习题2

> 1. 简述使用交叉编译的主要原因

- 跨平台开发：不同的硬件平台和操作系统通常具有不同的指令集和二进制格式。通过交叉编译，开发人员可以在一种平台上编写代码，并将其编译为在其他平台上运行的可执行文件或库。这使得跨平台开发更加容易和高效。
- 资源限制：某些嵌入式系统、嵌入式设备或嵌入式操作系统的资源有限，例如处理器速度较慢、内存容量较小等。在这些情况下，使用交叉编译可以在更强大的主机上编译代码，然后将生成的可执行文件传输到资源受限的目标平台上运行。
- 性能优化：通过交叉编译，可以在一种平台上进行优化编译，以获得更好的性能。例如，可以使用更高级的处理器、更大的内存和优化的编译器选项来生成在目标平台上运行的高性能代码。
- 开发效率：交叉编译可以提高开发效率。开发人员可以在他们熟悉和喜欢的开发环境中编写代码，并使用强大的开发工具和调试器进行调试。然后，他们可以使用交叉编译工具链将代码转换为目标平台上的可执行文件，而不需要在目标平台上进行开发和调试。
- 更新和维护：使用交叉编译可以更轻松地进行软件更新和维护。开发人员可以在开发主机上进行更新，然后将更新的版本编译为目标平台上的可执行文件或库，并将其传输到目标设备上。这样，更新和维护过程更加灵活和高效。

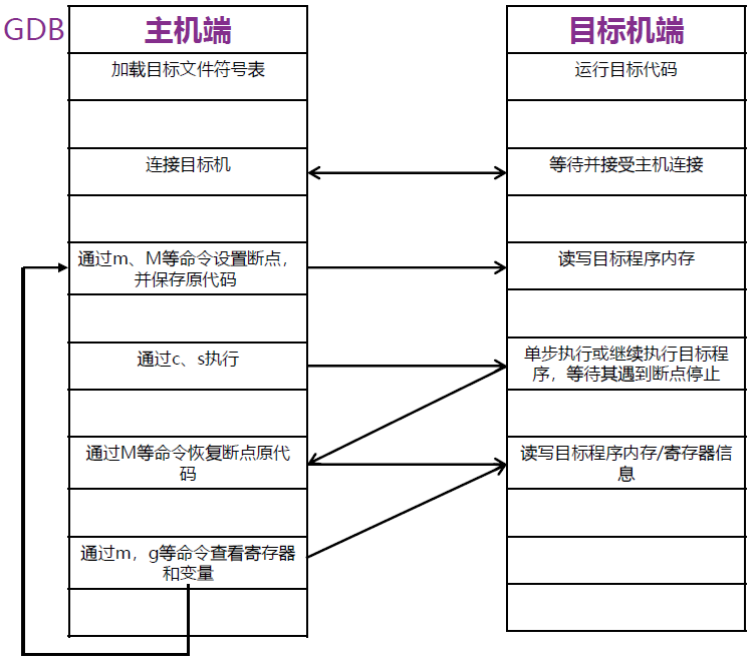
> 2. ARM微处理器的寄存器在Thumb状态与ARM状态下相同的是

- R0-R7，除此之外，两个状态下PC、SP、LR、CPSR寄存器也有对应关系

> 3. 简述GDB远程调试的步骤流程；列出GDB调试器中提供用于追踪变量的命令，并说明命令功能之间的异同点

- 步骤流程
  1. 主机端加载目标文件符号表，目标机端运行目标代码
  2. 主机端连接目标机，目标端等待并接受主机连接
  3. 主机端通过m、M等命令设置断点，并保存原代码，目标机端读写目标程序内存
  4. 主机端通过c、s执行，目标机端单步执行或继续执行目标程序，等待其遇到断点停止
  5. 主机通过M等命令恢复断点原代码，或通过m、g等命令查看寄存器和变量，目标机读写目标程序
     内存/寄存器信息。
  6. 重复3-5步骤，直至远程调试结束。

- 流程图可以总结为：



- GDB中追踪变量的命令
  - `print`、`display`、`watch`、`info`、`x`
- 异同
  - 相同点：都可以用于追踪变量的值
  - 不同点：具体用法和功能有差异，具体如下：
    - `print`用于查看变量的值，可以打印出变量的值、类型和地址等信息
    - `display`用于持续地查看变量的值，每次停在某个断点时自动显示指定变量的值；`display`命令比`print`命令更适用于需要持续查看变量值的情况
    - `watch`用于在变量被修改时停在程序，可以对指定的变量设置监视点，当变量被修改时程序将会停在相应的位置；相对于单纯的查看变量值，它可以更及时地发现变量被修改的情况
    - `info`可以查看一些值，`info locals`用于查看当前函数的本地变量，`info args`用于查看当前函数的参数，`info registers`用于查看除了浮点寄存器以外的寄存器
    - `x`用于查看地址中的内容，可以查看内存中某个地址的值、指定长度的值列表等

