在开始本次课程学习之前，请确定你已经在自己的计算机上准备好了调试环境。

《嵌入式系统》

5 嵌入式软件开发环境与调试技术

提纲

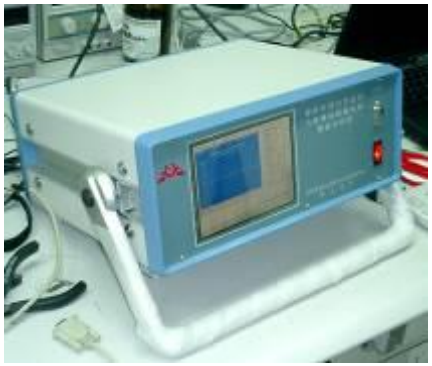
- 嵌入式软件开发概述
- 通用软件调试技术概述*
- 嵌入式软件调试技术

提纲

- 嵌入式软件开发概述
- 通用软件调试技术概述*
- 嵌入式软件调试技术

嵌入式系统不具备自主开发能力

- 由于计算、存储、显示等资源受限，嵌入式系统无法完成自主开发。

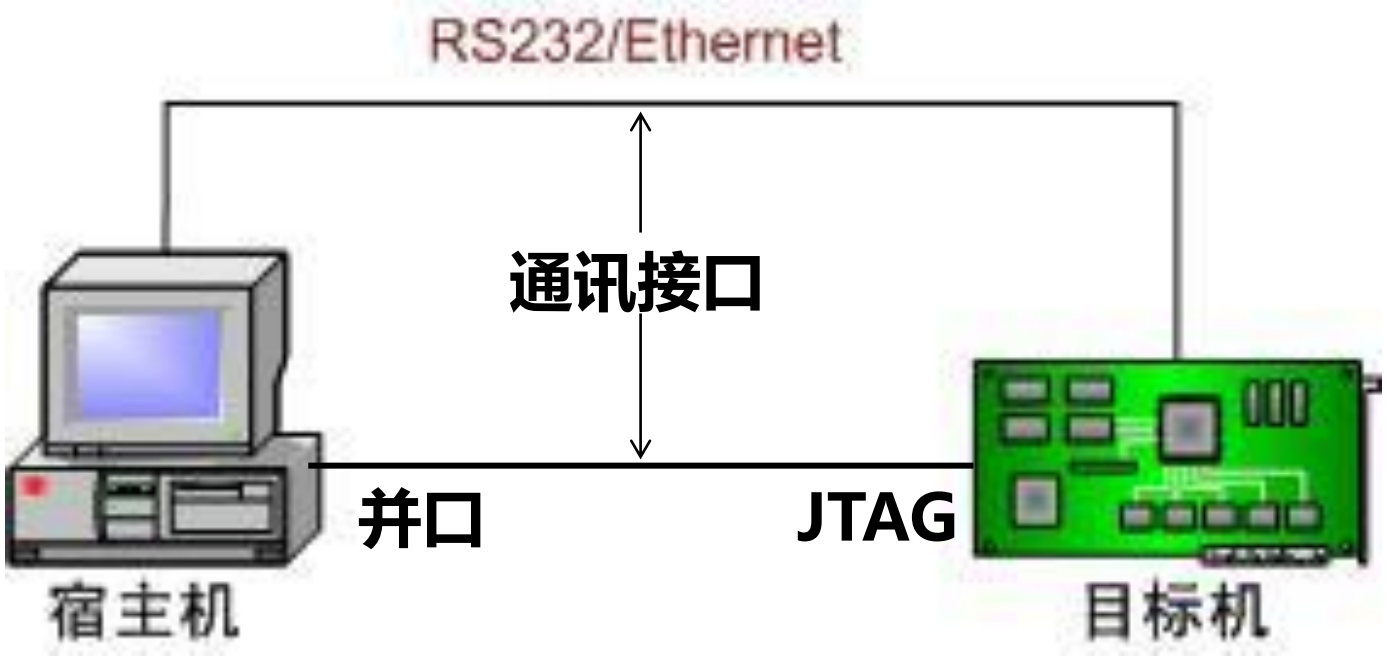


嵌入式软件开发模式

- 嵌入式系统资源受限，直接在嵌入式系统硬件平台上编写软件较为困难
- 解决方法
  - 首先在通用计算机上编写软件
  - 然后通过本地编译或者交叉编译生成目标平台上可以运行的二进制代码格式
  - 最后再下载到目标平台上运行

宿主机 - 目标机开发模式

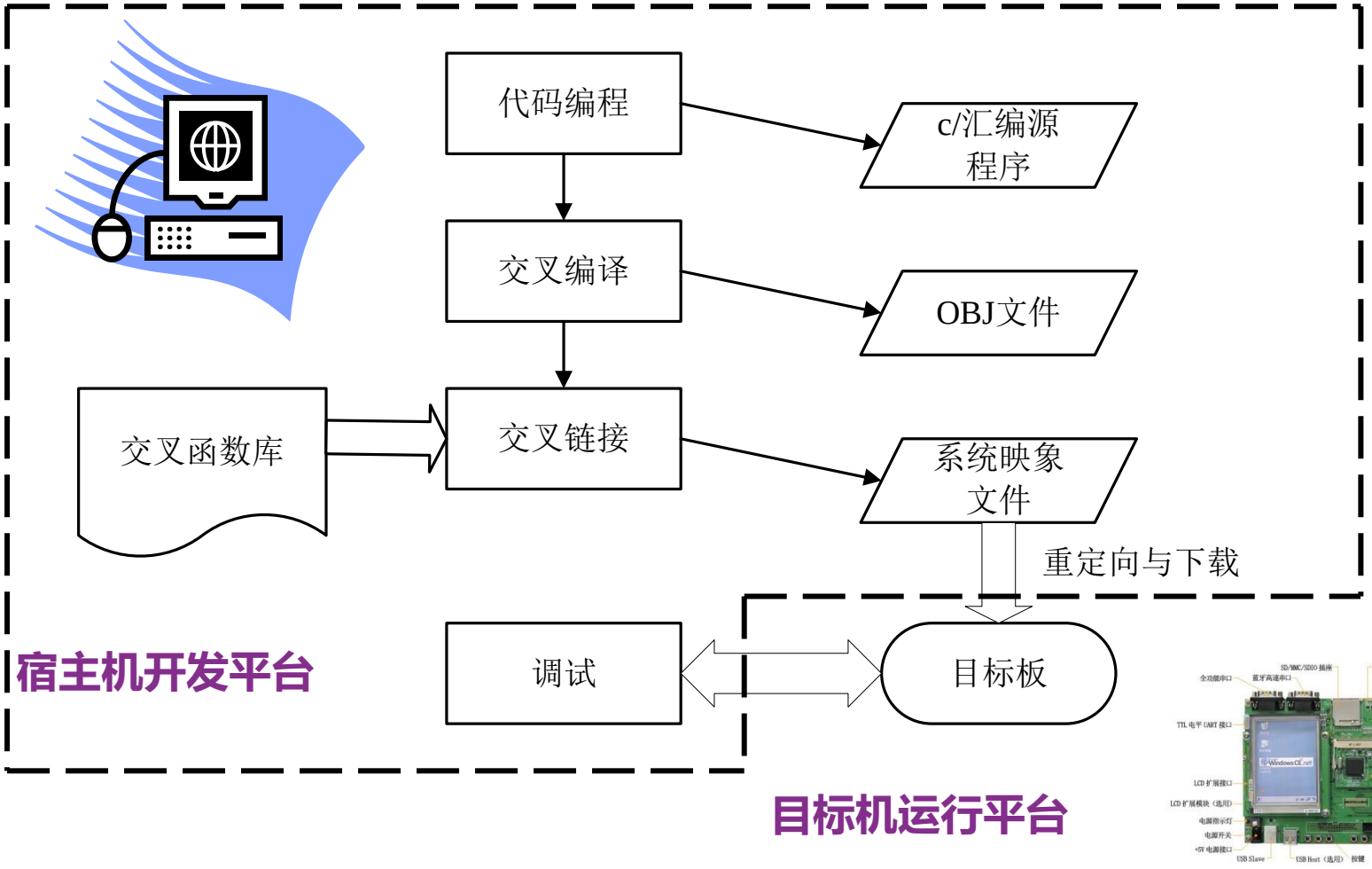
- 嵌入式系统采用双机开发模式：宿主机 - 目标机开发模式，利用资源丰富的PC机来开发嵌入式软件。



宿主机：资源丰富

目标机：资源受限

嵌入式软件开发流程

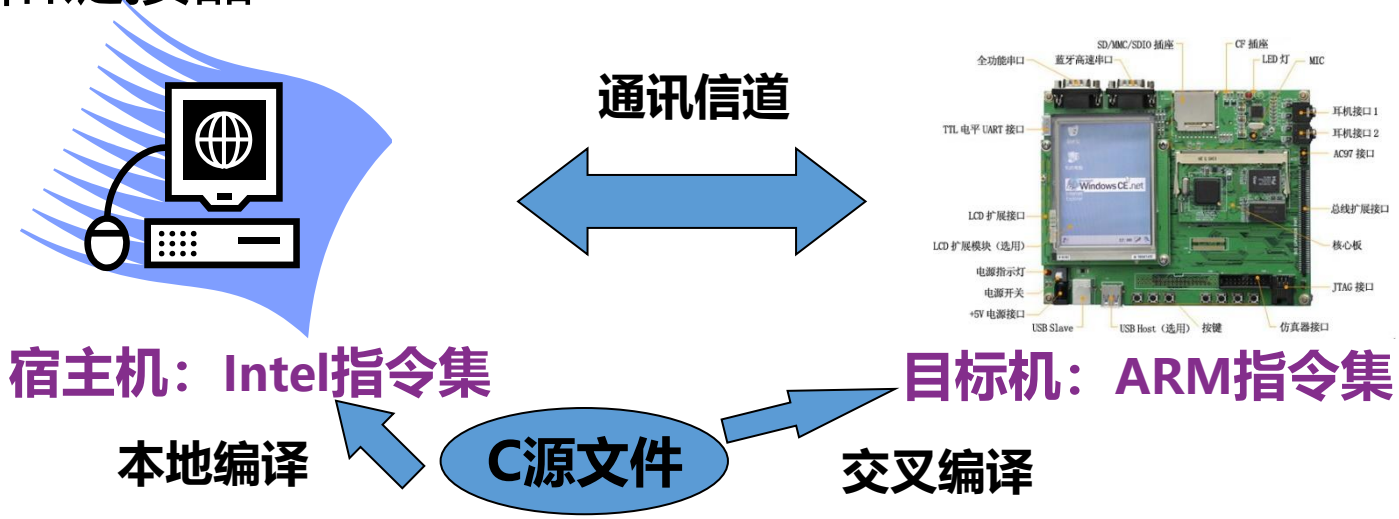

代码编程
c/汇编源程序
交叉编译
OBJ文件
交叉函数库
交叉链接
系统映象文件
重定向与下载
调试
目标板
宿主机开发平台
目标机运行平台

关于交叉编译

- 什么是交叉编译
  - 在一种平台上编译出能在另一种平台（体系结构不同）上运行的程序；
  - 在PC平台(X86)上编译出能运行在ARM平台上的程序，即编译得到的程序在X86平台上不能运行，必须放到ARM平台上才能运行；
  - 用来编译这种程序的编译器就叫交叉编译器；
  - 为了不与本地编译器混淆，交叉编译器的名字一般都有前缀，例如：arm-linux-gcc。

交叉编译 VS 本地编译

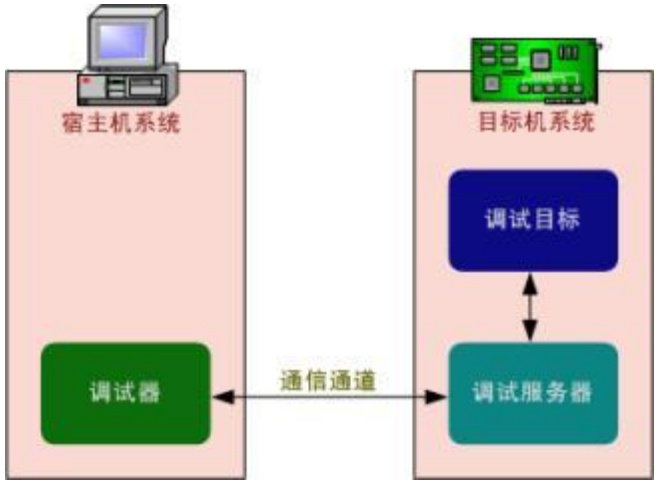
- 交叉编译器和交叉链接器是指能够在宿主机上安装，但是能够生成在目标机上直接运行的二进制代码的编译器和链接器

