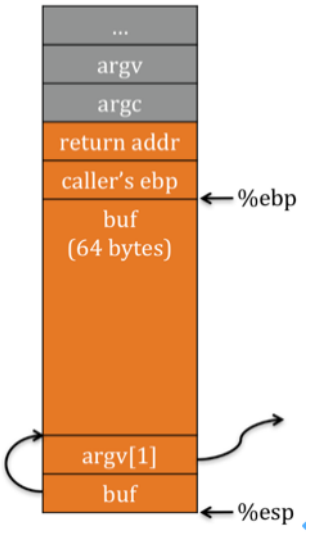
二、

（1）栈溢出攻击是通过缓冲区溢出覆盖栈上的返回地址，从而使程序的返回到攻击者构造的恶意地址处执行。

（2）



要点：在调用 strcpy 函数之前，main 函数负责构造 strcpy 的参数，argv[1]和 buf。参数 buf 指向栈上分配的数组，其大小为 64 字节。参数 argv[1]是用户输入数据，攻击者通过控制 argv[1]的长度可以实现覆盖 main 函数栈中的返回地址。具体来讲，当 argv[1]的长度为 72（64+4+4），最后 4 字节可覆盖 return addr，在 main 退出时达到控制程序执行流程的目的。

（3）三种防御方法分别是：Canaries（或 stack cookies）、DEP（Data Execution Prevention）、ASLR（Address Space Layout Randomization）。Canary 机制是在构成生成栈的时候，在其上放入特殊的数据字段，在函数返回时判断这些数据是否被更改了，如果被修改了则说明有潜在的攻击；DEP 是防止数据页中的代码被执行，一般来讲代码在运行时是不会改变的（只读），那么在栈中被注入的恶意代码(写操作)，可以通过 CPU 对内存页设置 W^X 来使得写和执行不能同时存在；ASLR 是对进程的地址空间（虚拟地址空间）进行随机化处理，使得返回的地址（包括代码段、堆和栈上的地址）不是固定的，程序每次执行生成的地址是随机的，使得攻击者猜测特定地址的难度增大，有效防止覆盖特定的地址。

三、

地址空间是操作系统提供的管理内存的一种抽象，进程所看到的只是虚拟的地址空间，好像每个进程都拥有一个独立的地址空间。它透明于运行的进程，地址转换高效、提供保护以防止进程间相互影响。

图（a）中的 ipr 指针指向的地址只分配了 4 个（genIPR 函数）和 5 个（genIPW 函数）整型数，genIPR 函数中*(ipr -1000)和*(ipr+1000)两个读操作超过了合法的范围，同样，genIPW 函数中*(ipr -1000)和*(ipr+1000)两个写操作也超过了合法的范围。图（b）中 plk++操作使得指针指向的地址值增加了，在释放通过 malloc 分配的地址时会出现错误。图（c）genPNH 函数中 free 不能释放栈上的空间，genFUM1 函数中 free 的参数 fum+1 指向的地址存在错误，释放空间时堆管理器无法准确知道将要释放的空间的大小，genFUM2 函数中第二次 free 操作会出现错误，因为重复释放同一个指针指向的地址空间。

四、

（1）

（2）对于 FIFO 算法，在缓存大小为 4 时 miss 率反而比缓存大小为 3 时更大，这违背了缓存大命中率应该越高的直观原则，这种现象被称为 Belady 异常。LRU 算法不存在这个问题，因为它符合一种称为 stack property 的属性，这种属性的一个特点就是大小为 N+1 的缓存一定包含大小为 N 的缓存的内容。

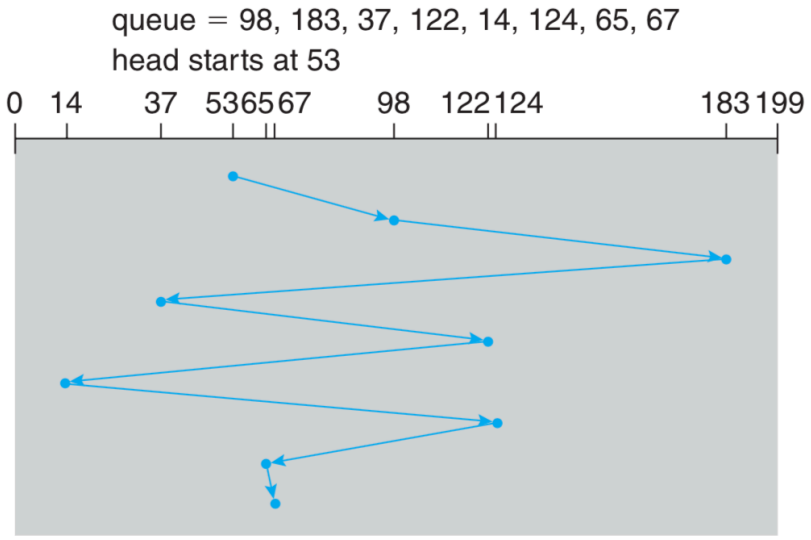
（3）LRU 算法是最近最少使用算法，在真实系统实现时，要跟踪每个页面的使用时间和频率，需要一个时间字段和一个频率字段来存储信息，需要的额外存储空间过多；另外，在进行判断时需要对所有页面的时间和频率字段进行遍历查找，以找到最近最少使用的页面，这个遍历过程时间开销很大，因此一般采用近似的 CLOCK 算法；CLOCK 算法的一般实现：首先，假设所有内存页以循环链表的方式组织，开始的时候，一个时钟指针（clock hand）指向某个特定的页；然后，当发生页面置换时，操作系统检查当前指向的页 P 的使用位（use bit）是 1 还是 0，如果是 1，表示当前指向的页最近被访问过，不应该被置换出去，将 clock hand 指向下一个页 P+1，并将 P 的 use bit 置 0；算法依次搜索循环链表，直到找到 use bit 为 0 的页，将其作为被置换的页；最坏情况下，需要搜索所有的内存页。

