# 1.内存映射的原理

将一块内存空间映射到不同的进程空间中

# 2.define和const的区别

1.define 是预处理指令，用于创建符号常量。`const` 是 C 和 C++ 的关键字，用于创建具有常量值的变量，本质是只读变量。

2.`define` 在预处理阶段执行。`const` 在编译阶段执行。

3.`define` 没有类型检查，仅进行文本替换。`const` 有类型检查，可以与变量类型关联。

# 3.数组和链表的区别

1.数组内存连续，链表内存不连续。

2.数组访问速度比链表快

3.链表增加删除操作比数组快

# 4.指针和引用的区别

1.指针：指针是一个变量，保存着内存地址。引用：引用是已存在变量的别名，没有自己的内存地址。

2.指针可以具有空值(NULL)，引用不能为空，必须在初始化时指向一个有效的对象。

3.可以修改指针的指向，可以将指针重新赋值为另一个地址。一旦引用被初始化，它始终指向同一个对象，不可更改。

4.指针需要额外的内存空间来存储地址值。引用不需要额外的内存空间，因为它是对已存在变量的别名。

# 5.解释一下QT的信号与槽

信号与槽是一种用于实现对象之间通信和事件处理的机制。

# 6.IIC为什么要加上拉电阻，为什么使用开漏输出

## 上拉电阻

1.当IIC总线在空闲状态，SDA和SCL需要处于高电平状态。

2.开漏输出无法输出高电平，使用上拉电阻可以完成高低电平之间的转换。

## 开漏输出

1.假如使用推挽输出可能导致器件的烧毁

2.实现线与功能

# 7.MQTT的通信过程

1.创建客户端

2.指定IP地址和端口号

3.进行连接

4.发布主题或者订阅主题

5.数据传输

6.断开连接

# 8.在Linux中怎么实现同步

1.互斥锁

2.信号量

3.条件变量

# 9.TCP和UDP的应用场景

1.TCP：文件传输、电子邮件、网页浏览。

2.UDP：实时音视频传输、在线游戏、实时监控。

# 10.什么是野指针，什么情况会产生野指针

## 什么是野指针

1.指向已被释放或无效的内存地址的指针是野指针。

## 什么情况下产生野指针

1.内存释放后未置空指针

```
int *ptr = (int*)malloc(sizeof(int));
free(ptr);
*ptr = 10; // 这里ptr成为了野指针
```

2.返回局部变量的指针

```
int* getIntPointer() {
  int num = 5;
  return &num; // 返回局部变量的指针
}

int *ptr = getIntPointer();
*ptr = 10; // getIntPointer返回一个野指针
```

3.未初始化指针

```
char* p;
```

# 11.什么是互斥锁

互斥锁是一种用于线程同步的机制，用于确保同一时间只有一个线程访问共享资源。

# 12.数组和指针的区别

数组是一块连续的内存空间，其大小在编译时确定，访问元素使用下标操作；而指针是一个变量，存储地址值，大小固定，可以指向不同类型的数据，通过解引用操作访问内存中的数据。

# 13.如何防止重复引用头文件

1.使用预处理指令

```
#ifndef HEADER_FILE_NAME_H
```

```
#define HEADER_FILE_NAME_H
```