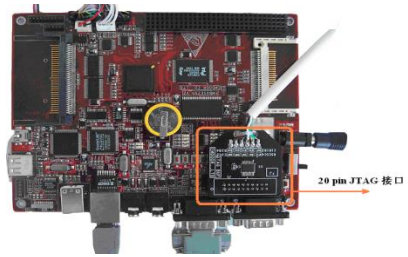


- 基于ARM体系结构的gcc交叉开发环境中，arm-linux-gcc是交叉编译器，arm-linux-ld是交叉链接器

交叉调试概述

- 一般而言，嵌入式软件需要交叉调试。





宿主机 **目标机**

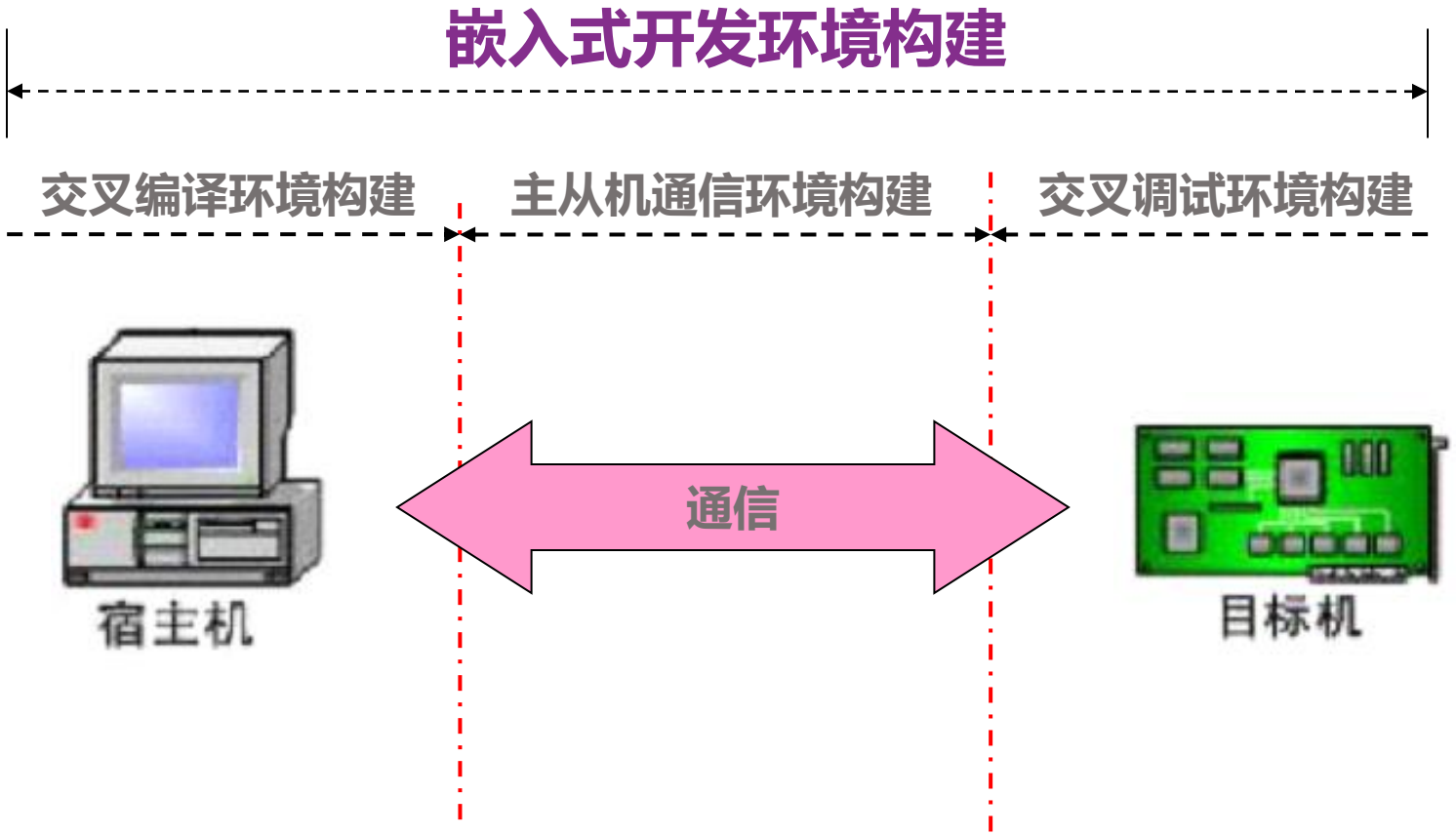
交叉调试 VS 本地调试

交叉调试	本地调试
Debugger和Debuggee运行在不同的计算机	Debugger和Debuggee运行在同一台计算机
需要运行时操作系统的调试支持（不一定）	需要运行时操作系统的调试支持
Debuggee的装载由Debugger（或Loader）完成	Debuggee的装载由专门的Loader程序完成
需要通过外部通信的方式来控制Debuggee	不需要通过外部通信的方式来控制Debuggee
可以调试不同指令集的程序	只能直接调试相同指令集的程序

交叉开发环境

- 需要交叉开发环境（Cross Development Environment）的支持，是嵌入式软件开发的一个显著特点。
- 交叉编译器只是交叉开发环境的一部分，完整的交叉开发环境是指包含交叉编译、交叉链接、交叉调试在内的嵌入式应用软件开发环境。

嵌入式开发环境构建



宿主机端的仿真

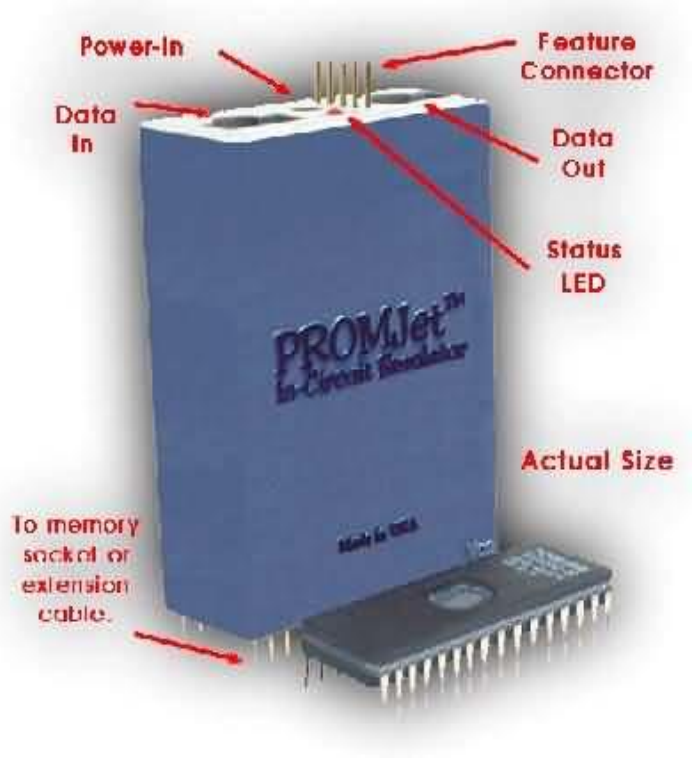
- 嵌入式应用开发中会出现宿主机操作系统（如Windows）与交叉开发环境中要求的宿主机操作系统（如Linux）不一致，因此，需要利用虚拟化、仿真化手段建立开发环境，包括：
  - API仿真器：Cygwin、MinGW
  - 虚拟机：Virtual PC、VMWare、 Virtualbox

目标机端的仿真

- 嵌入式应用的开发经常会遭遇缺少目标机环境、缺乏目标机芯片等资源的问题，因此提出了根据不同的应用需要，利用仿真器件、仿真环境进行开发的方法，包括：
- 硬件仿真开发
  - ROM Emulator
  - ICE
  - OCD
- 软件仿真开发

硬件仿真开发——ROM Emulator

- ROM Emulator可用于替代目标机上的ROM芯片。利用该设备，目标机可没有ROM，但目标机的CPU可读取ROM Emulator上ROM的内容。
- ROM Emulator的ROM地址可实时映射到目标机ROM地址空间，从而仿真目标机的ROM。



硬件仿真开发——ICE

- ICE（In-Circuit Emulator）是一种用于替代目标机上CPU的设备，即在线仿真器。
- 它比一般的CPU有更多的引出线，能够将内部的信号输出到被控制的目标机。
- ICE上的Memory也可以被映射到用户的程序空间，即使目标机不存在，也可进行代码的调试。
- ICE可支持软断点和硬件断点的设置、设置各种复杂的断点和触发器、实时跟踪目标程序的运行等。

硬件仿真开发——OCD

- OCD（On Chip Debugging）是CPU芯片提供的一种调试功能（片上调试），可以认为是一种廉价的ICE功能。
- OCD不占用目标机资源，调试环境和最终目标机运行环境基本一致，支持软硬断点、Trace功能，可提供精确计量程序的执行时间、时序分析等功能。

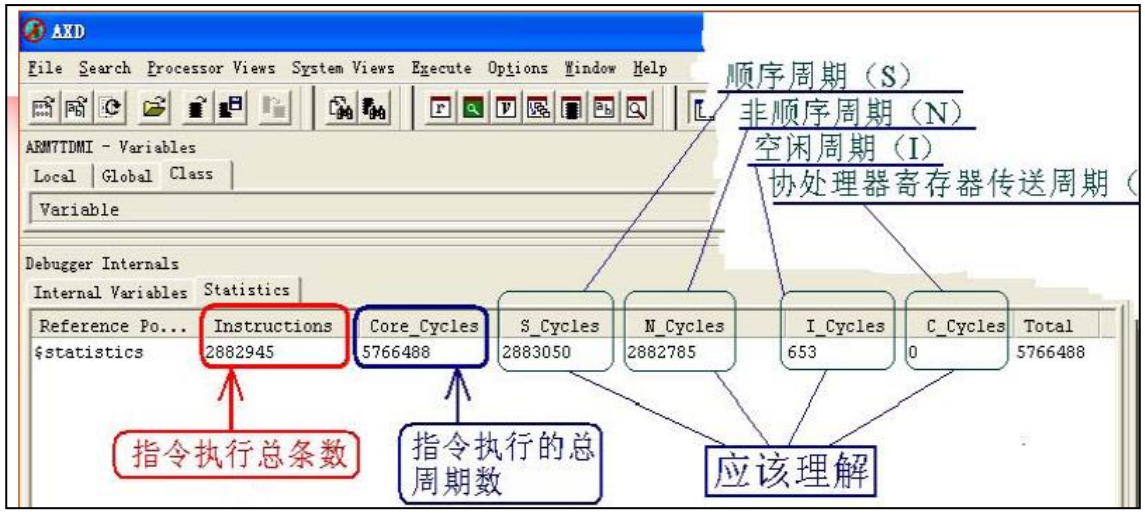
VITRA shown here with both debug and trace connections to ARM™ Integrator/CM-966E-S development board.

软件仿真开发

- 以软件仿真的方式在宿主机上创建一个虚拟的目标机环境，再将应用系统下载到这个虚拟目标机上运行 / 调试
- 仿真精度
  - 指令级
  - 周期级
  - 时序（节拍）级
- 仿真能力
  - 指令集仿真
  - 全系统仿真

软件仿真器举例

- ARMulator：ARMulator 是一个 ARM 公司推出的集成开发环境 ADS （ARM Developer Suite）中提供的指令集模拟器。ARMulator 不仅可以仿真 ARM 处理器的体系结构和指令集，还可以仿真存储器和处理器外围设备。

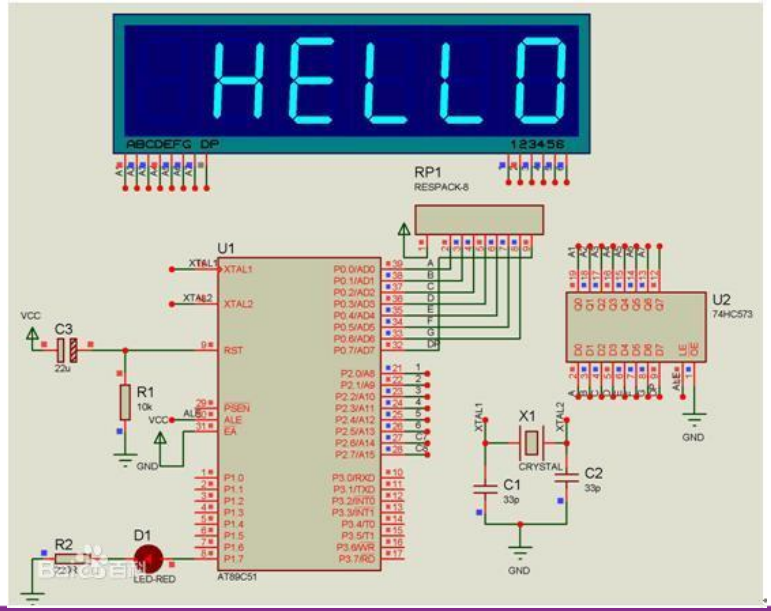


软件仿真器举例

- SkyEye: SkyEye是一个开源软件项目，中文名字是“天目”。SkyEye的目标是在通用的Linux和Windows平台上实现一个纯软件集成开发环境，模拟常见的嵌入式计算机系统;可在SkyEye上运行μCLinux以及μC/OS-II等多种嵌入式操作系统和各种系统软件（如TCP/IP，图形子系统，文件子系统等），并可对它们进行源码级的分析和测试。

