2.

（1）链接；源程序—预处理—编译—汇编—链接—完全可执行的目标文件。

（2）模块化，使一个程序被划分成多个模块，由不同的程序员进行编写，并且可以构建公共的库函数以提供给不同的程序进行重用。效率高，每个模块可以分开编译。

（3）可执行目标文件由链接器将若千个相互关联的可重定位目标文件组合起来而生成。可重定位文件中的代码和数据的地址是相对于起始地址0而得到的，可执行文件中代码和数据的地址则按照操作系统规定的存储器映像来确定起始地址，并且可重定位文件中代码和数据的地址将会被修改，使得它们被重定位到运行时的虚拟存储空间中相应地址处。可执行目标文件格式与可重定位目标文件格式类似。

（4）符号解析和重定位。

（5）.text：目标代码部分

.rodata：只读数据

.data：已初始化的全局变量

.bss：未初始化的全局变量

（6）同上

（7）只读代码段和可读写数据段；.init、.fini、.text、.rodata合成只读代码段；.data、.bss合成可读写数据段。

（8）可执行目标文件的程序头表中记录了可执行目标文件中所有存储段的相关信息，如存储段类型，段在虚拟地址空间中的起始地址、长度、对齐方式、访问权限等。这些信息反映了可执行目标文件在运行时的存储器映像，即可执行目标文件中的代码段和数据段在虚拟地址空间中的映射关系。加载可执行目标文件时，加载器可根据可执行目标文件中的程序头表对可装人段进行映射。

（9）每个可执行目标文件都采用布局相对一致的存储器映像方式，即映射到一个统一的虚拟地址空间上，使得链接器在重定位时可以完全按照一个统一的虚拟存储空间来确定每个符号的地址，而不用管其数据和代码将来存放在主存或磁盘的何处。因此，引入虚拟存储管

理简化了链接器的设计和实现。同样，引入虚拟存储管理也简化了程序加载过程。因为统一的虚拟地址空间映像使得每个可执行目标文件的只读代码段都映射到同一个地址(例如Linux系统是0x08048000 )开始的一块连续区域，可读写数据段映射到虚拟地址空间中的一块连续区域，因而加载器可以非常容易地对这些连续区域进行分页，并初始化相应页表项的内容。加载时，只读代码段和可读写数据段对应的页表项都被初始化为“未缓存页”(即有效位为0)，并指向磁盘中可执行

目标文件中适当的地方。因此，程序加载过程中，实际上并没有真正从磁盘上加载代码和数据到主存，而是仅仅创建了只读代码段和可读写数据段对应的页表项。只有在执行代码过程中发生了“缺页"异常，才会真正从磁盘加载代码和数据到主存。

（10）对于并发运行的十几个进程来说，静态链接会造成极大的主机资源浪费；动态链接的共享库的方式，能够节省资源。

3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 符号 | 是否在tese.o的符号表中 | 定义模块 | 符号类型 | 节 |
| a | 在 | main.o | extern | .data |
| val | 在 | test.o | global | .data |
| sum | 在 | test.o | global | .text |
| i | 不在 | —— | —— | —— |

4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 符号 | 是否在swap.o的符号表中 | 定义模块 | 符号类型 | 节 |
| buf | 在 | main.o | extern | .data |
| bufp0 | 在 | swap.o | global | .data |
| bufp1 | 在 | swap.o | local | .bss |
| incr | 在 | swap.o | local | .text |
| count | 在 | swap.o | local | .data |
| swap | 在 | swap.o | global | .text |
| temp | 不在 | —— | —— | —— |

5.

（1）main.c中强符号有x、z、main，弱符号有proc1；proc1.c中强符号有proc1，弱符号有x

（2）x=0，z=0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| &z | 02 | 00 | …… | …… |
| &x | 01 | 01 | 00 | 00 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| &z | 00 | 00 | F8 | BF |
| &x | 00 | 00 | 00 | 00 |

x=0，z=-16392

（3）static double x;

void proc1()

{

x=-1.5;

}