// 头文件内容

```
#endif // HEADER_FILE_NAME_H
```

2.使用#pragma once

# 14.栈和队列的区别

1.栈是后进先出的数据结构，而队列是先进先出的数据结构。

2.栈常用于表达式求值、函数调用的调用栈、括号匹配等需要后进先出的场景。队列常用于任务调度、缓冲区管理、消息传递等需要先进先出的场景。

# 15.为什么中断不能传递参数

中断是异步调用，无法知道什么时候会被调用，不能够像函数一样主动调用。

# 16.串口数据帧格式

起始位，数据位，校验位，停止位。

# 17.中断的概念

中断是计算机系统中一种重要的处理机制，用于响应某种事件或条件的发生。它是一种异步的事件，可以打断当前正在执行的程序或任务，以处理紧急情况或外部设备的请求。

# 18.static的作用

控制变量生命周期，控制作用域，控制文件可见性。

# 19.中断的执行过程

中断的执行过程包括中断请求、中断控制器响应、中断响应、中断向量确定、中断处理程序执行和中断处理程序结束。

# 20.什么是多态

多态是面向对象编程中的一种特性，它允许以一种统一的方式处理不同类型的对象，通过相同的接口可以表现出不同的行为。

# 21.C语言中的内存分配方式有几种

1.静态内存分配。

**2.栈上内存分配。**

3.堆上内存分配。

# 22.struct和class的区别

1.默认访问权限不同，struct的默认访问权限是public，class的默认访问权限是private。

2.继承方式不同，struct的默认继承方式是公有继承，class的继承访问是私有继承。

3.在一般情况下，struct被用于表示数据结构，而class则更多用于表示具有行为和数据的对象。

# 23.函数和中断的区别

1.函数调用不会发生上下文切换，中断调用会发生上下文切换。

2.函数可以主动被调用，中断无法主动被调用。

3.函数调用是同步的，中断是异步的。

4.函数可以有返回值和参数，中断没有返回值和参数。

# 24.自旋锁和信号量的区别

自旋锁是一种忙等待的方式，适用于临界区执行时间短暂、锁冲突概率低的情况；而信号量是一种阻塞机制，适用于临界区执行时间长、锁冲突概率高的情况，自旋锁不会进入休眠，信号量会进入休眠。

# 25.怎么判断链表是否有环

使用快慢指针。

思路：

1. 创建两个指针，一个指针称为快指针（fast），另一个指针称为慢指针（slow），初始时都指向链表的头节点。
2. 快指针每次向前移动两个节点，慢指针每次向前移动一个节点。
3. 如果链表中存在环，那么快指针和慢指针最终会相遇。
4. 如果链表中不存在环，那么快指针最终会先到达链表尾部，此时可以判断链表无环。

# 26.使用多线程时需要注意什么

1. 线程安全：多线程环境下，多个线程同时访问共享资源可能会引发竞态条件（Race Condition），导致数据不一致或其他异常情况。确保共享资源的访问是线程安全的，可以通过使用互斥锁（Mutex）、条件变量（Condition Variable）等同步机制来保护共享资源的访问。

2. 线程间通信：在多线程编程中，不同线程之间可能需要进行通信和协作。合理地设计和使用线程间通信机制，如队列（Queue）、信号量（Semaphore）、事件（Event）等，可以有效地实现线程之间的同步和传递信息。
3. 死锁：死锁是指两个或多个线程在互相等待对方释放资源而无法继续执行的状态。避免死锁的方法之一是按照固定的顺序获取锁，避免循环依赖。另外，可以使用资源分配图等方法进行死锁检测和预防。
4. 上下文切换开销：线程切换需要保存当前线程的上下文并加载下一个线程的上下文，这涉及到时间和空间的开销。在设计多线程应用程序时，需要注意减少线程切换的频率，避免过度创建线程和过度频繁地切换线程，以提高程序性能。
5. 共享资源的合理使用：多线程环境下，共享资源可能被多个线程同时访问，需要注意共享资源的正确使用和保护。避免线程之间的竞争和冲突，需要考虑线程安全性，使用适当的同步机制对共享资源进行保护。
6. 有效的线程调度和任务划分：在多线程编程中，线程的调度和任务的划分对系统性能和响应能力有重要影响。合理规划线程数量和调度策略，均衡地分配任务，避免线程之间的争抢和饥饿现象，以提高系统整体的吞吐量和响应性能。
7. 错误处理和异常处理：在多线程环境下，错误和异常的处理需要更加谨慎。及时捕获和处理线程中的异常，确保程序的稳定性和可靠性。

# 27.实现strcpy函数

```
char* mystrcpy(char* str, const char* str1)
{
    char* temp = str;

    while (*temp++ = *str1++);

    return str;
}
```

# 28.实现strcat函数

```
char* mystrcat(char* str1, char* str2)
{
    char* temp = str1;
    while (*temp != '\0')
    {
        temp++;
    }

    while (*str2 != '\0')
    {
        *temp = *str2;
        temp++;
        str2++;
    }

    *temp = '\0';

    return str1;
```

```
}
```

# 29.实现strlen函数

```
int mystrlen(char* str)
{
    int i = 0;
    while (*str++ != '\0')
    {
        i++;
    }
    return i;
}
```