软件仿真器举例

- Proteus: Proteus是英国著名的EDA工具(仿真软件)，是世界上唯一将电路仿真软件、PCB设计软件和虚拟模型仿真软件三合一的设计平台，还可以直接在基于原理图的虚拟原型上编程，再配合显示及输出，能看到运行后输入输出的效果。



Linux本地软件开发模式

1、代码编写

```
› vim debug.c
```

```c
#include <stdio.h>

int power(int, int);

int main() {

  int i;
  printf("Program to calculate power\n");
  for (i = 0; i < 10; i++)
    printf("%d %d\n", i, power(2, i));
  return 0;
}
```

2、程序编译

```
› gcc -g debug.c -o debug
```

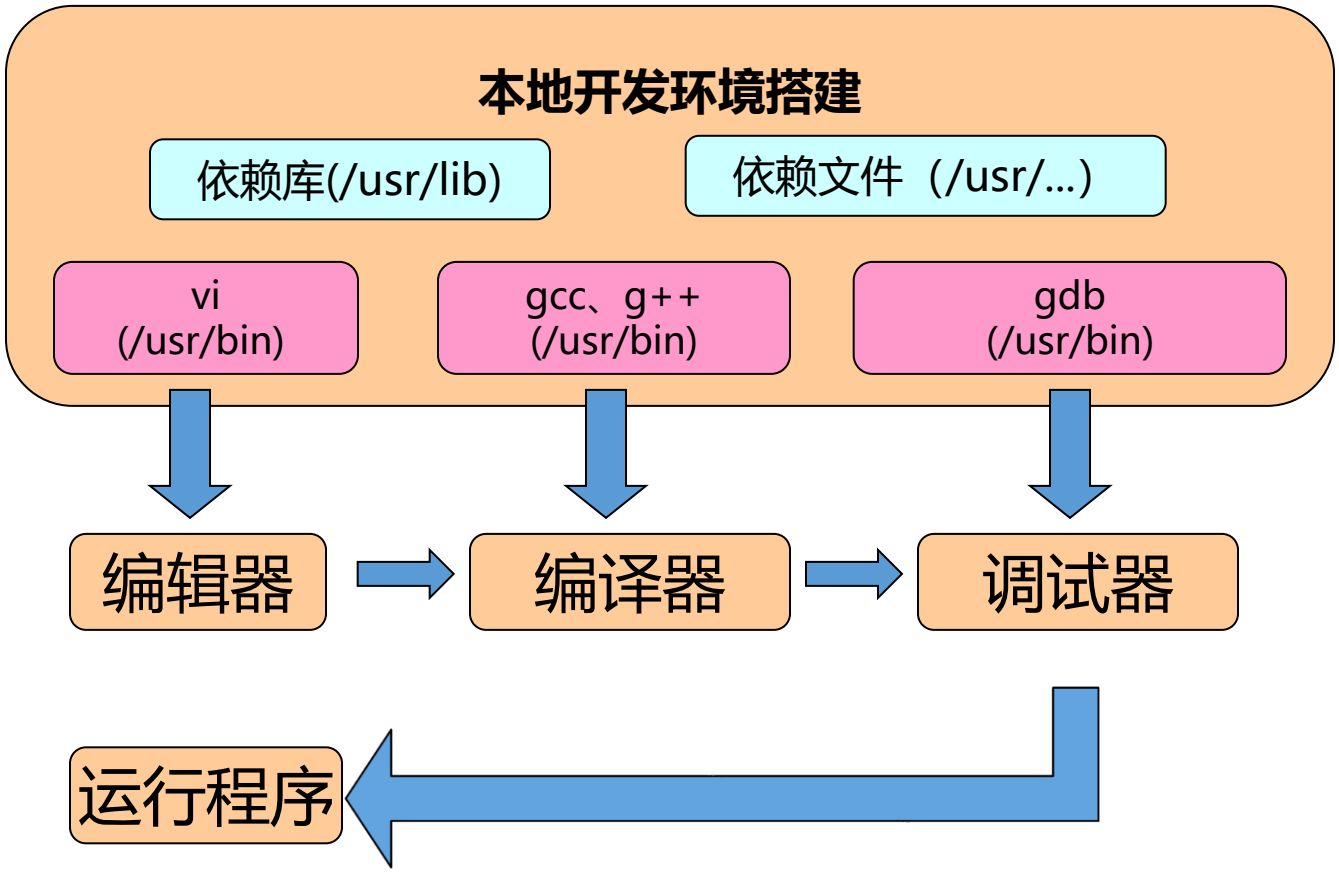
3、程序运行

```
› ./debug
```

4、程序调试

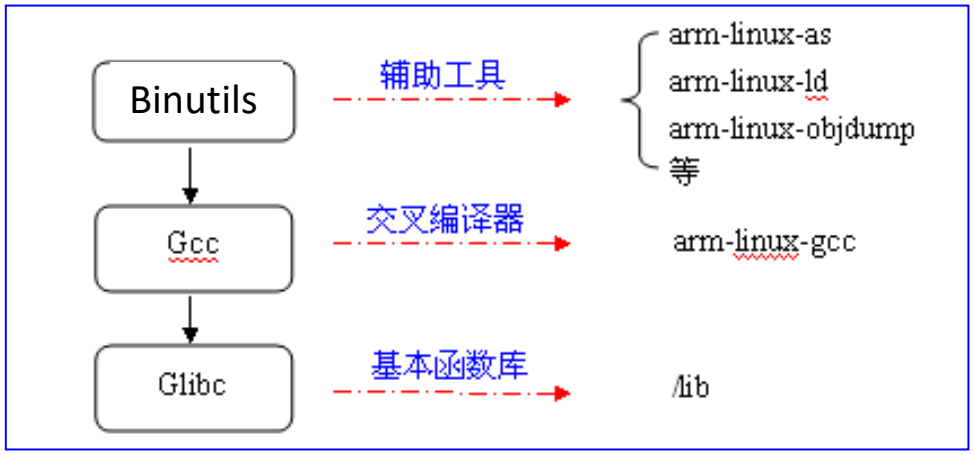
```
› gdb debug
```

Linux本地软件开发环境



构建交叉编译环境

- 构建交叉编译环境所需的工具链主要包括：
  - 交叉编译器，例如大作业使用的arm-ostl-linux-gnueabi-gcc
  - 交叉汇编器，例如arm-linux-as
  - 交叉链接器，例如arm-linux-ld
  - 用于处理可执行程序和库的一些基本工具，例如arm-linux-strip



名称	归属	作用
arm-linux-as	binutils	编译 ARM 汇编程序
arm-linux-ar	binutils	把多个.o 合并成一个.o 或静态库(.a)
arm-linux-ranlib	binutils	为库文件建立索引，相当于 arm-linux-ar -s
arm-linux-ld	binutils	连接器(Linker), 把多个.o 或库文件连接成一个可执行文件
arm-linux-objdump	binutils	查看目标文件(.o)和库(.a)的信息
arm-linux-objcopy	binutils	转换可执行文件的格式
arm-linux-strip	binutils	去掉 elf 可执行文件的信息，使可执行文件变小
arm-linux-readelf	binutils	读 elf 可执行文件的信息
arm-linux-gcc	gcc	编译.c 或.S 开头的 C 程序或汇编程序
arm-linux-g++	gcc	编译 c++程序

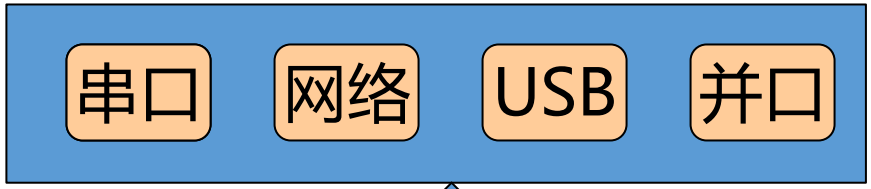
生成交叉编译器

- 为什么需要自行生成交叉编译器?
  - 针对特定的嵌入式体系结构，不一定有现成的交叉编译器，因而，我们不得不使用现有的GCC代码来生成交叉编译器!

生成交叉编译器

- 交叉编译器的生成过程
  - 制作交叉的binutils二进制工具
  - 制作不带库的gcc交叉编译器
  - 用制作好的gcc交叉编译器，生成所需要的C库（glibc、newlib、uclibc等）
  - 重新编译带库的gcc，生成完整的交叉编译器

主从机通信环境构建

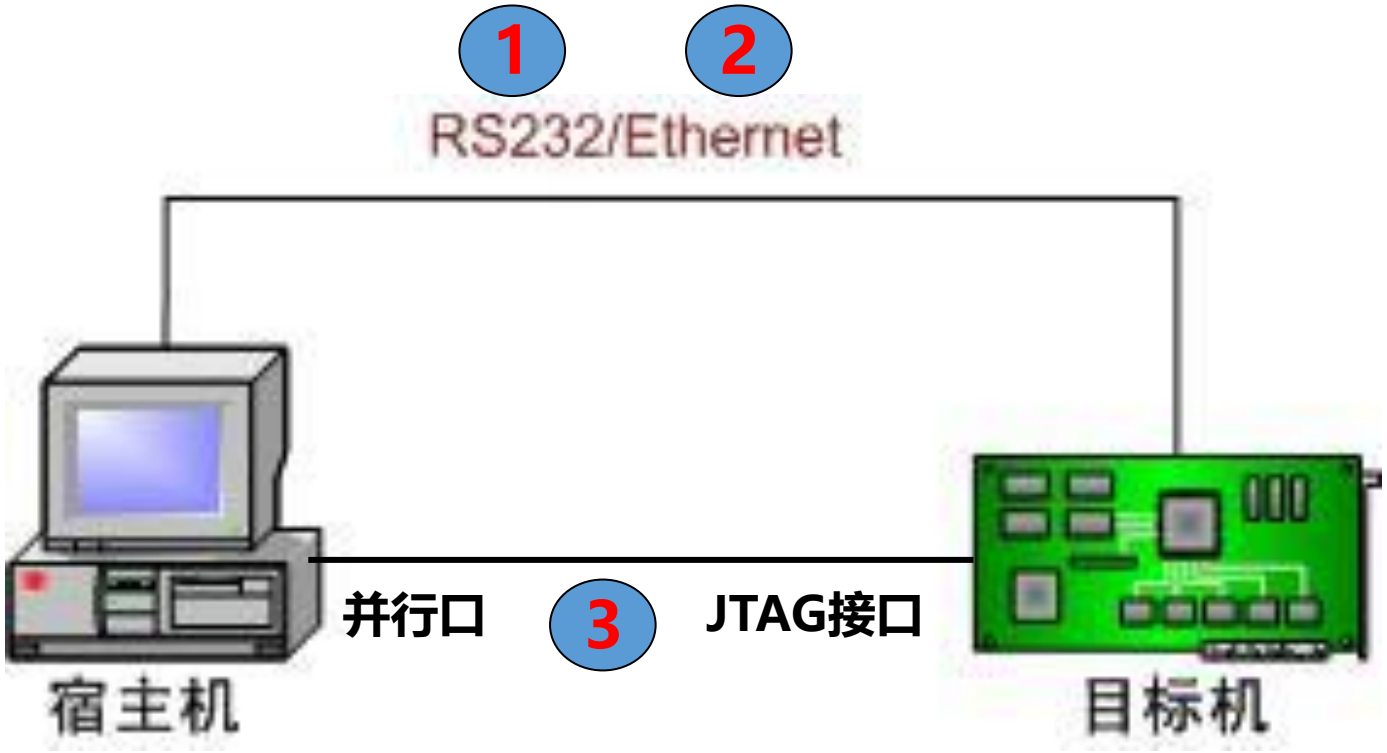


如何建立通信连接?

如何下载程序?

如何访问宿主机?

