static linking - linker copying all library routines used in the program into the .exe image - faster/portable but requires more memory/disk space

Dynamic linking - places name of a sharable library in .exe img - multiple programs can share a single copy of the library

Relocation - process of assigning load addresses to various parts of a program

strong/weak symbols - functions/initilized global variables = strong; unintialized global vars get weak

virtual memory - technique that virtualizes various forms of computer storage (RAM/disk storage) allowing a program to be desgined as though there is only one kind of memory, behaves like normal ram

sandbox – security mechanism, seperates running programs. Used to execute untested or untrusted code from unverified suppliers, users, websites etc..

2. Compare distributing a statically-linked application to distributing a VM image (such as one for VMWare) for a commercial software firm.

3. How does operator/method overloading complicate a linker's job? How do they work with compilers to resolve these issues?

4. How does a DLL/.so file differ from a typical .o file? A '.o' is an object file where as a DLL/.so are static libraries

5. Why would the ability to link a DLL into a running application be useful? Give two examples.

DLL can be shared between multiple processes, if a bug is found in the DLL you can just rebuild the DLL rather then the whole program.

6. Why would "library interpositioning" be useful for both security professionals and hackers?

Will allow a security professional to preload a specific set of security functions when they open a program. Likewise a hacker could just as easily preload malicious code

7. Would static or dynamic inking tend to offer better locality on a typically PC with virtual memory enabled?

Dynamic linking – reduces overall space required for a program thus offering more locality