PG - DESD

Module – Embedded C Programming

Trainer - Devendra Dhande
Email – devendra.dhande@sunbeaminfo.com
Mobile No - 9890662093

Function Calling Conventions

- How functions are called on particular CPU architecture?
  - How arguments are pushed on the stack? (left to right or right to left).
  - Who pop arguments from the stack? (calling function or called function)
- How assembly code is generated by the compiler to call a function?
- Calling conventions depends on CPU architecture.
  - **x86 architecture**
    - pascal
    - cdecl
    - stdcall
  - **ARM architecture**
    - AAPCS or ATPCS

- **Pascal Calling Convention**
  - Outdated. Was supported in Turbo C compiler.
  - arg push order: left to right
  - stack cleanup: called function
- **cdecl calling convention**
  - C Declarator. Default calling convention for C programs.
  - arg push order: right to left
  - stack cleanup: calling function
- **stdcall calling convention**
  - Standard Call. Used in some technologies like COM.
  - arg push order: right to left
  - stack cleanup: called function
  - Generated assembly code is more compact (when same function is called multiple times).

Static and Dynamic Linking of Libraries

- process of collecting and combining multiple object files to create a final executable.
- Linking can be performed at
  - **Compile time** – when machine code is generated from source code **(Static Linking)**
  - **Runtime time** – when program is loaded into memory **(Dynamic Linking)**

- **Static linking**
  - all library modules used in the program are copied into final executable file.
  - Performed by linker and is last step of compilation.
  - Executable size is larger comparatively.

- **Dynamic linking**
  - Linking of all library modules is performed on the fly as program starts running on the system.
  - Name of the shared library is placed in the final executable file.
  - Actual linking takes place at run time (both executable file and library are loaded in the memory).
  - Executable size is reduced.

Steps to create static libraries

1. Create header files and source files for your library. (create mymath.h, add.c, sub.c)
2. Compile all source files.
   - gcc -c sub.c
   - gcc -c add.c
3. Create a static library by combining all object files.
   - ar rs mymath.a add.o sub.o
4. Write a program which will use functions of your library. (create demo.c)
5. Compile your program
   - gcc -I . -c demo.c
6. Link your program with static library to get final executable file.
   - gcc -o demo demo.o libmymath.a
   - gcc –o demo -L . demo.o –lmymath
7. Run your program

Steps to create shared libraries

shared library (on Linux) or a dynamic link library (dll on Windows)

1. Create header files and source files for your library. (create mymath.h, add.c, sub.c)
2. Compile all source files.
   - gcc -fPIC -c sub.c
   - gcc -fPIC -c add.c
3. Create a shared library by combining all object files.
   - gcc -shared -o libmymath.so add.o sub.o
4. Install shared library
   - Add your library in standard directory and run command ldconfig
5. Write a program which will use functions of your library. (create demo.c)
6. Compile your program
   - gcc -c demo.c
7. Link your program with static library to get final executable file.
   - gcc -o demo demo.o libmymath.so
   - gcc -o demo demo.o –lmymath
8. Run your program

Thank you!

Devendra Dhande <devendra.dhande@sunbeaminfo.com>