3种常见通信模式



通讯模式1 - 串口通讯

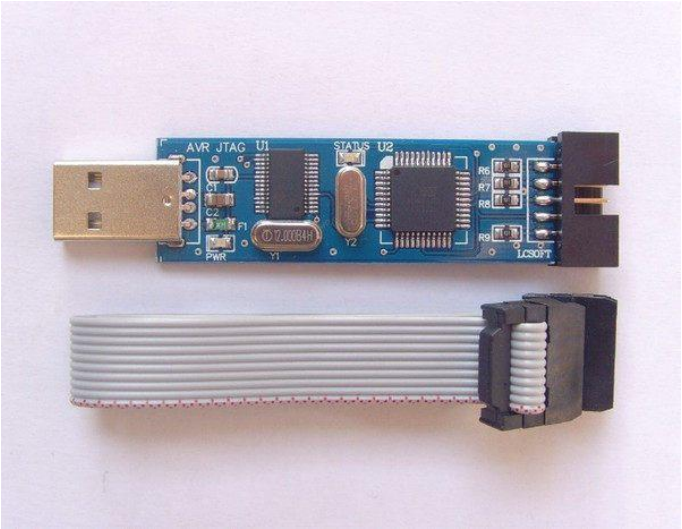
- 特点及应用场合
  - 驱动实现最简单
  - 传输速度慢，距离短，不适合大数据量、长距离数据传输
  - 需要在宿主机、目标机两端均提供驱动
  - 常用于宿主机 - 目标机的字符流通讯

通讯模式2 - 网络通讯

- 特点及应用场合
  - 驱动实现相对复杂，一般采用精简的网络通讯协议，如TFTP进行通讯
  - 常用于宿主机 - 目标机的大数据量数据传输，可以作为串口通讯的补充
  - 需要在宿主机、目标机两端均提供驱动
  - 宿主机端实现服务器，目标机端提供客户端

通讯模式3 - JTAG简介

- JTAG(Joint Test Action Group)是1985年制定的检测PCB和IC芯片的一个标准。
- 1990年被修改后成为IEEE的一个标准，即IEEE1149.1-1990。
- 通过这个标准，可以对具有JTAG接口的芯片硬件电路进行边界扫描和故障检测。



提纲

- 嵌入式软件开发概述
- 通用软件调试技术概述*
- 嵌入式软件调试技术

嵌入式系统调试概述

- 嵌入式系统开发过程中常见的错误和通用PC机上软件开发中常见错误有着较大的区别。
- 常见错误分类如下
  - 硬件相关错误
  - 通讯相关错误
  - 多任务相关错误
  - 用户界面相关错误
  - 内存访问相关错误

硬件相关错误

- 嵌入式软件开发过程中，常出现硬件相关错误，例如：
  - WYNOT错误（“Worked Yesterday, NOt Today”）
  - 评估板上运行正确、现场运行错误
  - 低频运行正确、高频运行错误
  - ……
- 对于仅经历过纯软件编程培训，对数字电路知识之至甚少的软件工程师而言，首先需准确区分这些错误是由于软件设计产生的错误，还是由于硬件问题所导致。
- 对专用集成电路（ASIC）理解错误：
  - 寄存器设置、高速缓存（cache）控制、中断/内存控制
- 与信号干扰有关

通讯相关错误

- 由于嵌入式设备需要与PC、各种专用设备，以及其他嵌入式子系统进行通信，可能采用标准的协议：
  - USB
  - UART
  - PCI
  - ……
- 也可能采用各种专用的协议，出现与标准协议不兼容等异常情况。

多任务相关错误

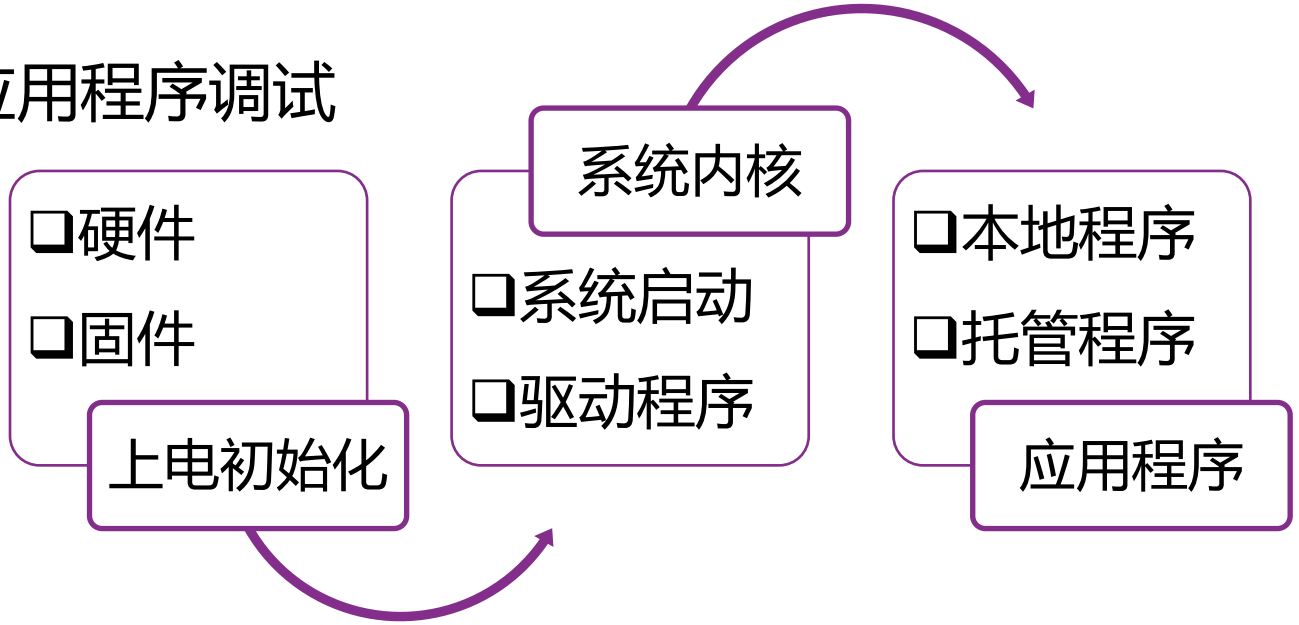
- 由于多任务开发中的缺陷所产生的问题，是嵌入式系统开发中最难解决的问题之一，直接影响到系统的可靠性、健壮性、执行效率和可维护性。
- 嵌入式多任务环境下可能出现的问题包括：
  - 划分问题：任务～中断
  - 优先级设置问题：任务～任务、任务～中断
  - 同步问题：中断～任务、任务～任务
  - 互斥问题：中断～任务、任务～任务
  - 通讯问题：中断～任务、任务～任务
  - 异常处理问题：出错与恢复、执行任务取消

其他错误

- 用户界面相关错误
  - 由于输入、输出设备的多样性，所导致的错误
  - 表现直观，是最好解决的一类错误。
- 内存访问相关错误
  - 非法访问：非法修改内核、其他任务的内存空间
  - 内存碎片问题：无法进行内存碎片整理，导致空闲内存总数够，但连续内存数目不够情况。

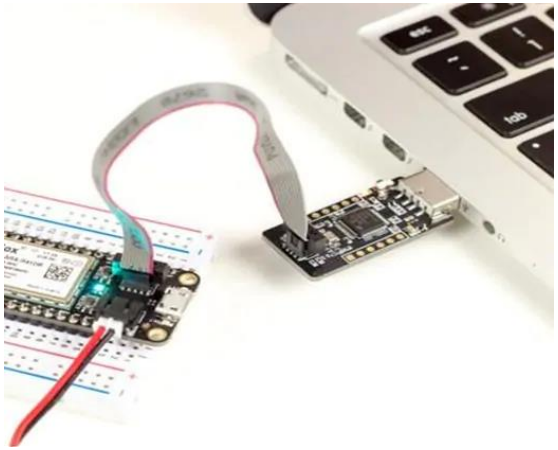
嵌入式系统调试的三个层次

- 从嵌入式系统开发和执行过程看，嵌入式系统调试可以分为三个层次
  - 上电（power-on）初始化调试：硬件、固件
  - 系统内核调试：bootloader、设备驱动程序、操作系统
  - 应用程序调试



上电初始化调试概述

- 系统上电和处理器复位后，首先执行的是固化在ROM或者flash上的硬件初始化程序，即通常所说的固件（firmware）。
- 在这一阶段，因为运行环境很简陋，软件调试器还无法工作，所以比较有效的一种调试手段是使用基于JTAG协议的硬件调试器（JTAG调试器）。

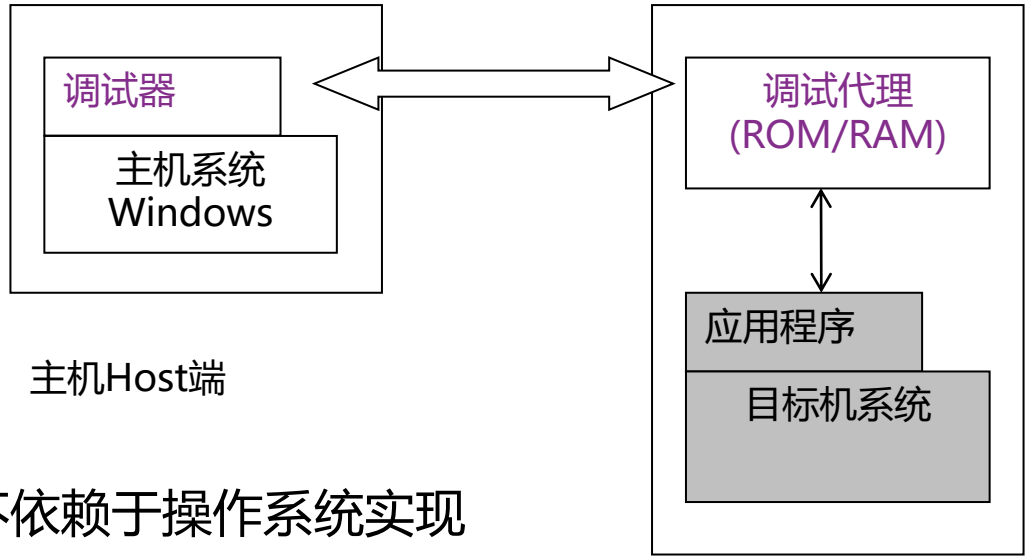


远程调试特点

- 调试器和被调试程序运行在不同机器上
- 目标机上具备某种形式的调试代理（如gdbserver）
- 符号表驻留在主机端，目标机端工作在无符号状况下
- 调试器通过某种端口（串口、网络、JTAG），按预定的通信协议与目标机端的调试代理建立联系
- 支持ROM型代码（即只读代码）断点调试

