файла.
  + buf: Указатель на буфер для чтения.
  + count: Максимальное количество байт для чтения.
* **Возвращает**: Количество прочитанных байт, или -1 в случае ошибки.

***3) Поиск и удаление строки:***

Модифицируйте содержимое, удаляя нужную строку.

**strstr**: Находит первое вхождение подстроки в строке.

* **Параметры**:
  + haystack: Указатель на строку, в которой производится поиск.
  + needle: Указатель на подстроку для поиска.
* **Возвращает**: Указатель на первое вхождение подстроки или NULL, если не найдено.

**strchr**: Находит первое вхождение символа в строке.

* **Параметры**:
  + s: Указатель на строку.
  + c: Символ для поиска.
* **Возвращает**: Указатель на первое вхождение символа или NULL, если не найдено.

**memmove**: Копирует блоки памяти с учетом перекрытия.

* **Параметры**:
  + dest: Указатель на целевую область памяти.
  + src: Указатель на исходную область памяти.
  + n: Количество байт для копирования.

***4) Запись обратно в файл:***

Используйте функцию write для записи измененного содержимого обратно в файл.

**lseek**: Изменяет текущую позицию чтения/записи.

* **Параметры**:
  + fd: Дескриптор файла.
  + offset: Смещение (0 для начала файла).
  + whence: Откуда начинать (например, SEEK\_SET для начала файла).
* **Возвращает**: Новую позицию в байтах, или -1 в случае ошибки.

**write**: Записывает данные в файл.

* **Параметры**:
  + fd: Дескриптор файла.
  + buf: Указатель на буфер с данными.
  + count: Количество байт для записи.
* **Возвращает**: Количество записанных байт, или -1 в случае ошибки.

**ftruncate**: Обрезает файл до заданной длины.

* **Параметры**:
  + fd: Дескриптор файла.
  + length: Длина, до которой нужно обрезать файл.
* **Возвращает**: 0 при успехе, или -1 в случае ошибки.

***5) Закрытие файла:***

Используйте функцию close для закрытия дескриптора файла.

**close**: Закрывает открытый дескриптор файла.

* **Параметры**:
  + fd: Дескриптор файла.
* **Возвращает**: 0 при успешном закрытии, или -1 в случае ошибки.

1. ***Опишите алгоритм чтения строки файла, отображенного в память средствами WinAPI.***

***1) Открытие файла:***

Используйте функцию CreateFile для открытия файла.

**CreateFile**: Открывает или создает файл.

* **Параметры**:
  + lpFileName: Имя файла для открытия.
  + dwDesiredAccess: Желаемый доступ (например, GENERIC\_READ для чтения).
  + dwShareMode: Режим совместного доступа (например, 0 для исключительного доступа).
  + lpSecurityAttributes: Атрибуты безопасности (обычно NULL).
  + dwCreationDisposition: Диспозиция создания (например, OPEN\_EXISTING для открытия существующего файла).
  + dwFlagsAndAttributes: Атрибуты файла (например, FILE\_ATTRIBUTE\_NORMAL).
  + hTemplateFile: Дескриптор шаблонного файла (обычно NULL).
* **Возвращает**: Дескриптор файла, или INVALID\_HANDLE\_VALUE в случае ошибки.

***2) Получение размера файла:***

Используйте функцию GetFileSize для получения размера файла.

**GetFileSize**: Получает размер указанного файла.

* **Параметры**:
  + hFile: Дескриптор файла.
  + lpFileSizeHigh: Указатель на переменную для хранения верхних 32 бит (обычно NULL).
* **Возвращает**: Размер файла в байтах, или INVALID\_FILE\_SIZE в случае ошибки.

***3) Отображение файла в память:***

Используйте функции CreateFileMapping и MapViewOfFile для отображения файла в память.