> 4. 开发嵌入式系统时，需要构建一个宿主机-目标机的开发环境。若目标机是裸机，那么为了调试和下载软件需要将调试仿真器连接到目标机的哪一种接口？为什么？
> A. SPI 接口
> B. 以太网接口
> C. JTAG 接口
> D. USB 接口

- C. JTAG接口
- JTAG接口是一种用于测试PCB板级和芯片级电子设备的硬件标准接口，可以用于目标机资源受限的情况。目标机为裸机时，无法使用需要复杂驱动的网络通讯等方式，也没有操作系统，通过JTAG接口，调试仿真器可以访问目标机CPU的调试接口，对目标机的软硬件进行调试和下载，包括读写寄存器、内存的内容，单步执行程序等操作，实现了对嵌入式系统软硬件的全面掌控。

> 5. 下列关于 Bootloader 的陈述中，不正确的是（）
> A. Bootloader 主要完成内存加电自检、外设存在自检、初始化外围设备、加载和启动操作系统等功能
> B. QNX 是支持多种嵌入式 CPU 的 Bootloader 程序
> C. 大多数从 Flash 存储器上启动的 bootloader 采用 stage1 和 stage2 两个阶段完成操作系

> 统的引导
> 加载
> D. Bootloader 的实现依赖于 CPU 的体系结构

- B. QNX 是支持多种嵌入式 CPU 的 Bootloader 程序
- QNX是一个实时操作系统，而不是Bootloader程序。虽然QNX可能包含一些引导加载程序的功能，但它本身不是一个专门的Bootloader程序。

> 6. 在基于 ARM 微处理器为核心的硬件平台上，开发其 Linux 环境下的应用程序 exp1.c。若需要编译后生成 exp1.c 对应的汇编程序，应使用的命令是（__）
> A. arm-linux-gcc -S exp1.c
> B. arm-linux-gcc -o exp1 exp1.c
> C. arm-linux-gcc -c exp1.c
> D. arm-linux-gcc -g -oexp1 exp1.c

- A. arm-linux-gcc -S exp1.c
- `-S` 参数表示仅编译成汇编代码，而不进行汇编和链接，这样可以查看编译器在翻译源代码时生成的汇编代码

> 7. ARM Linux 中，在进入中断响应之前，CPU 将依次进行哪些操作，从下列选项中选出正确的进行排序：
> （10分）
> a. 将 sp 指向 0x18。
> b. 将 cpsr 原来的内容装入 spsr_irq，即中断模式的 spsr；同时改变 cpsr 的内容 使 CPU 运行于中断模式，并关闭中断。
> c. 将进入中断响应前的内容装入 r14_irq，即中断模式的 lr，使其指向中断点。
> d. 从中断请求寄存器获取中断源。
> e. 将 pc 指向 0x18。
> f. 将堆栈指针 sp 切换成中断模式的 sp_irq。
> g. 从中断控制寄存器获取中断源。

- 正确的顺序是：c、b、f、e

> 8. 内存碎片分为：内部碎片和外部碎片。内部碎片就是已经被分配出去（能明确指出属于哪个进程）却不能被利用的内存空间；外部碎片指的是还没有被分配出去（不属于任何进程），但由于太小了无法分配给申请内存空间的新进程的内存空闲区域。
>
> 请分别阐述内部碎片和外部碎片产生的原因。Buddy 方案对应的是哪种碎片？简要说明 Buddy 算法的工作原理。

- 内部碎片原因：内部碎片就是已经被分配出去（能明确指出属于哪个进程）却不能被利用的内存空间。它通常是由于分配给进程的内存块的大小超过了进程所需的实际内存大小而产生的。这块存储块因为占用而无法被系统重新利用，直到被进程释放或进程结束，系统才有可能利用这个存储块。例如，如果一个进程需要分配10个字节的内存，但是系统只能以页的大小（通常是4KB）为单位进行内存分配，那么该进程将占用整个4KB的内存块，其中3,990个字节将被浪费。
- 外部碎片原因：还没有被分配出去（不属于任何进程），但由于太小了无法分配给申请内存空间的新进程的内存空闲区域。它是由于进程的内存释放不是连续的，导致内存空闲区域被分割成多个小块，而无法满足新进程所需的内存大小。即使总体上有足够的可用内存，但由于这些可用内存分散在不连续的小块中，无法满足某些进程对较大连续内存块的需求。
- Buddy方案对应的是外部碎片
- Buddy算法工作原理：核心是任何正整数都可以由2的幂之和组成。算法工作时，将所有可用的空闲页分为若干个块链表，每个链表都包含大小为2的自然数幂个连续页的块。当需要分配内存时，先从对应链表中查找是否有可用的空闲块，若有则直接分配；若没有，则在页数更大的链表中查找并分裂可用的块，并将剩余的块放回较小的链表中。反之，释放内存时先从对应链表中查找是否有