远程调试模式 - 系统级调试

- 根据调试器对被调试程序的调试能力分类，可以把远程调试器模型分为：系统级调试和任务级调试，下面是系统级调试原理示意图：



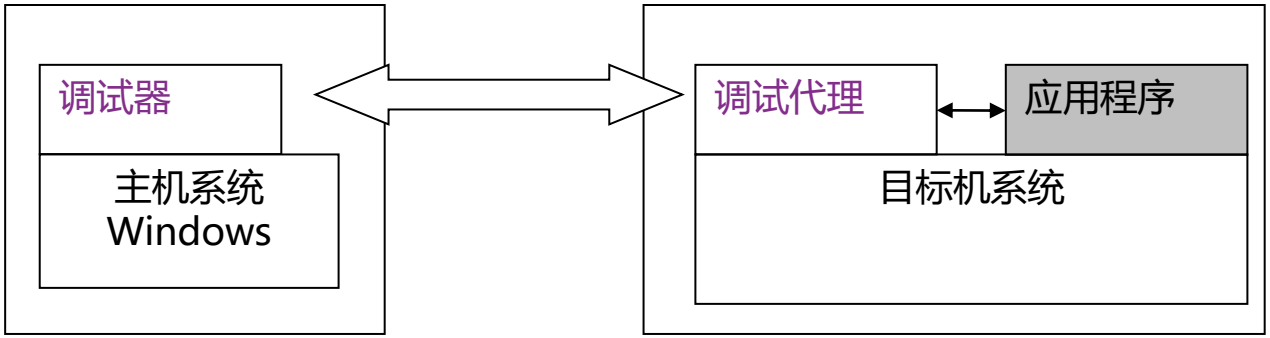
主机Host端

目标机Target端

- 特点：
  - 调试器不依赖于操作系统实现
  - 调试器直接接管调试异常
  - 可以调试操作系统内核、ISR
  - 但无法调试单个任务

远程调试模式 - 任务级调试

- 特点：
  - 调试器基于操作系统功能实现
  - 操作系统接管调试异常，再转交调试器
  - 可以调试多任务情况
  - 无法调试操作系统内核、ISR



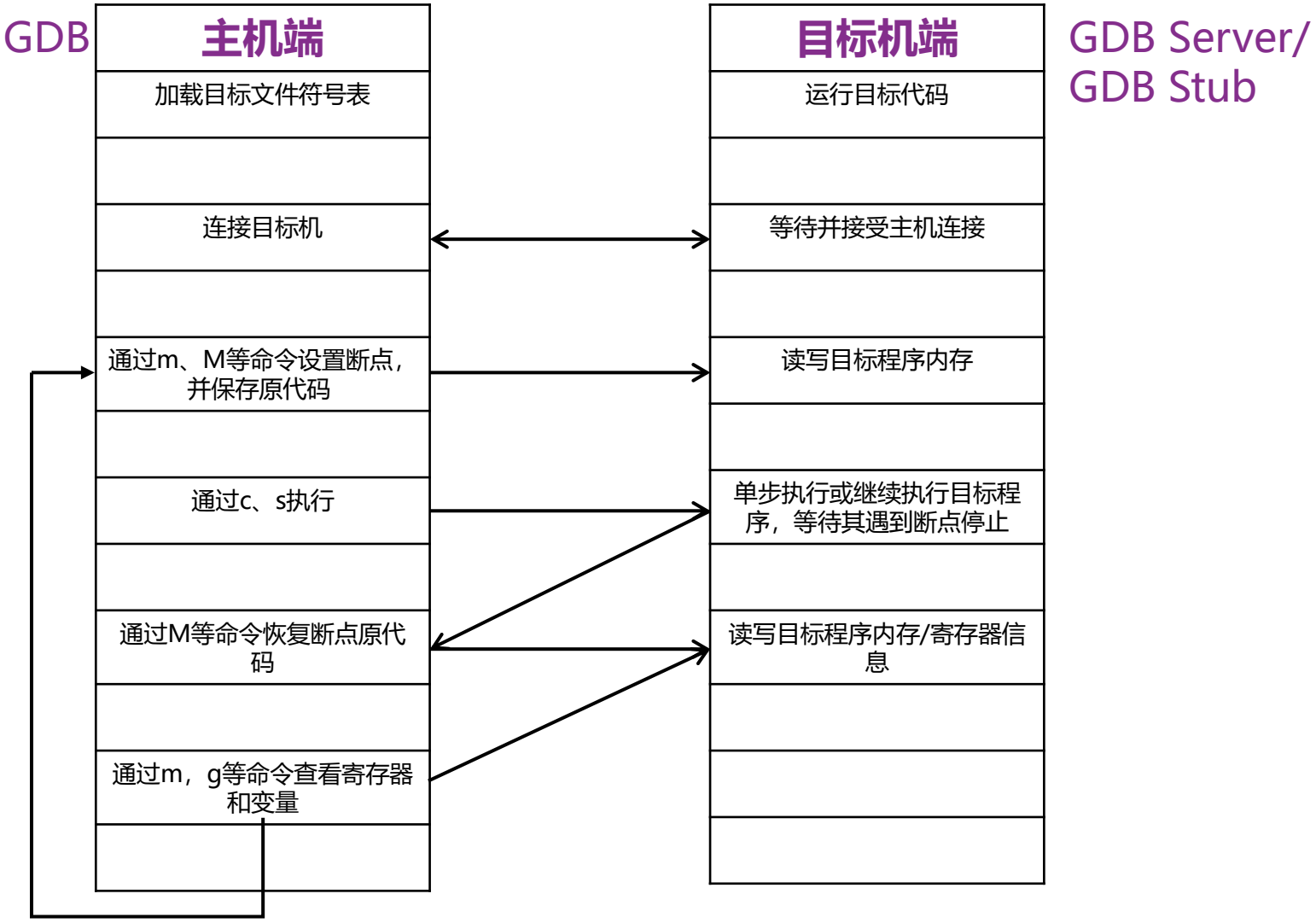
GNU调试器GDB简介

- 使用GDBServer进行调试的过程中，需要用到GDB相关知识，先来学习一下GDB调试工具
- GDB（Gnu DeBugger）是GNU C自带的调试工具，使用GDB可以完成下面这些任务：
  - 支持a.out、coff、ecoff、xcoff、elf、pe格式；
  - 支持条件断点（如循环次数、条件表达式）设置；
  - 提供运行状态查看（变量、寄存器）；
  - 提供调试阶段的数据修改功能;
  - ……

GDB远程调试解决方案

- GDB为远程调试提供了两种解决方案：GDB Server、GDB Stub
  - GDB Server对Unix/Linux操作系统具有依赖性，只能完成应用程序级调试 - 任务级调试；
  - GDB Stub不依赖于操作系统，可对整个系统进行调试 - 系统级调试。

GDB远程调试流程


GDB
主机端
加载目标文件符号表
连接目标机
通过m、M等命令设置断点，并保存原代码
通过c、s执行
通过M等命令恢复断点原代码
通过m，g等命令查看寄存器和变量
目标机端
运行目标代码
等待并接受主机连接
读写目标程序内存
单步执行或继续执行目标程序，等待其遇到断点停止
读写目标程序内存/寄存器信息
GDB Server/
GDB Stub

任务级调试—GDB Server

- GDB Server能够远程调试类Unix系统下的应用程序。
- GDB Server通过类Unix系统提供的ptrace系统调用来实现对被调试应用程序的访问和控制。
- GDB Server对操作系统具有依赖性，导致它无法实现操作系统级调试，只能实现应用程序级调试。
- GDB Server的优势在于：应用程序代码无需与调试模块链接就能实现远程调试。

GNU调试器GDB简介

- 启动GDB
  - 键入gdb gdb_test命令来启动GDB并载入程序gdb_test，命令行进入了GDB模式。
  - 命令补齐功能。

```
> gdb gdb_test

(gdb) >
```

GNU调试器GDB简介

❑GDB中的常用命令如下：

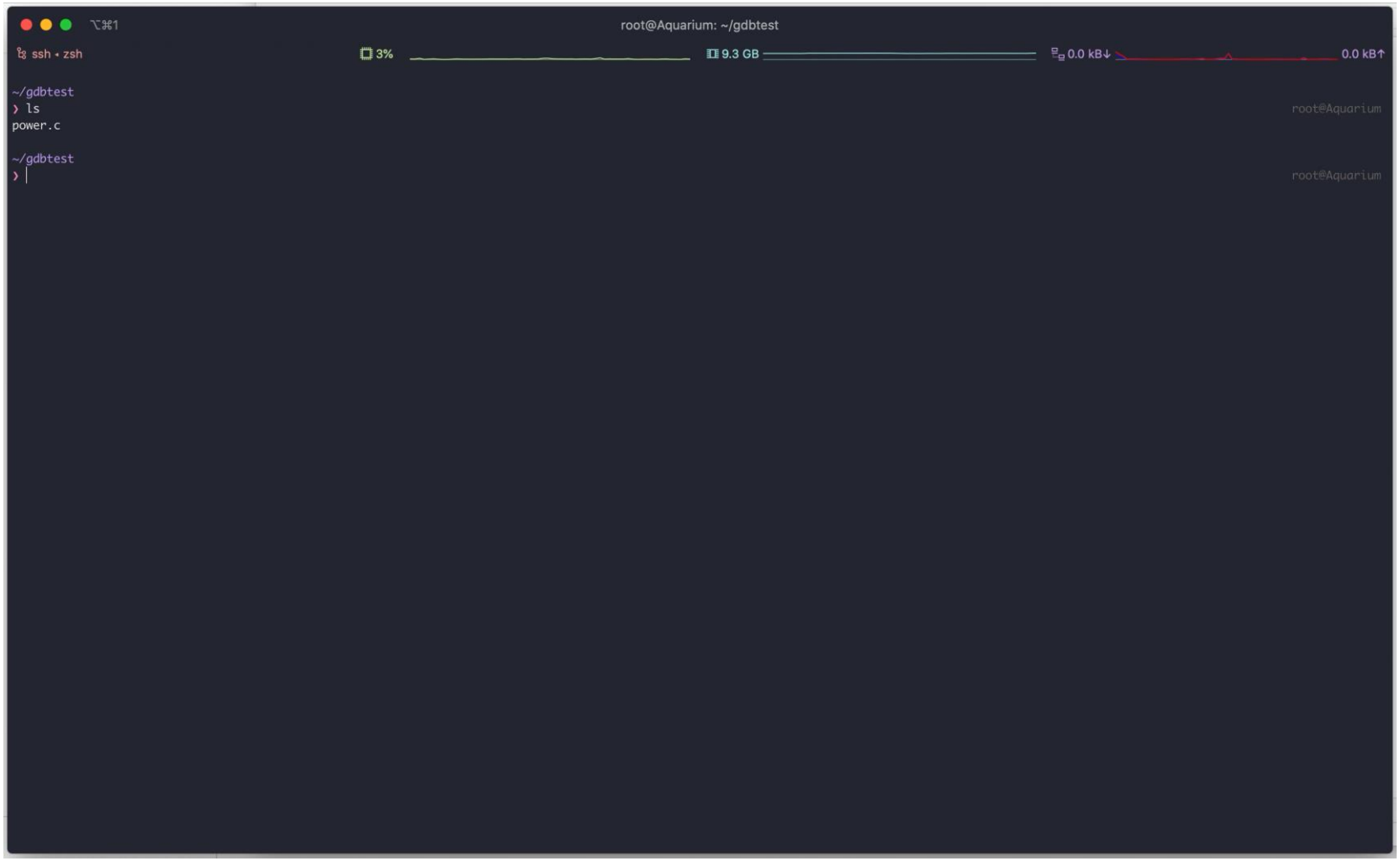
指令	说明
file	载入程序。如file hello。当然，程序的路径名要正确。
quit	退出GDB。也可以输入'C-d'来退出GDB。
run	执行载入后的要调试的程序。可以输入参数。
info	查看程序的信息。多用来查看断点信息。可以用help info来查看具体帮助。 info source查看当前文件的名字，路径，所使用的程序语言等信息。 info stack 查看调用栈。 info local 查看局部变量信息。 info br　br是断点break的缩写，用这条指令，可以得到所设置的所有断点的详细信息。
list	list FUNCTION列出被调试程序某个函数 list LINENUM以当前源文件的某行为中间显示一段源程序 list　接着前一次继续显示 list -　显示前一次之前的源程序 list FILENAME:FUNCTION显示另一个文件的一段程序，

GNU调试器GDB简介

break	最常用和最重要的命令：设置断点。break FUNCTION在函数入口设置断点 break LINENUM在当前源文件的某一行上设置断点 break FILENAME:LINENUM在另一个源文件的某一行上设置断点 break *ADDRESS在某个地址上设置断点
watch	监视某个表达式或变量，当它被读或被写时让程序中断。格式如下： watch EXPRESSION
set	修改变量值。格式如下： set variable=value
step	单步执行，进入遇到的函数。
next	单步执行，不进入函数调用，即视函数调用为普通语句。
continue	恢复中断的程序执行。
help	通过下面的方法获得帮助，下例为获得list指令。 help list