**CreateFileMapping**: Создает объект отображения файла.

* **Параметры**:
  + hFile: Дескриптор файла.
  + lpFileMappingAttributes: Атрибуты безопасности (обычно NULL).
  + flProtect: Защита (например, PAGE\_READONLY).
  + dwMaximumSizeHigh: Верхние 32 бита максимального размера (обычно 0).
  + dwMaximumSizeLow: Нижние 32 бита максимального размера (обычно 0).
  + lpName: Имя объекта отображения (обычно NULL).
* **Возвращает**: Дескриптор объекта отображения, или NULL в случае ошибки.

**MapViewOfFile**: Отображает область файла в адресное пространство процесса.

* **Параметры**:
  + hFileMappingObject: Дескриптор объекта отображения.
  + dwDesiredAccess: Желаемый доступ (например, FILE\_MAP\_READ для чтения).
  + dwFileOffsetHigh: Верхние 32 бита смещения (обычно 0).
  + dwFileOffsetLow: Нижние 32 бита смещения (обычно 0).
  + dwNumberOfBytesToMap: Количество байт для отображения (обычно 0 для отображения всей области).
* **Возвращает**: Указатель на отображенную область, или NULL в случае ошибки.

***4) Чтение строки из отображенной памяти:***

Используйте стандартные функции для работы со строками, такие как strchr или strncpy.

**strchr**: Находит первое вхождение символа в строке.

* **Параметры**:
  + s: Указатель на строку.
  + c: Символ для поиска.
* **Возвращает**: Указатель на первое вхождение символа или NULL, если не найдено.

**strncpy**: Копирует заданное количество символов.

* **Параметры**:
  + dest: Указатель на целевой буфер.
  + src: Указатель на исходную строку.
  + n: Количество символов для копирования.

***5) Отмена отображения и закрытие файла:***

Используйте функции UnmapViewOfFile и CloseHandle для освобождения ресурсов.

**UnmapViewOfFile**: Отменяет отображение области памяти.

* **Параметры**:
  + lpBaseAddress: Указатель на область, которую нужно отменить.
* **Возвращает**: TRUE при успешном завершении, или FALSE в случае ошибки.

**CloseHandle**: Закрывает открытый дескриптор.

* **Параметры**:
  + hObject: Дескриптор, который нужно закрыть.
* **Возвращает**: TRUE при успешном закрытии, или FALSE в случае ошибки.

1. ***Опишите алгоритм чтения строки файла, отображенного в память средствами POSIX API.***

***1) Открытие файла:***

Используйте функцию open для открытия файла.

**open**: Открывает файл.

* **Параметры**:
  + pathname: Имя файла для открытия.
  + flags: Флаги доступа (O\_RDONLY для чтения).
* **Возвращает**: Дескриптор файла, или -1 в случае ошибки.

***2) Получение размера файла:***

Используйте функцию lseek для получения размера файла.

**lseek**: Изменяет текущую позицию чтения/записи.

* **Параметры**:
  + fd: Дескриптор файла.
  + offset: Смещение (0 для начала).
  + whence: Откуда начинать (например, SEEK\_END для конца файла).
* **Возвращает**: Новую позицию в байтах, или -1 в случае ошибки.

***3) Отображение файла в память:***

Используйте функцию mmap для отображения файла в память.

**mmap**: Отображает файлы или устройства в память.

* **Параметры**:
  + addr: Адрес, по которому нужно отобразить (обычно NULL для автоматического выбора).
  + length: Длина отображаемой области (размер файла).
  + prot: Защита (например, PROT\_READ для чтения).
  + flags: Флаги отображения (MAP\_PRIVATE для частного отображения).
  + fd: Дескриптор файла.
  + offset: Смещение в файле (обычно 0).
* **Возвращает**: Указатель на отображенную область или MAP\_FAILED в случае ошибки.

***4) Чтение строки из отображенной памяти:***

Используйте стандартные функции для работы со строками, такие как strchr или strncpy.

**strchr**: Находит первый вхождение символа в строке.

* **Параметры**:
  + s: Указатель на строку.
  + c: Символ для поиска.
* **Возвращает**: Указатель на первое вхождение символа или NULL, если не найдено.