五、

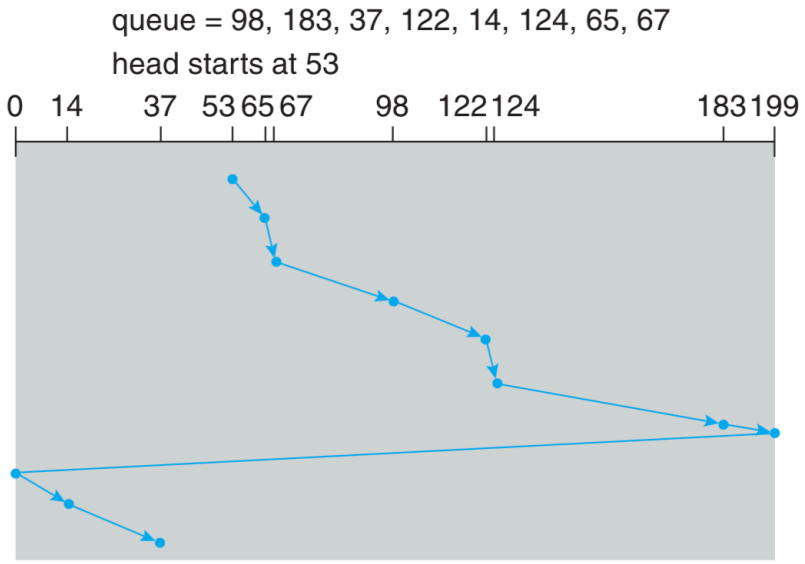
（1）FCFS 是先到先服务调度算法，根据请求到达的先后顺序进行服务，先到达的先服务；SSTF 是最短寻道时间调度算法，优先调度请求队列中寻道时间短的请求；SCAN 算法是磁头从在最外侧与最内侧磁道直接来回扫描，在扫描过程中服务请求队列中的请求，C-SCAN 是当磁头达到一侧磁道时，立即移动至另一侧磁道，移动过程中不服务请求。

（2）图中所示的是 SSFT 调度算法。

FCFS 调度算法如图：



C-SCAN 调度算法如图：



六、

RAID-1 通过在两个磁盘镜像数据实现数据冗余，从而能够容忍一个磁盘失效。在顺序读的情况下，在磁盘达到稳定传输状态时，磁盘数据访问模式如下图所示，**此时从单个磁盘来看，单个磁盘上的数据不是连续访问的**，所以顺序读的吞吐率为 N*S/2。

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

RAID-4 的数据冗余是通过将校验码存放在一个单独的校验磁盘上，如下图所示。

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

在随机写的情况下，RAID-4 **需要对校验磁盘进行一读一写，也就是两次磁盘访问，这种情况下，随机写性能受限于单个校验盘的性能**，所以在稳定状态下吞吐率是 R/2。

RAID-5 针对 RAID-4 将校验码存放在单个磁盘的问题，将校验码分布循环存放在多个磁盘中，如下图所示。

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

在随机写的情况下，**每个写操作都会导致对 RAID-5 的两个读两个写，也就是四次磁盘访问**，在稳定状态下，整个系统中的每个磁盘为满足随机写，都需要四次磁盘访问，因此吞吐率为 N*R/4。

七、

（1）inode 是 unix 类操作系统中文件系统保存目录和普通文件元信息的数据结构；inode 中包含的信息有文件大小、用户 id、最后访问时间、创建时间、最后修改时间、组 id、块数量、访问模式（读、写或执行）等，如下图表示 ext2 文件系统的 inode。

Size	Name	What is this inode field for?
2	mode	can this file be read/written/executed?
2	uid	who owns this file?
4	size	how many bytes are in this file?
4	time	what time was this file last accessed?
4	ctime	what time was this file created?
4	mtime	what time was this file last modified?
4	dtime	what time was this inode deleted?
2	gid	which group does this file belong to?
2	links_count	how many hard links are there to this file?
4	blocks	how many blocks have been allocated to this file?
4	flags	how should ext2 use this inode?
4	osd1	an OS-dependent field
60	block	a set of disk pointers (15 total)
4	generation	file version (used by NFS)
4	file_acl	a new permissions model beyond mode bits
4	dir_acl	called access control lists
4	faddr	an unsupported field
12	i_osd2	another OS-dependent field

（2）根据假设条件可知，4KB 的磁盘数据块，每个指针 4 字节，那么每个磁盘数据块可存放 4KB/4=1024 个指针。直接指针指向的就是实际存储数据的数据块，间接指针指向的数据块存放的仍然是指针（一级或多级）。那么，10 个直接指针可寻址的文件大小为 10*4KB=40KB，1 个一级间接指针指向的数据块有 1024 个指针，每个指向一个 4KB 的数据块，总和是 1024*4KB=4096KB，所以 10 个直接指针和 1 个间接指针可寻址的文件大小为 40KB+4096KB=4136KB，约等于 4.04MB。

根据上面分析可知，一个一级间接指针可指向 1024 个磁盘块，那么 1 个二级间接指针可以指向的磁盘块总数有 1024*1024 个，也就是有 1024*1024 个直接指针，那么可寻址的文件大小为 1024*1024*4KB=4GB。

八、

Ken Thompson 是 UNIX 操作系统和 C 语言的发明者之一。（2 分）
比如 Dijkstra 发明的 P, V 原语等，唐纳德等。（2 分）
能够结合云计算、多核计算、异构计算、嵌入式计算任何一个点分析操作系统发展趋势。（6 分）