DYNAMIC LINKING

Dynamic linking, often implemented with dynamically linked libraries (DLL), is a common way to partition applications and subsystems into smaller portions, whichcan be compiled, tested, reused, managed, deployed, and installed separately.

- **Several applications can use the library** in such a fashion that only one copy ofthe library is needed, thus saving memory.
- Application-specific tailoring can be handled in a convenient fashion, provided thatsupporting facilities exist.
- **Smaller compilations and deliveries are enabled**
- **Composition of systems becomes more flexible, because only a subsetof all possible software can be included in a device when creating a device for a certain segment**.
- **It is easier to focus testing** to some interface and features that can beaccessed using that interface.
- **Library structure eases scoping of system components and enables thecreation** of an explicit unit for management.
- Work allocation can be done in terms of dynamic libraries, if the implementation iscarried out using a technique that does not support convenient mechanisms for modularity.

STATIC VERSUS DYNAMIC DLLS

- While dynamically linked libraries are all dynamic in their nature, there are two differentimplementation schemes.
- **One is static linking, which most commonly means that the library is instantiated at the starting time of a program**, and the loaded library resides in the memory as long as the program that loaded the library into its memoryspace is being executed.
- In contrast to static DLLs, **dynamic DLLs, which are often also referred to asplug-in, especially if they introduce some special extension, can be loaded and unloaded whenever needed,** and the facilities can thus be alteredduring an execution.
- The benefit of the approach is that one can introduce new features using such libraries.For instance, in the mobile setting one can consider that sending a message is an operation that is similar to different message types (e.g. SMS, MMS, email), but differentimplementations are needed for communicating with the network in the correct fashion.

CHALLENGES WITH USING DLLS

1. A common risk associated with dynamic libraries is the **fragmentation of thetotal system** into small entities that refer to each other seemingly uncontrollably

2. Another problem is that if a dynamic library is used by only one application, **memory consumption increases due to the infrastructure** needed for management of the library.

IMPLEMENTATION TECHNIQUES

- Fundamentally, **dynamically linked libraries can be considered as components that offer a well-defined interface for other pieces of**

software to access it.

- Additional information may be provided to ease their use. This is not a necessity,however, but they can be **self-contained as well,** in which case the parts of the program that use libraries must find the corresponding information from libraries.
- Usually, this is implemented with standard facilities and an application programmerhas few opportunities to optimize the system.

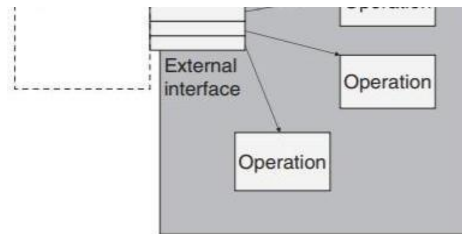

Figure 4.1 Dynamically linked library

Dynamically linked libraries can be implemented in two different fashions.

1. Offset based linking
2. Signature based linking

OFFSET BASED LINKING

- Linking based on offsets is probably **the most common way to loaddynamic libraries.**
- The core of the approach is to add a table of function pointers to the library file, which identifies where the different methods or procedures exported from the dynamically linked library are located, thus resembling the virtual function table used ininheritance.

SIGNATURE BASED LINKING

- In contrast to offset-based linking of dynamically linked libraries**, also language-level constructs, such as class names and method signatures, can be used as the basis for linking.**
- Then, the linking is based on loading the whole library to the memory and then performing the linking against the actual signatures of the functions, which must then be present in one form or another.

-