**strncpy**: Копирует заданное количество символов.

* **Параметры**:
  + dest: Указатель на целевой буфер.
  + src: Указатель на исходную строку.
  + n: Количество символов для копирования.

***5) Отмена отображения и закрытие файла:***

Используйте функцию munmap для отмены отображения, и close для закрытия файла.

**munmap**: Отменяет отображение области памяти.

* **Параметры**:
  + addr: Адрес области, которую нужно отменить.
  + length: Длина области.

**close**: Закрывает открытый дескриптор файла.

* **Параметры**:
  + fd: Дескриптор файла.
* **Возвращает**: 0 при успешном закрытии, или -1 в случае ошибки.

1. ***Опишите алгоритм сборки программы, использующей статическую библиотеку с использованием CMake и Clang***

***1) Создание статической библиотеки:***

Напишите код для библиотеки и создайте CMakeLists.txt для сборки.

Пусть будет файл mylib.c

***2) Создание программы, использующей библиотеку:***

Напишите код программы, которая будет использовать функции из библиотеки.

Пусть программа будет main.c

***3) Создание CMakeLists.txt для библиотеки:***

Настройте CMake для сборки статической библиотеки.

-----------------------------------------------------------------------------------------------

# CMakeLists.txt для библиотеки

cmake\_minimum\_required(VERSION 3.10)

project(MyLib)

add\_library(mylib STATIC mylib.c) # Создание статической библиотеки

-----------------------------------------------------------------------------------------------

**add\_library**: Создает библиотеку.

* **Параметры**:
  + mylib: Имя библиотеки.
  + STATIC: Указывает, что это статическая библиотека.
  + mylib.c: Исходный файл для библиотеки.

***4) Создание CMakeLists.txt для программы:***

Настройте CMake для сборки программы и связывания с библиотекой.

**add\_executable**: Создает исполняемый файл.

* **Параметры**:
  + main: Имя исполняемого файла.
  + main.c: Исходный файл для программы.

**target\_link\_libraries**: Связывает исполняемый файл с библиотекой.

* **Параметры**:
  + main: Имя исполняемого файла.
  + PRIVATE: Указывает на область видимости (только для этого исполняемого файла).

***5) Сборка проекта:***

Используйте CMake для генерации файлов сборки и сборки проекта.

mkdir build

cd build

cmake .. ---Генерирует файлы сборки в директории build.

make ---Собирает проект, создавая как библиотеку, так и исполняемый файл.

***6) Запуск программы:***

Запустите скомпилированную программу.

./main

1. ***Опишите алгоритм сборки программы, использующей статическую библиотеку с использованием только Clang***

***1) Создание статической библиотеки:***

Напишите код для библиотеки и скомпилируйте его в статическую библиотеку.

Пусть будет файл mylib.c.

Компилируем в объектный файл: *clang -c mylib.c -o mylib.o*

-c: Указывает компилятору, что нужно создать объектный файл, а не исполняемый файл.

-o mylib.o: Имя выходного файла (объектного файла).

Теперь создайте статическую библиотеку из объектного файла с помощью команды ar: *ar rcs libmylib.a mylib.o*

**ar**: Утилита для создания и манипуляции статическими библиотеками.

* **Параметры**:
  + r: Вставляет файлы в библиотеку.
  + c: Создает библиотеку, если она не существует.
  + s: Создает индекс библиотеки (опционально).
  + libmylib.a: Имя выходного файла (статической библиотеки).
  + mylib.o: Объектный файл для добавления в библиотеку.

***2) Создание программы, использующей библиотеку:***

Напишите код программы, которая ссылается на функции из библиотеки.

Пусть программа называется main.c.

***3) Компиляция программы и библиотеки:***

Используйте Clang для компиляции программы и библиотеки.

Команда: *clang -o main main.c -L. –lmylib*

-o main: Имя выходного файла для программы.