与之相邻的（地址上连续的）块，若没有则将新块插入到链表中，若有则将两个连续的块合并成更大的块，并将此块插入到页数更大的链表中，以便后续分配使用。

> 9. Ext3日志文件系统的特点是（）
>
> A. 高可用性
> B. 数据的完整性
> C. 数据转换快
> D. 多日志模式

- ABCD

> 10. 在 ARM Linux 系统中，中断处理程序进入 C 代码以后，ARM 的处于（ ）工作模式。
> A. 超级用户（SVC）
> B. 中断（IRQ）
> C. 快速中断（IRQ）
> D. 和进入中断之前的状态有关系

- A

> 11. 在 ARM Linux 体系中，用来处理外设中断的异常模式是（）
> A. 软件中断（SWI）
> B. 未定义的指令异常
> C. 中断请求（IRQ）
> D. 中止中断请求

- C

> 12. 简述 Ext3 的日志模式及工作原理

- 日志模式： 日志在磁盘的某个区域上完整记录整个磁盘的写入动作，以便于有需要时可以回溯追踪。ext3提供多种日志模式，即无论改变文件系统的元数据，还是改变文件系统的数据（包括文件自身的改变），ext3 文件系统均可支持。具体来说，分为以下日志模式：
  - Journal: 文件系统所有数据和元数据的改变都被记入日志。这种模式减少了丢失每个文件修改的机会，但是它需要很多额外的磁盘访问。例如，当一个新文件被创建时，它的所有数据块都必须复制一份作为日志记录。这是最安全和最慢的EXT3日志模式。
  - Ordered: 只有对文件系统元数据的改变才被计入日志。然而，EXT3文件系统把元数据和相关的数据块进行分组，以便在元数据之前把数据块写入磁盘。这样，就可以减少文件内数据损坏的机会。这是缺省的EXT3日志模式。
  - Writeback: 只有对文件系统元数据的改变才能计入日志；这是在其他日志文件系统中发现的方法，也是最快的模式。
- 工作原理：ext3是对ext2扩展形成的日志文件系统——日志文件系统会将整个磁盘的写入动作完整记录在磁盘的某个区域上，以便于有需要时可以回溯追踪，目标是避免对整个文件系统进行耗时的一致性检查。ext3日志文件系统的思想是对文件系统进行的任何高级修改都分两步进行。首先，把待写块的一个副本存放在日志中；其次，当发往日志的I/O数据传送完成时（把数据提交到日志后），待写块就被写入文件系统。当发往文件系统的I/O数据传送终止时（把数据提交到文件系统后），日志中的块的副本就被丢弃。

> 13. 简述ARM Linux的各个进程状态的转变关系，试举例每种转变如何触发。

- ARM Linux的进程状态分为5种：
  - TASK_RUNNING：运行态和就绪态的合并，表示进程正在运行或准备运行（包括Running队列中等待被安排到CPU的进程）
  - TASK_INTERRUPTIBLE：浅度睡眠态（被阻塞），等待资源到来时唤醒，也可以通过其他进程信号或时钟中断唤醒，进入运行队列

    - TASK_UNINTERRUPTIBLE：深度睡眠态。不可中断，指的并不是不响应外部硬件的中断，而是指进程不响应异步信号。直接等待硬件条件，资源有效时才唤醒
    - TASK_STOPPED：进程被暂停，等待其他进程的信号才能唤醒。在调试期间进程都会进入这个状态。向进程发送一个SIG_CONT信号，可以让其从TASK_STOPPED状态恢复到TASK_RUNNING状态
    - TASK_ZOMBIE：僵死状态，进程已经结束但未释放PCB，等待其父进程收集相关信息
- 它们之间的转变关系和触发条件为：
    - TASK_RUNNING -> TASK_RUNNING：进程从就绪状态转换到运行状态，触发条件为调用scheduler()或时钟中断发现时间片到。
    - TASK_RUNNING -> TASK_INTERRUPTIBLE：进程从运行状态转换到浅度睡眠状态，通过 interruptible_sleep_on() 等触发，等待资源就绪，进入浅度睡眠。
    - TASK_RUNNING -> TASK_UNINTERRUPTIBLE：进程从运行状态转换到深度睡眠状态，等待资源就绪，调用 sleep_on()，进入不可中断睡眠状态。
    - TASK_RUNNING -> TASK_STOPPED：进程从运行状态转换到暂停状态，收到 SIGSTOP 等信号，挂起进程。
    - TASK_RUNNING -> TASK_ZOMBIE：进程从运行状态转换到僵尸状态，进程已经结束但尚未释放PCB，主动调用 do_exit()。
    - TASK_UNINTERRUPTIBLE -> TASK_RUNNING：进程从深度睡眠状态转换到运行状态，有效资源就绪，通过外界 wake_up() 唤醒。
    - TASK_INTERRUPTIBLE -> TASK_RUNNING：进程从浅度睡眠状态转换到运行状态，有效资源就绪，通过 wake_up()，wake_up_interruptible() 或者收到信号唤醒。
    - TASK_STOPPED -> TASK_RUNNING：进程从暂停状态转换到运行状态，有收到 SIGCONT 信号，线程进入就绪状态。