- 接下来我们通过Demo来详细学习各指令的用法

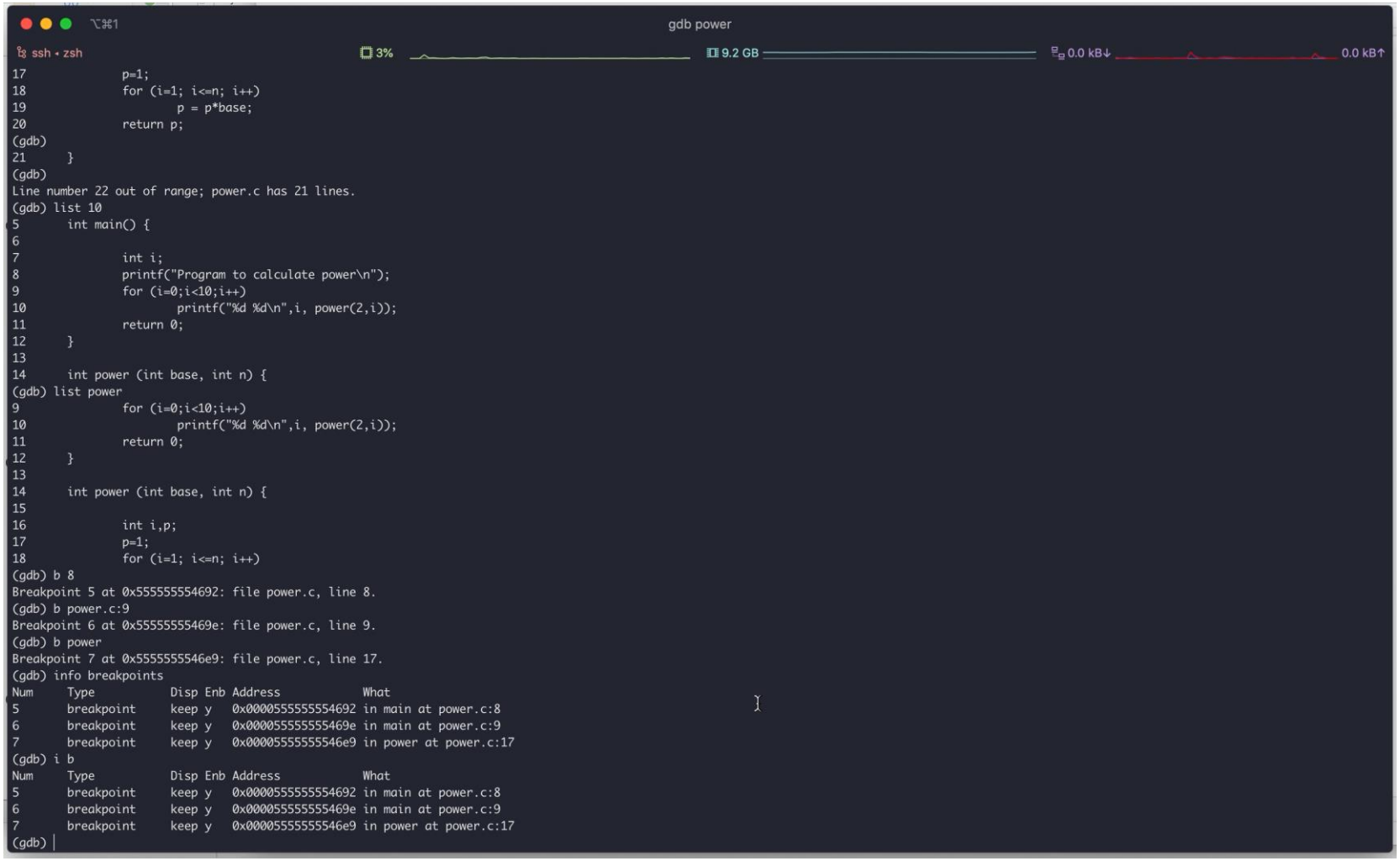
GDB Demo – 安装、编译、加载



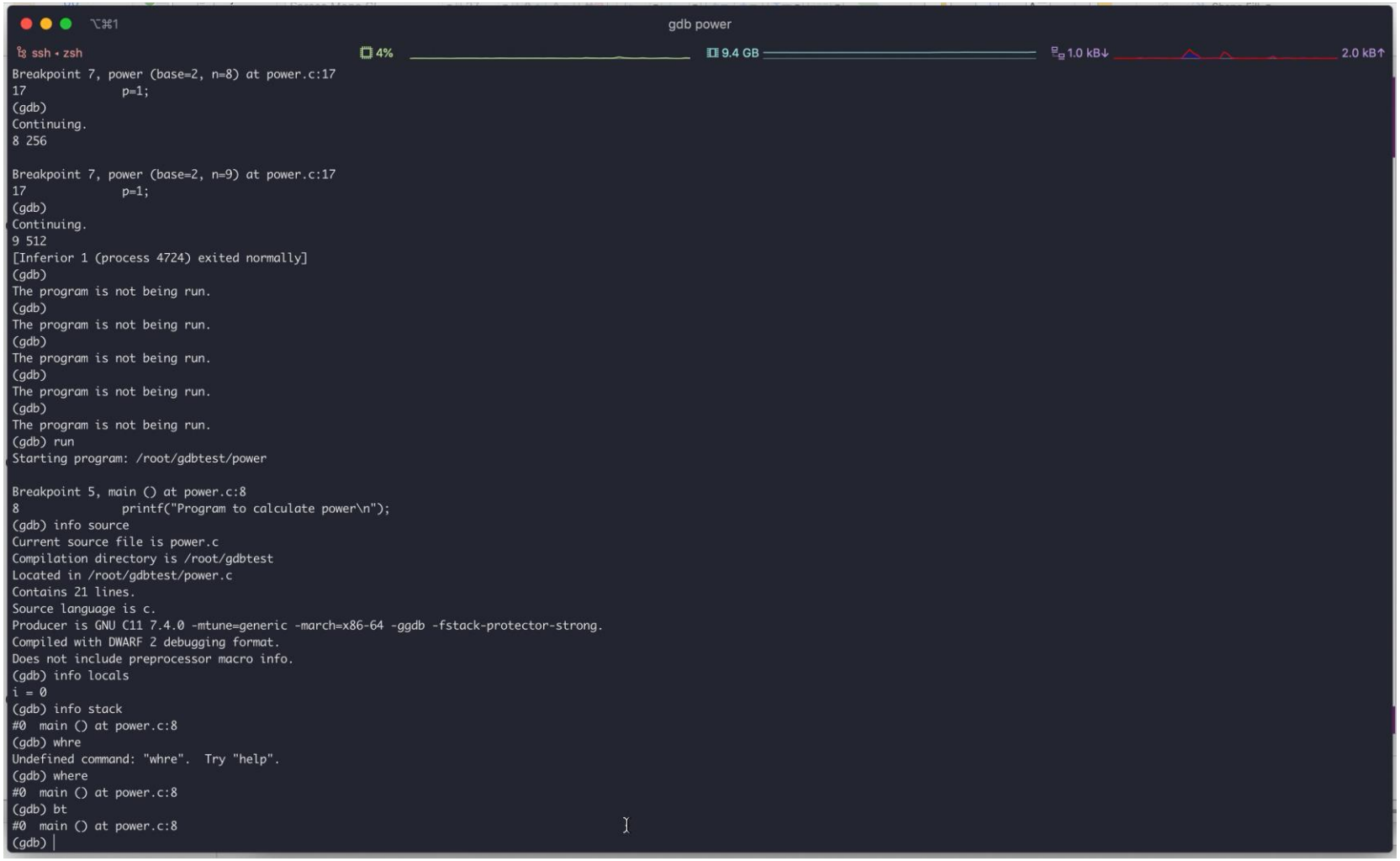
GDB Demo – List 指令

GDB Demo – Break 指令

GDB Demo – Info 指令



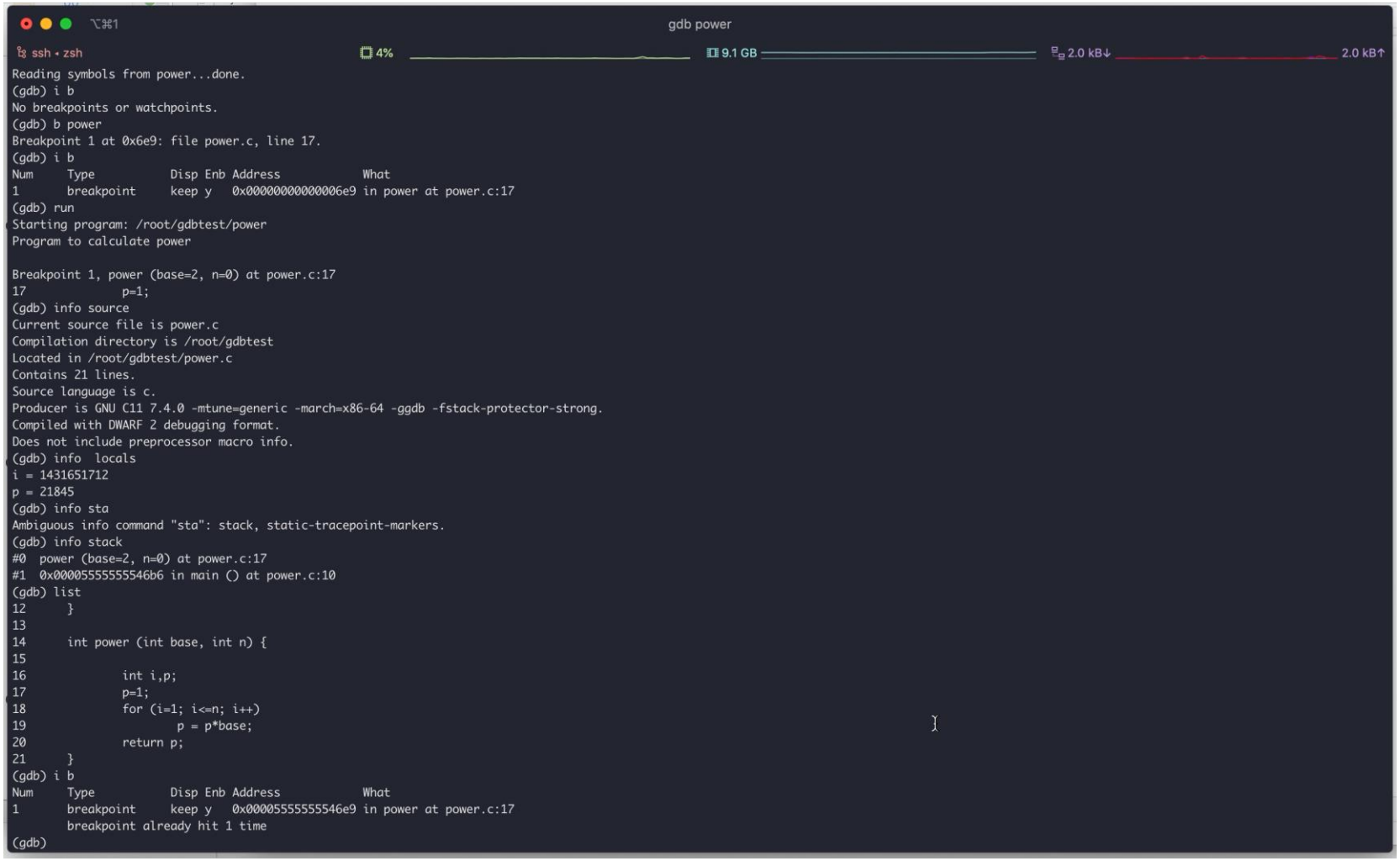
GDB Demo – Step/Next/Continue 指令



GDB Demo – Disable 指令

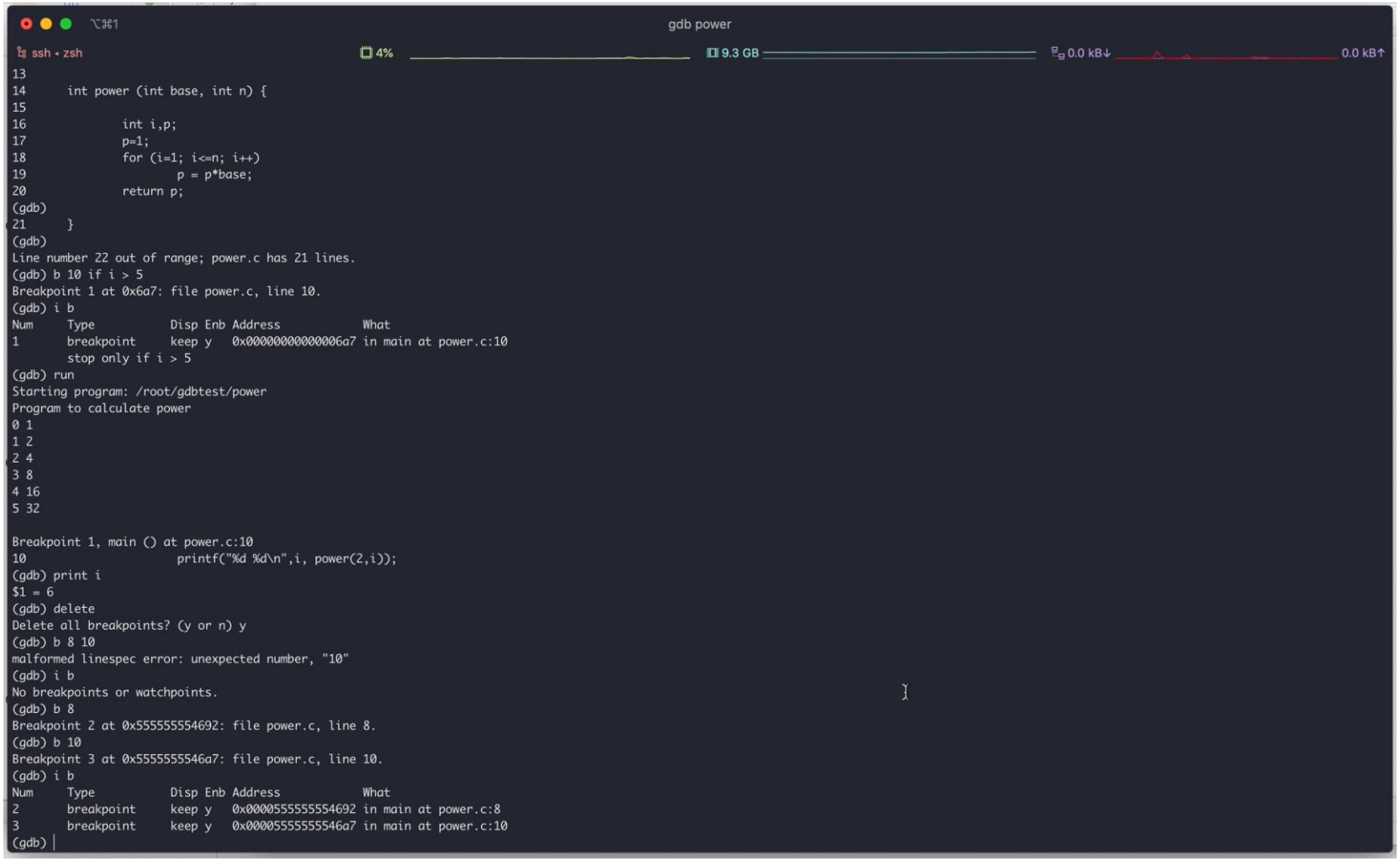
GDB Demo – Delete/Enable 指令

GDB Demo – Layout 指令