-L.: Указывает компилятору искать библиотеки в текущей директории.

-lmylib: Указывает на подключение библиотеки libmylib.a. Префикс lib и суффикс .a не указываются.

***4) Запуск программы:***

Запустите скомпилированную программу.

1. ***Опишите алгоритм сборки программы, неявно использующей динамическую библиотеку с использованием CMake и Clang***

***1) Создание динамической библиотеки:***

Напишите код для библиотеки и скомпилируйте его с помощью CMake.

Пусть файл библиотеки будет называться mylib.c

***2) Создание программы, использующей библиотеку:***

Напишите код программы, которая ссылается на функции из библиотеки.

Пусть файл программы, которая будет использовать библиотеку будет называться main.c.

***3) Создание CMakeLists.txt для библиотеки:***

Настройте CMake для сборки динамической библиотеки.

**add\_library**: Создает библиотеку.

* **Параметры**:
  + mylib: Имя библиотеки.
  + SHARED: Указывает, что это динамическая библиотека.
  + mylib.c: Исходный файл для библиотеки.

**add\_executable**: Создает исполняемый файл.

* **Параметры**:
  + main: Имя исполняемого файла.
  + main.c: Исходный файл для программы.

**target\_link\_libraries**: Связывает исполняемый файл с библиотекой.

* **Параметры**:
  + main: Имя исполняемого файла.
  + PRIVATE: Указывает на область видимости. В данном случае это означает, что библиотека используется только в этом исполняемом файле.

***4) Создание CMakeLists.txt для программы:***

Настройте CMake для сборки программы и связывания с библиотекой.

***5) Сборка проекта:***

Используйте CMake для генерации файлов сборки и сборки проекта.

mkdir build

cd build

cmake .. (генерация файла сборки в директории build)

make (собирает проект, создавая как библиотеку, так и исполняемый файл)

***6) Запуск программы:***

Убедитесь, что библиотека доступна для загрузки во время выполнения.

1. ***Опишите алгоритм сборки программы, неявно использующей динамическую библиотеку с использованием только Clang***

***1) Создание динамической библиотеки:***

Напишите код для библиотеки и скомпилируйте его в динамическую библиотеку.

Создайте файл mylib.c.

Команда: clang -shared -o libmylib.so mylib.c –fPIC

-shared: Указывает, что необходимо создать динамическую библиотеку.

-o libmylib.so: Имя выходного файла.

-fPIC: Генерирует код, независимый от адреса, что необходимо для динамических библиотек.

***2)Создание программы, использующей библиотеку:***

Напишите код программы, которая ссылается на функции из библиотеки.

Пусть программа называется main.c.

***3)Компиляция программы и библиотеки:***

Используйте Clang для компиляции программы и библиотеки.

clang -o main main.c -L. –lmylib

-o main: Имя выходного файла для программы.

-L.: Указывает компилятору искать библиотеки в текущей директории.

-lmylib: Указывает на подключение библиотеки libmylib.so. Префикс lib и суффикс .so не указываются.

***4)Запуск программы:***

Убедитесь, что библиотека доступна для загрузки во время выполнения, и запустите программу.

LD\_LIBRARY\_PATH=. ./main

LD\_LIBRARY\_PATH=.: Указывает системе поиска библиотек, чтобы она искала библиотеки в текущей директории.

./main: Запускает скомпилированную программу.

1. ***Опишите алгоритм написания программы, явно использующей динамическую библиотеку средствами WinAPI.***

***1) Создание динамической библиотеки:***

Напишите код для библиотеки и скомпилируйте его в динамическую библиотеку (DLL).

Пусть библиотека называется *mylib.c.*

Компилируем в динамическую библиотеку: *gcc -shared -o mylib.dll mylib.c*

***2) Создание программы, использующей библиотеку:***

Напишите программу, которая загружает и использует динамическую библиотеку.

***3) Загрузка динамической библиотеки во время выполнения:***

Используйте функции WinAPI для загрузки библиотеки.

