3. (1) 在标准输出设备上显示字符串“Hello, world.”；

(2) 第 16 行和第 20 行的 int $0x80 指令；

(3) 该用户程序第 16 行调用了 4 号系统调用 write，第 20 行调用了 1 号系统调用 exit。

4. (1) 用“objdump -d”反汇编，由反汇编代码可得 main 栈帧中存放情况，其中 14（0xe）存放在 R[esp]+8 处，0x8048510 存放在 R[esp]+4 处，1（0x1）存放在 R[esp]处；

(2) 首先通过 call 指令进入 write，其中通过 mov 指令将 main 栈帧中字符串长度、字符指针和文件描述符分别装入 EDX、ECX、EBX，调用号 4 装入 EAX，然后执行 int $0x80 从用户态陷入内核态，调用系统调用处理程序执行，其中根据系统调用号 4 跳转到 sys_write，完成将字符串写入文件的功能，执行结束后依次返回系统调用处理程序和主程序:

用户空间、运行在用户态: 内核空间、运行在内核态:

main --> write --> int $0x80 --> system_call --> sys_write

(3) 第 3 题程序设计的便捷性和灵活性不如本题，前者采用汇编程序设计，若参数不同则需重新编写不同的指令，本题采用高级语言程序设计，改变 write 的实参就可以实现不同的功能。

然而执行时间第 3 题更短，因为汇编语言比高级语言中调用函数省时。

5. (1) 因为 hello.c 中使用了 C 标准库函数 printf。

因为在 stdio.h 中有 printf 的声明，且 printf 是 C 标准库函数，编译器预处理时能得到 printf 的原型声明和相关信息，链接器链接时能根据 C 标准库完成 printf 模块的链接；

(2) 需要经过预处理、编译、汇编和链接生成可执行文件 hello，然后启动 hello 程序执行。其中预处理阶段对带#语句进行处理，编译将预处理后的文件编译得到汇编语言程序，汇编将汇编语言程序转化为可重定位的机器语言目标代码文件，链接将多个可重定位的机器语言目标代码文件及库函数链接起来，得到最终的可执行文件；

(3) 因为 printf 默认的输出设备就是标准输出设备 stdout，所以无需额外指定字符串的输出目的地，运行可执行文件 hello 即可在屏幕上显示设定的字符串；

(4) 机器码可以根据对应的 ascii 码得到，即 48H 65H 6CH 6CH 6FH 2CH 77H 6FH 72H 6CH 64H 0AH 00H。该序列为只读数据，存放在 hello.o 文件的.rodata 节，hello 的只读代码段；

(5) printf.o 模块在静态库 libc.a 中。静态链接后，printf.o 的代码部分（.text 节）被映射到虚拟地址空间的只读代码段；若采用动态链接，printf 的代码在虚拟地址空间的共享库映射区；

(6) //EBX 入栈

//字符串长度装入 EDX

//字符串首地址装入 ECX

//文件描述符装入 EBX

//调用号 4 装入 EAX

//系统调用指令

//EBX 的旧值出栈

//比较系统调用返回值和-4095

//若大于等于转到系统调用出错处理

//返回到调用 write 的下一行指令执行

该 Linux 系统中系统调用返回的最大错误号是 4095（范围在 1~4095）；

(7) 便捷性和可移植性本题 > 第 4 题 > 第 3 题，第 3 题需要针对不同参数编写指令，第 4 题直接调用 write 函数，只能在支持该函数系统调用的平台上运行，本题调用是 C 标准库函数，可以在不同平台上运行。执行时间最短的是第 3 题，因为汇编语言比高级语言调用函数更省时。

6. (1) 在内核的设备驱动程序层；
(2) 在内核的与设备无关软件层；
(3) 在用户 I/O 软件层；
(4) 在内核的设备驱动程序层与中断服务程序层；
(5) 在内核的设备驱动程序层与中断服务程序层。

8. 要达到最快打印速度，则打印机数据传输率为 6x50x80/60=400 字符/s，若采用中断控制 I/O 方式来进行字符打印输出，最长没 1/400=2.5ms 需处理一次中断申请，实际中断响应和处理时间为每字符 1000x1/500MHzx1000=0.02ms，远小于 2.5ms，故可以采用中断控制 I/O 方式。