The Kernel Symbol Table in Linux is a data structure that maps symbol names (like function or variable names) to their corresponding memory addresses within the kernel. It's crucial for debugging, dynamic module loading, and symbol versioning. You can find a snapshot of the kernel symbol table in `/proc/kallsyms` (or `/proc/ksyms` in older kernels).

Here's a more detailed breakdown:

Purpose:

- **Debugging:**

  During debugging, the symbol table helps map symbol names to memory locations, aiding in identifying the source of issues in the kernel.

- **Dynamic Module Loading:**

  When modules are loaded, the symbol table resolves dependencies between the module and the existing kernel, allowing modules to use kernel functions and data.

- **Symbol Versioning:**
  The table also helps track changes to symbols over time, maintaining compatibility with older modules.

Key Aspects:

- **Visibility:**

  Symbols within the kernel can have different levels of visibility, such as static (only visible within a compilation unit), external (visible to other parts of the kernel), and exported (visible to loadable modules).

- `/proc/kallsyms`:

  This file provides a text-based representation of the kernel symbol table, allowing users to see which symbols are available and their addresses.

- **Module Stacking:**

  The symbol table enables modularity by allowing modules to use symbols exported by other modules. This layered approach simplifies development and maintenance.

- `EXPORT_SYMBOL`:

  Kernel developers use macros like `EXPORT_SYMBOL` and `EXPORT_SYMBOL_GPL` to make functions and variables accessible to modules.

- `System.map`:
  In addition to `/proc/kallsyms`, the `System.map` file, found in `/boot/System.map-<kernel_version>`, also contains a symbol table, especially useful for debugging kernel crashes.