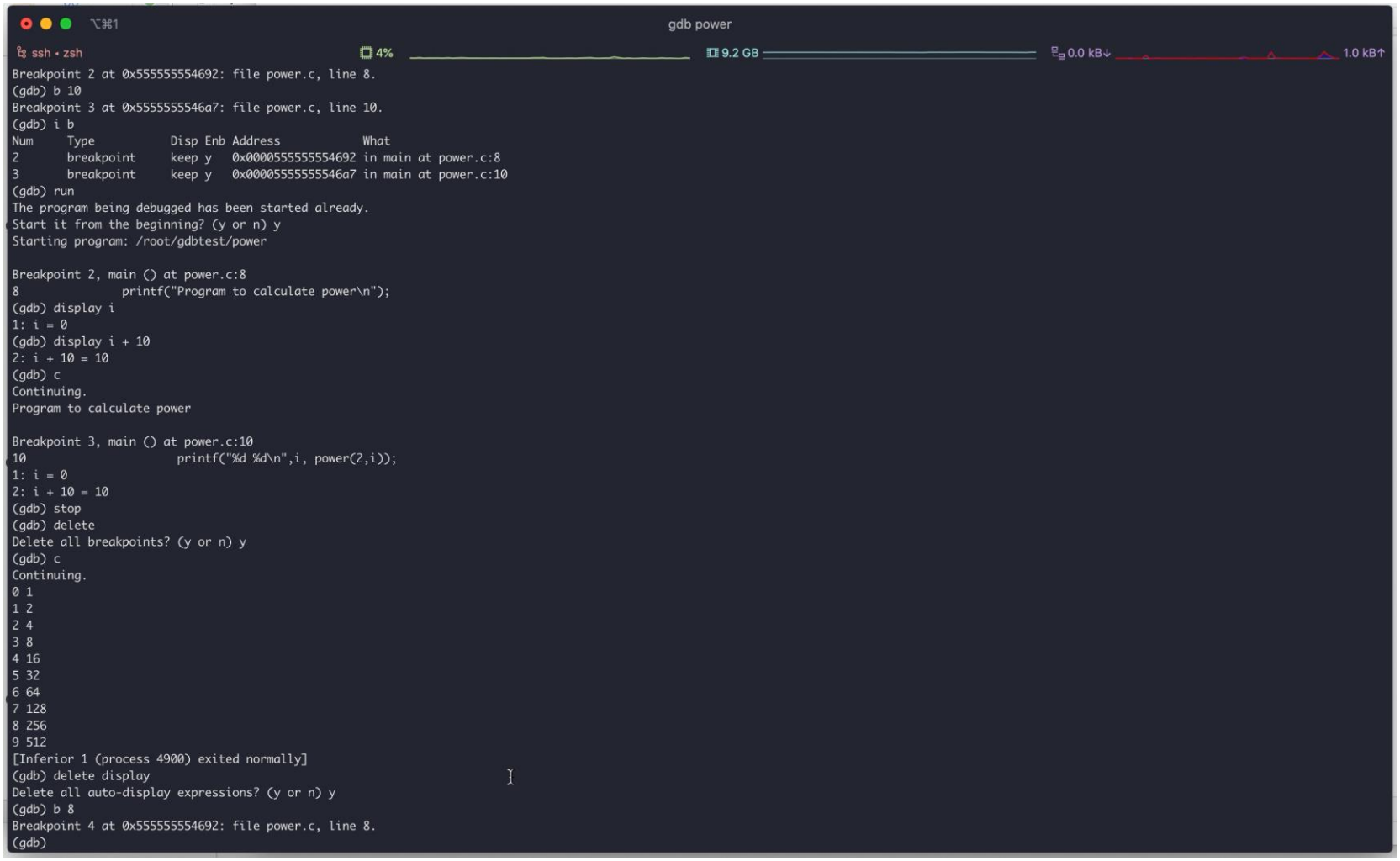


GDB Demo – 条件 Break 指令

GDB Demo – Display 指令



GDB Demo – Watch 指令

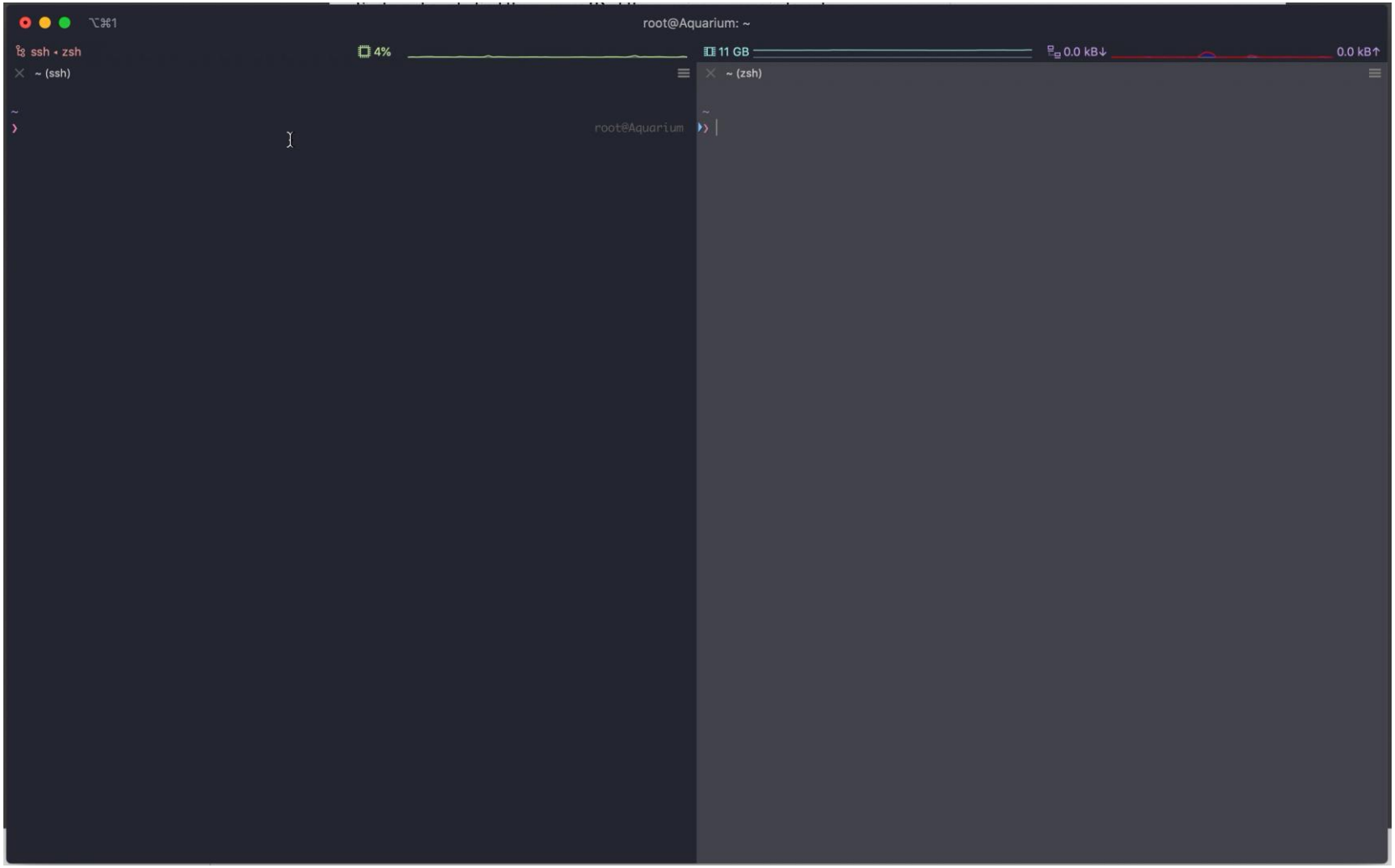


GDB Demo – Help 指令

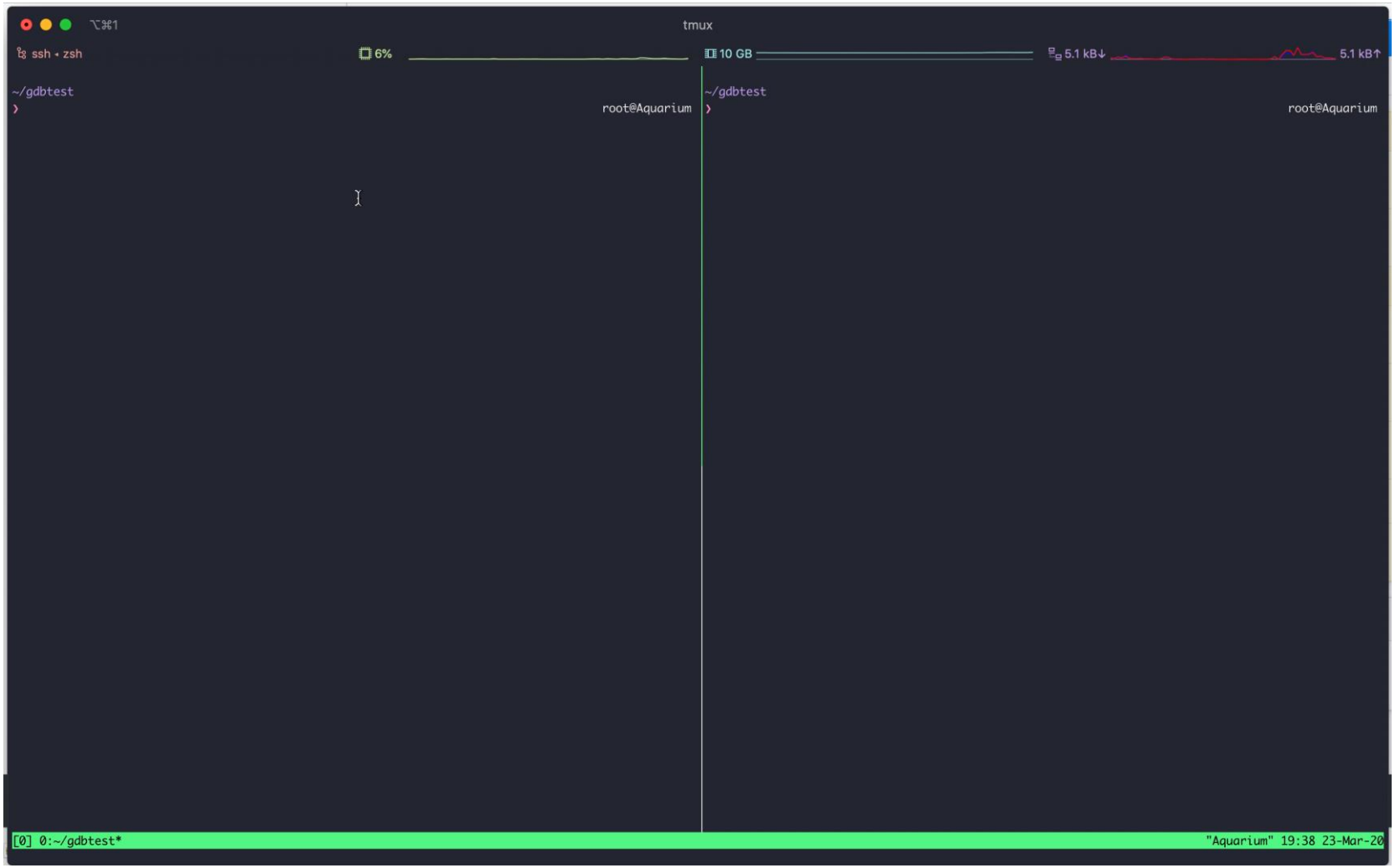
GDB Server主要工作流程

- GDB Server通过通讯端口与GDB建立连接。
- GDB Server与被调试程序绑定/关联：让被调试程序成为GDB Server的子进程
- GDB通过GDB Server设置断点等信息。
- 被调试程序运行产生异常，操作系统通知GDB Server接管。
- GDB通过读写被跟踪程序的指令空间、数据空间或寄存器，完成用户调试命令。