# 30.new和malloc的区别

1.new是C++中的关键字，malloc是标准库中的函数。

2.使用new创建对象会调用构造函数，使用malloc创建对象不会调用构造函数。

3.使用new不需要指定对象的大小，使用malloc需要指定对象的大小。

4.new返回的是一个对象的指针，malloc返回的是void*

# 31.可执行程序生成的过程

预处理--编译--汇编--链接

# 32.UART，SPI，IIC的区别和概念

1.UART概念：UART是一种异步串行通信协议，用于在两个设备之间实现简单的点对点通信。它使用两根传输线（TX和RX）进行数据传输，其中TX（发送线）负责发送数据，RX（接收线）负责接收数据。

2.SPI概念：SPI是一种同步串行通信协议，用于在一个主设备（主控器）和一个或多个从设备之间实现全双工的高速数据传输。

3.IIC概念：I2C是一种串行双线制通信协议，用于在多个设备之间进行数据传输。它使用两根传输线（SDA和SCL）进行数据传输，其中SDA（串行数据线）负责发送和接收数据，SCL（串行时钟线）用于数据同步。

区别：

UART是异步通信协议，用于点对点通信；SPI是同步通信协议，适用于高速数据传输；I2C是双线制通信协议，适用于连接多个低速外设的场景。

# 33.软件中断和硬件中断的区别

1.软件中断是由特殊指定触发，硬件中断是由外部设备或处理器内部产生的信号触发的。

2.软件中断可以是同步也可以是异步的，硬件中断是异步的。

3.软件中断响应时间较长，硬件中断响应时间非常短。

# 34.内联函数和宏函数的区别

1.内联函数在编译时展开，而宏函数在预处理时展开。

2.内联函数会进行错误检查，宏函数是直接替换不会进行错误检查。

3.内联函数可以进行调试，宏函数无法进行调试。

4.内联函数有作用域规则，只能在定义它的源文件内使用，不能被其他源文件调用。而宏函数没有作用域限制，可以在整个程序中使用。

# 35.冒泡排序

冒泡排序（Bubble Sort）是一种简单且直观的排序算法，属于比较排序算法的一种。它通过多次迭代比较和交换相邻的元素，将较大（或较小）的元素逐渐“浮”到数组的一端，从而实现排序。

```
void bubbleSort(int* array, int size)
{
    for (int i = 0; i < size - 1; i++)
    {
        for (int j = 0; j < size - i - 1; j++)
        {
            // 比较相邻的两个元素，如果顺序错误就交换它们
            if (array[j] > array[j + 1])
            {
                int temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
}
```

# 36.选择排序

选择排序（Selection Sort）是一种简单且直观的排序算法，属于比较排序算法的一种。它的基本思想是从未排序部分的数组中选择最小（或最大）的元素，然后将其放置在已排序部分的末尾，不断重复这个过程，直到整个数组排序完成。

```
void selectionSort(int* array, int size)
{
    for (int i = 0; i < size - 1; i++)
    {
        int minIndex = i;
        for (int j = i + 1; j < size; j++)
        {
            // 寻找未排序部分的最小元素的索引
            if (array[j] < array[minIndex])
            {
                minIndex = j;
            }
        }
        // 将最小元素与当前位置交换
        int temp = array[i];
```

```
        array[i] = array[minIndex];
        array[minIndex] = temp;
    }
}
```

# 37.插入排序

插入排序（Insertion Sort）是一种简单且直观的排序算法，属于比较排序算法的一种。它的基本思想是将数组分为已排序部分和未排序部分，初始时已排序部分包含第一个元素，然后逐步将未排序部分的元素插入到已排序部分的合适位置，直到整个数组排序完成。

```
void insertionSort(int array[], int size)
{
    for (int i = 1; i < size; i++)
    {
        int key = array[i];
        int j = i - 1;

        // 在已排序部分中找到合适的位置插入元素
        while (j >= 0 && array[j] > key)
        {
            array[j + 1] = array[j];
            j--;
        }
        array[j + 1] = key;
    }
}
```

# 38.快速排序

```
// 交换两个元素的值
void swap(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// 将数组划分为两个子区间，并返回枢轴的索引
int partition(int arr[], int low, int high)
{
    int pivot = arr[high];  // 选择最后一个元素作为枢轴
    int i = (low - 1);  // 记录小于枢轴的元素的索引

    for (int j = low; j <= high - 1; j++)
    {
        // 如果当前元素小于或等于枢轴，则交换位置
        if (arr[j] <= pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
```

```
    // 把枢轴放到正确的位置上，并返回索引
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

// 快速排序
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        // 划分数组，获取枢轴索引
        int pivotIndex = partition(arr, low, high);

        // 递归对枢轴左侧的子数组进行排序
        quickSort(arr, low, pivotIndex - 1);

        // 递归对枢轴右侧的子数组进行排序
        quickSort(arr, pivotIndex + 1, high);
    }
}
```