**LoadLibrary**: Загружает динамическую библиотеку (DLL).

* **Параметры**:
  + lpFileName: Имя библиотеки (например, "mylib.dll").
* **Возвращает**: Дескриптор библиотеки или NULL в случае ошибки.

***4)Вызов функций из библиотеки:***

Получите указатели на функции из загруженной библиотеки и вызовите их.

**GetProcAddress**: Получает адрес функции из загруженной библиотеки.

* **Параметры**:
  + hModule: Дескриптор библиотеки, полученный от LoadLibrary.
  + lpProcName: Имя функции (например, "hello").
* **Возвращает**: Адрес функции или NULL в случае ошибки.

***5)Очистка ресурсов:***

Освободите библиотеку после завершения ее использования.

**FreeLibrary**: Освобождает библиотеку, загруженную с помощью LoadLibrary.

* **Параметры**:
  + hModule: Дескриптор библиотеки, который нужно освободить.

1. ***Опишите алгоритм написания программы, явно использующей динамическую библиотеку средствами POSIX API.***

***1) Создание динамической библиотеки:***

Напишите код для библиотеки и скомпилируйте его в динамическую библиотеку.

***2)Создание программы, использующей библиотеку:***

Напишите программу, которая загружает и использует динамическую библиотеку.

Например библиотека называется *mylib.c*

Компиляция в динамическую библиотеку: *gcc -shared -o libmylib.so mylib.c -fPIC*

***3)Загрузка динамической библиотеки во время выполнения:***

Используйте функции POSIX для загрузки библиотеки.

Для загрузки библиотеки нужно использовать функцию dlopen:

* **Параметры**:
  + filename: Путь к библиотеке (например, "./libmylib.so").
  + flag: Флаги загрузки (например, RTLD\_LAZY для отложенной резолвации символов).

***4)Вызов функций из библиотеки:***

Получите указатели на функции из загруженной библиотеки и вызовите их.

**dlsym**: Получает адрес функции из загруженной библиотеки.

* **Параметры**:
  + handle: Дескриптор библиотеки, полученный от dlopen.
  + symbol: Имя функции (например, "hello").

***5)Очистка ресурсов:***

Освободите библиотеку после завершения ее использования.

**dlclose**: Освобождает библиотеку, загруженную с помощью dlopen.

* **Параметры**:
  + handle: Дескриптор библиотеки, который нужно освободить.

1. ***Опишите алгоритм создания Windows-сервиса средствами WinAPI (без SCM).***

***1) Создание основного кода сервиса:***

Напишите код, который реализует логику работы сервиса.

***2) Создание потока для выполнения сервиса:***

Используйте функции WinAPI для создания потока, который будет выполнять основную работу сервиса.

**CreateThread**: Создает новый поток.

* **Параметры**:
  + lpThreadAttributes: Атрибуты безопасности (обычно NULL).
  + dwStackSize: Размер стека (обычно 0 для стандартного размера).
  + lpStartAddress: Указатель на функцию потока (ServiceThread).
  + lpParameter: Параметры для функции потока (обычно NULL).
  + dwCreationFlags: Флаги создания (обычно 0).
  + lpThreadId: Указатель на идентификатор потока (обычно NULL).

***3) Обработка сигналов остановки:***

Реализуйте обработку сигналов для корректной остановки сервиса.

**SetConsoleCtrlHandler**: Устанавливает или удаляет обработчик для сигналов управления консолью.

* **Параметры**:
  + HandlerRoutine: Указатель на функцию обработчика (CtrlHandler).
  + Add: TRUE для добавления обработчика, FALSE для удаления.

***4) Запуск сервиса как отдельного процесса:***

Используйте CreateProcess для запуска вашего приложения как сервиса.