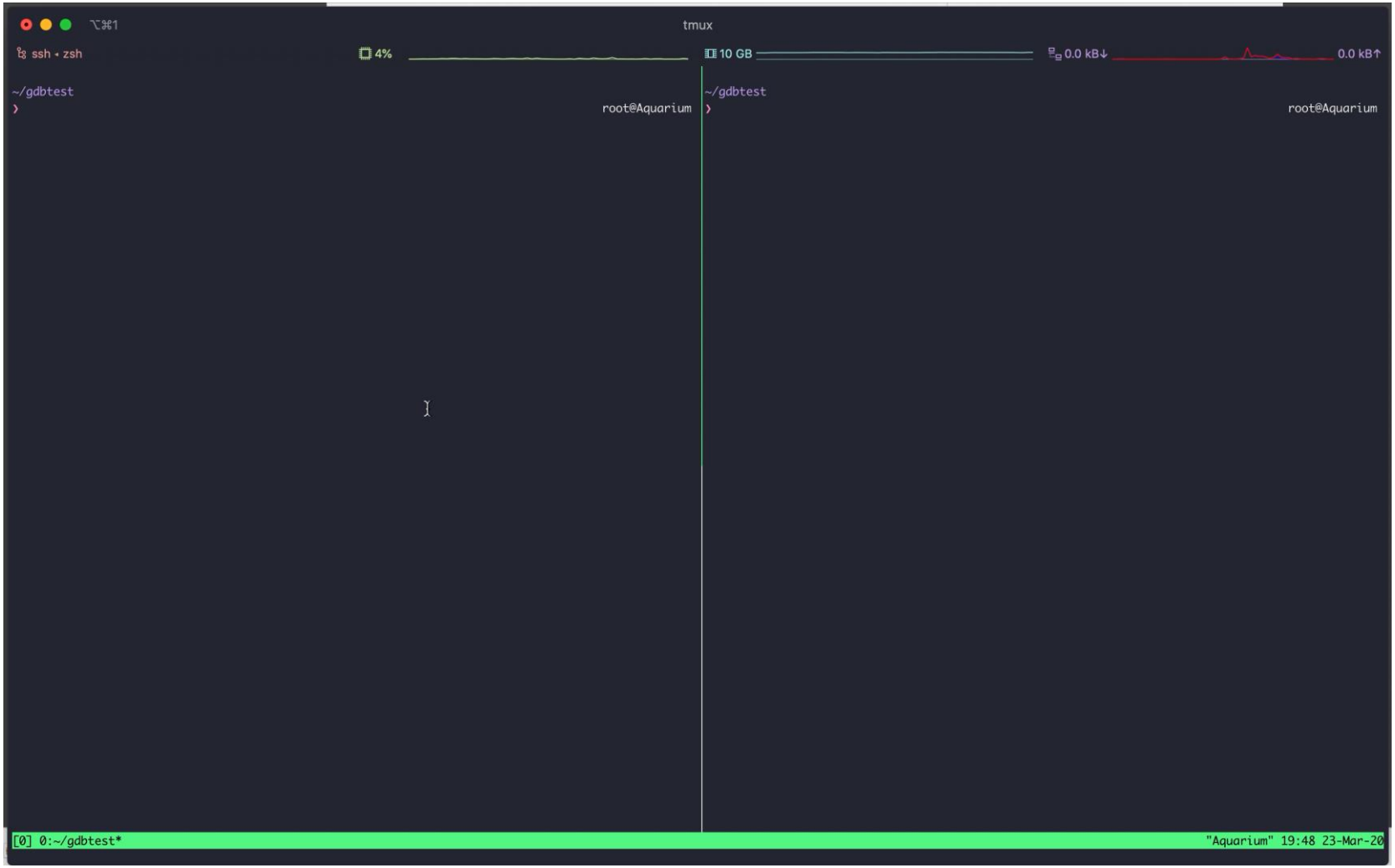
GDBServer Demo – Host/Target建立连接



GDBServer Demo – 本地模拟 Host/Target

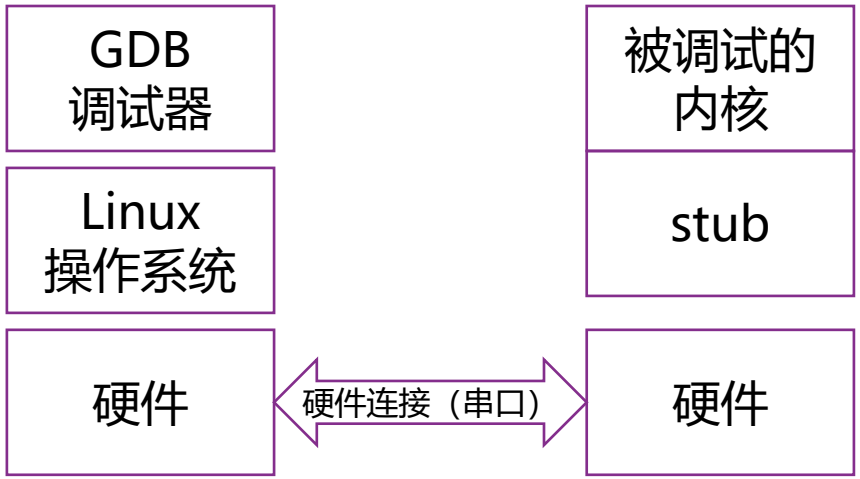


GDBServer Demo – 远程调试



GDB Stub 技术简介

- KGDB采用了插桩（stub）技术实现系统级调试。在目标操作系统内核中加入调试功能模块（插桩），如：通信模块、异常处理模块等。
- 插桩的基本思想是：接管操作系统的中断或者异常处理，使得当中断或者异常发生时，调试器获得整个系统的控制权，进而获取当前的调试信息，如：寄存器、程序运行状态等信息。



系统级调试—GDB Stub

- KGDB在Linux内核上提供调试器功能补丁，属于GDB Stub类型，是一种内核调试器，实现系统级软件调试，适用于下述情况：
  - 开发者需要调试操作系统内核
  - 开发者需要调试设备驱动模块

KGDB功能分析

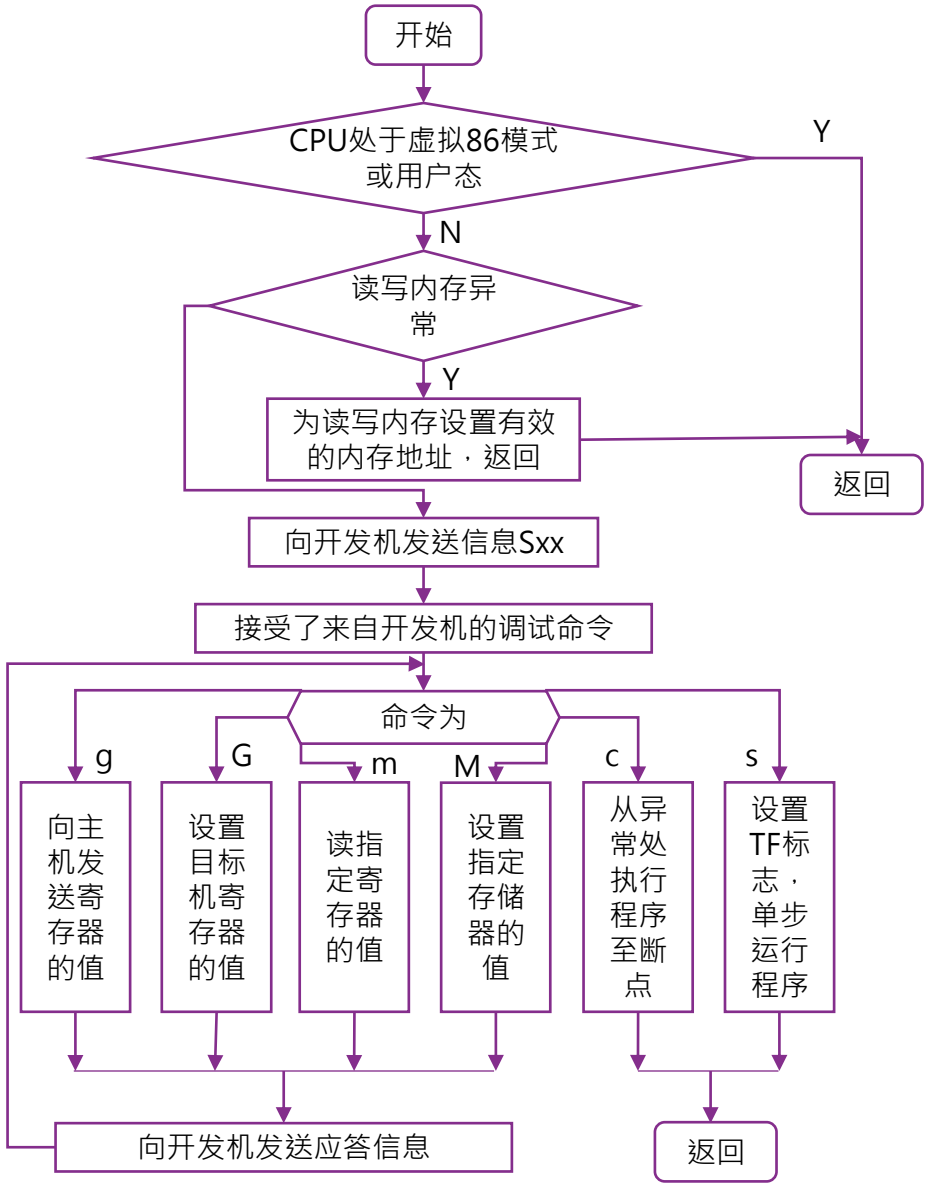
- KGDB主要有二大模块组成：
  - 初始化模块：完成初始化过程，接管系统的所有异常、设置串口通信等低层实现；
  - 控制模块：实现通信，对GDB发送的信息包进行解析并执行，对应答包进行打包发送。

KGDB初始化模块实现

- 接管系统异常处理
  - KGDB对各个需要捕获的异常处理函数进行修改。当发生异常时，使异常处理事件进入统一的异常处理函数handle_excepton。
  - void set_debug_traps(void)向系统注册调试过程中的处理函数：handle_exception（）。
- 进行串口初始化
  - struct serial_state* gdb_serial_setup(int ttyS, int baud) 参数ttyS为串口号，baud为传输波特率。函数返回该串口的状态。

KGDB控制模块实现（1/2）

- 异常处理函数：
  - handle_exception



KGDB控制模块实现（2/2）

- 相关通讯处理函数：
  - static int read_char(void) 从串口读取一个字节的数据。
  - static void write_char(int chr) 向串口写一个字节的数据。
  - static void getpacket(char *buffer) 根据串口通讯协议RSP，获取一个命令帧数据，并进行分析。
  - static void putpacket(char *buffer) 根据串口通讯协议RSP，组装一个命令帧数据，并发送。
  - 注：上述串口通讯函数只能以轮询方式收/发数据。

内核进入调试状态的路径

- 内核进入调试状态有两种方法：
  - 方法一：通过在内核启动的时候向内核传入参数，可以调试系统启动过程内核的运行状况；参数（Gdb、gdbttyS、gdbbaud）
  - 方法二：在内核完全导入系统正常运行的情况下，通过使用一个gdbstart工具将驱动串口设备，内核的控制权交给本地主机。

网络调试 - tcpdump

- 如果嵌入式平台之间需要进行网络通信，那么可能就需要使用嵌入式平台上的网络调试和诊断工具了。
- 在Linux和众多类UNIX操作系统中，最为著名的网络调试和诊断工具非tcpdump莫属。

tcpdump 原理

- tcpdump是一款功能强大、截取灵活的开源嗅探器工具，它广泛应用于很多类UNIX系统上。
  - 嗅探器工具实际上是网络上的一个抓包工具，同时可以对抓到的包进行分析，这在网络调试过程中非常有用。
  - 在共享式的网络中，数据包会以广播的形式发送给网络中所有主机，但是默认情况下，主机的网卡会自行判断该数据包是否该接收，这样就会抛弃不需要接收的数据包。而使用了嗅探器工具之后，它会拦截所有经过主机网卡的数据包，从而达到监听的效果。

tcpdump 原理

- tcpdump，即dump traffic on network，它可以根据使用者的定义有选择性地对网络上的数据包进行拦截，它支持针对网络层、协议、主机、网络或端口的过滤，并且提供and、or和not等逻辑关系运算符来加强过滤功能。
- tcpdump的精髓在于它的高效的过滤表达式。
  - 如果不给出过滤表达式的话，则所有经过主机网卡的数据包都会被输出。
  - 如果明确给出了过滤表达式，则匹配此表达式的数据包信息才会被输出。

tcpdump

- tcpdump 命令的语法如下：

```
› tcpdump -h
tcpdump version tcpdump version 4.9.3 -- Apple version 90.100.1
libpcap version 1.9.1
LibreSSL 2.8.3
Usage: tcpdump [-aAbdDefhHIJKlLnNOpqStuUvxX#] [ -B size ] [ -c count ]
                [ -C file_size ] [ -E algo:secret ] [ -F file ] [ -G seconds ]
                [ -i interface ] [ -j tstamptype ] [ -M secret ] [ --number ]
                [ -Q in|out|inout ]
                [ -r file ] [ -s snaplen ] [ --time-stamp-precision precision ]
                [ --immediate-mode ] [ -T type ] [ --version ] [ -V file ]
                [ -w file ] [ -W filecount ] [ -y datalinktype ] [ -z postrotate-command ]
                [ -g ] [ -k ] [ -o ] [ -P ] [ -Q meta-data-expression]
                [ --apple-tzo offset] [--apple-truncate]
                [ -Z user ] [ expression ]
```

- tcpdump有众多的命令行选项，由于篇幅有限，在此不一一列举，具体可以参考tcpdump的man文件


Thank you!