# 39.归并排序

# 40.堆排序

# 41.C语言中struct和C++中struct的区别

1.C++中的struct可以支持继承，C语言中的struct不支持继承。

2.C++中的struct可以包含成员函数，C语言中的struct不可以包含成员函数。

# 42.进程和线程的区别

1.线程是程序调度的基本单位，进程是资源分配的基本单位。

2.一个进程崩溃不会导致其他进程崩溃，一个线程崩溃可能会导致整个程序崩溃。

3.创建和销毁进程所消耗的资源比较多，创建和销毁线程所消耗的资源比较少。

4.每个进程都有独立的内存空间和系统资源，线程是在进程内部创建的，共享相同的内存空间和系统资源。

# 43.volatile的本质和作用

volatile的本质是仿真编译器对所修饰的变量进行优化。

作用：

1.访问硬件寄存器：当需要直接访问硬件寄存器或外设寄存器时，使用 `volatile` 可以确保编译器每次访问该寄存器时都从内存中读取或写入，而不是从寄存器的缓存中获取值。

2.处理并发访问和中断：在多线程或多进程环境中，多个线程或进程可能同时访问共享的变量。

# 44.什么是大小端

大小端（Endian）指的是在计算机存储和处理多字节数据时，字节的存储顺序。

- 大端序（Big Endian）：高位字节存储在低地址，低位字节存储在高地址。类似于将数字的高位放在左边，低位放在右边的表示方式。例如整数值 `0x12345678` 在内存中按照大端序排列为 `12 34 56 78`。
- 小端序（Little Endian）：低位字节存储在低地址，高位字节存储在高地址。类似于将数字的低位放在左边，高位放在右边的表示方式。例如整数值 `0x12345678` 在内存中按照小端序排列为 `78 56 34 12`。

# 45.判断系统大小端

在C语言中，可以使用联合（union）的特性来判断系统的大小端（Big Endian 或 Little Endian）。

```c
#include <stdio.h>

union EndianCheck
{
    int i;
    char c[sizeof(int)];
};

int main()
{
    union EndianCheck ec;
    ec.i = 1;

    if (ec.c[0] == 1)
    {
        printf("Little Endian\n");
    }
    else
    {
        printf("Big Endian\n");
    }

    return 0;
}
```

# 46.define和typedef的区别

1. `define` 用于创建宏定义，在预处理时进行简单的文本替换。

2. `typedef` 用于创建类型别名，在编译时进行类型检查，并为已有类型创建新的名称。

3. `define` 是预处理指令，`typedef` 是关键字。

4. `define` 不会进行类型检查，而 `typedef` 会。

5. `typedef` 在定义新类型时更为灵活和安全。

# 47.使用fork函数和vfork函数创建进程的区别

1.使用fork创建的子进程和父进程有独立的地址空间。

2.vfork 创建的子进程与父进程共享地址空间，包括代码段、数据段和堆栈等。

3.使用fork时父子进程的执行顺序是不确定的，取决于系统调度。

4.vfork会暂停父进程的执行，直到子进程调用exec或_exit，在此期间，父进程一直等待子进程的结束。

# 48.进程间通信方式

1. 管道（Pipe）：
    - 匿名管道是一种单向通信机制，允许一个进程向另一个进程传输数据。
    - 管道可用于具有亲缘关系的进程（例如父子进程）之间进行通信，常用于单个读者和单个写者的情况。
2. 命名管道（Named Pipe）：
    - 命名管道是一种通过文件系统路径来命名的管道，可用于不具有亲缘关系的进程之间进行通信。
    - 命名管道可以实现多个读者和写者之间的通信。
3. 共享内存（Shared Memory）：
    - 共享内存是一种高效的进程间通信方式，允许多个进程直接访问相同的物理内存区域。
    - 通过将内存区域映射到各个进程的地址空间，进程可以直接读写共享内存，避免了数据的复制。
4. 信号量（Semaphore）：
    - 信号量是一种用于多进程之间同步和互斥的机制。
    - 进程可以通过操作信号量来进行进程间的互斥访问，以保护共享资源。
5. 消息队列（Message Queue）：
    - 消息队列是一种通过内核维护的消息缓冲区实现的通信机制，用于进程之间传递数据。
    - 进程可以将消息发送到消息队列，并由其他进程接收和处理。
6. 套接字（Socket）：
    - 套接字是一种网络编程中常用的进程间通信机制，可用于不同主机上的进程进行通信。
    - 套接字提供了一种基于网络协议的可靠的双向数据传输方式。
7. 文件（File）：
    - 进程可以通过读写共享文件来进行通信，多个进程可以访问同一个文件并进行数据的读写。