**CreateProcess**: Создает новый процесс и его основной поток.

* **Параметры**:
  + lpApplicationName: Имя исполняемого файла (обычно NULL).
  + lpCommandLine: Команда для запуска (с указанием пути к исполняемому файлу).
  + lpProcessAttributes: Атрибуты безопасности процесса (обычно NULL).
  + lpThreadAttributes: Атрибуты безопасности потока (обычно NULL).
  + bInheritHandles: Указывает, должны ли дескрипторы наследоваться (обычно FALSE).
  + dwCreationFlags: Флаги создания (например, 0 для обычного процесса).
  + lpEnvironment: Указатель на окружение (обычно NULL).
  + lpCurrentDirectory: Рабочая директория (обычно NULL).
  + lpStartupInfo: Указатель на структуру STARTUPINFO, содержащую информацию о запуске.
  + lpProcessInformation: Указатель на структуру PROCESS\_INFORMATION, содержащую информацию о созданном процессе.

1. ***Опишите алгоритм создания Windows-сервиса средствами SCM.***

***SCM*** – Service Control Manager

***1) Создание исполняемого файла сервиса:***

Напишите программу, которая будет выполнять задачи вашего сервиса, реализуя методы управления, такие как OnStart и OnStop.

***2) Создание скрипта установки:***

Напишите скрипт или используйте утилиту, чтобы зарегистрировать ваш сервис в SCM.

***3) Регистрация сервиса:***

Используйте команду sc create для добавления сервиса в SCM.

С помощью sc: sc create MyService binPath= "C:\Path\To\MyService.exe"

***4) Запуск сервиса:***

Запустите сервис с помощью команды sc start.

С помощью sc: sc start MyService

***5) Управление сервисом:***

Используйте команды sc stop, sc delete и другие для управления сервисом.

sc stop: sc stop MyService

sc delete: sc delete MyService

1. ***Опишите алгоритм создания Linux-сервиса средствами init.***

***1) Создание исполняемого файла сервиса:***

Напишите скрипт или программу, которая будет выполняться как сервис.

***2) Создание скрипта инициализации:***

Создайте скрипт, который будет управлять запуском, остановкой и перезапуском вашего сервиса.

***3) Размещение скрипта в правильной директории:***

Поместите скрипт в директорию /etc/init.d/.

***4)Настройка прав на выполнение:***

Установите права на выполнение для созданного скрипта.

Команда: sudo chmod +x [файл и где он]

***5) Добавление сервиса в управление уровнями выполнения:***

Используйте команды для добавления вашего сервиса в соответствующие уровни выполнения.

Команда: sudo update-rc.d [имя файла] defaults

Функции в скрипте инициализации:

**start:** Запускает сервис, выполняя команду, указанную в скрипте.

**stop:** Останавливает сервис, используя killall для завершения процесса.

**restart:** Перезапускает сервис, вызывая функции stop и start.

**status:** Проверяет, запущен ли сервис, используя pgrep для поиска процесса.

**case:** Структура управления, которая позволяет выполнять различные действия в зависимости от аргумента, переданного скрипту.

1. **Опишите алгоритм создания Linux-сервиса средствами systemd.**

**1) Создание исполняемого файла сервиса:**

Напишите скрипт или программу, которую вы хотите запустить как сервис.

**2) Создание конфигурационного файла для сервиса:**

Создайте файл с расширением .service в директории /etc/systemd/system/.

**3) Написание конфигурации сервиса:**

Определите основные параметры сервиса в конфигурационном файле.

**4) Перезагрузка конфигурации systemd:**

Используйте команду для перезагрузки конфигурации systemd.

Команда: *sudo systemctl daemon-reload*

**5) Запуск и управление сервисом:**

Запустите сервис и настройте его на автоматический старт при загрузке системы.

----Запуск сервиса: *sudo systemctl start myservice*

----Проверка статуса сервиса: *sudo systemctl status myservice*

----Установка автозапуска при загрузке: *sudo systemctl enable myservice*

----Остановка сервиса: *sudo systemctl stop myservice*

*Обязательные в конфигурационном файле надо указывать команду для запуска сервиса*