# 49.全局变量和局部变量的区别

1.作用域：

全局变量：全局变量在整个程序中都是可见和可访问的，被定义在全局作用域中。

局部变量：局部变量只在其被定义的特定函数、代码块或作用域内部可见和可访问。

2.生命周期

全局变量：全局变量的生命周期从程序开始运行到程序结束，它在内存中一直存在。

局部变量：局部变量的生命周期仅在其定义的特定作用域内部存在。

3.初始值

全局变量：全局变量可以在定义时显示地指定初始值，如果没有指定初始值，全局变量会被默认初始化为零值。

局部变量：局部变量在定义时不会被自动初始化，其初始值是未定义的（即不确定的），需要显式地给定一个初始值。

4.存储位置

全局变量：全局变量通常被存储在数据段（data segment）中，具有固定的存储位置，且在整个程序的执行期间都保持不变。

局部变量：局部变量通常被存储在栈（stack）中，它们的存储位置在函数调用过程中动态分配和释放，每次函数执行时都会创建一个新的副本。

# 50.什么是交叉编译

交叉编译（Cross-compilation）是指在一台计算机上使用特定的编译工具链将代码编译成目标平台上可执行的程序或库。

# 51.static和const的区别

1.可变性

# 52.define和typedef的区别

1.用法和语法

define：#define是一个预处理指令，用于在代码中创建常量或宏定义。

typedef：typedef关键字用于为已存在的数据类型创建新的名称，可以通过typedef为存在的类型创建别名。

2.处理阶段

define：#define指令是在预处理阶段进行处理，即在编译之前进行文本替换。它将标识符替换为指定的文本，不进行任何类型检查。

typedef：typedef是在编译阶段进行处理，在编译器解析代码时处理类型信息。它为已存在的类型创建一个新的别名。会进行类型检查

3.作用范围

define：#define指令的作用范围是整个文件或者被它所定义的预处理区域，可以在文件的任何位置进行定义和使用。

typedef：typedef创建的别名具有更窄的作用域，仅在指定的代码块、函数、结构体或枚举内部有效。

# 53.构造函数能否是虚函数

构造函数不能被声明为虚函数。

# 54.析构函数能否是虚函数

析构函数可以被声明为虚函数。

# 55.有名管道和无名管道

1.创建方式不同

有名管道使用mkfifo函数创建，无名管道使用pipe函数创建。

2.生存周期

有名管道可以持久存在于文件系统中，无名管道的生命周期仅限于创建它的父子进程的生命周期。

3.通信限制

无名管道只能用于亲缘关系间的进程进行通信，有名管道可以进行非亲缘关系之间的进程进行通信。

# 56.进程有几种状态

1.创建状态（New）：当一个进程刚刚被创建时，它处于新建状态。此时，操作系统为进程分配必要的资源，并对其进行初始化。

2.就绪状态（Ready）：当一个进程已经被创建并具备运行所需的所有资源时，但尚未被调度执行，它处于就绪状态。在就绪状态下，进程等待操作系统的调度，以便在合适的时机执行。

3.运行状态（Running）：当操作系统将 CPU 资源分配给处于就绪状态的进程时，进程进入运行状态。在运行状态下，进程正在执行其指令和操作。

4.阻塞状态（Blocked）：当一个进程无法继续执行，因为它正在等待某种事件的发生，比如等待输入/输出、等待资源释放等，此时进程进入阻塞状态。在阻塞状态下，进程会释放 CPU 资源，让其他可运行的进程有机会执行。

5.终止状态（Terminated）：当一个进程完成了其任务或由于某种原因被操作系统终止时，它进入终止状态。在终止状态下，进程释放它所占用的资源，并等待系统回收。