CSEN 383 : PROJECT - 1

NAME : MANASA MADIRAJU
SCU ID : 07700009942

<u>Working on MacOS</u>

1. Be able to have reliable Linux environment

Running on M2 MacBook

Command : sw_vers

```
manasa@Manasas-MBP Downloads % sw_vers
ProductName:            macOS
ProductVersion:         14.4.1
BuildVersion:           23E224
manasa@Manasas-MBP Downloads %
```

2. Compile and run the C program (forktest.c) on a Linux system and to provide a screenshot of the compilation and execution.

Commands : gcc forktest.c -o forktest
./forktest

```
manasa@Manasas-MBP Downloads % gcc forktest.c -o forktest
manasa@Manasas-MBP Downloads % ./forktest
Parent: Process started
Parent: Forking a child.
Parent: Wait for child to complete.
Child: Process started.
Child: Start 10 second idle: 10  9  8  7  6  5  4  3  2  1  0 done!
Child: Terminating.
Parent: Terminating.
manasa@Manasas-MBP Downloads %
```

3. Read the gcc man page

Command : gcc -v

```
manasa@Manasas-MBP Downloads % gcc -v
Apple clang version 15.0.0 (clang-1500.3.9.4)
Target: arm64-apple-darwin23.4.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
manasa@Manasas-MBP Downloads %
```

Apple opts for Clang over GCC for several reasons:

1. *Performance and Resource Efficiency* : Clang outperforms GCC in terms of speed and utilizes less memory.

2. *Diagnostic Capabilities* : Clang offers clearer and more concise error and warning messages compared to GCC.

3. *Language Compatibility* : Clang demonstrates superior language compliance, enhancing compatibility with newer standards and languages.

4. *Integration* : Due to its large codebase, GCC doesn't integrate well with Apple's IDE, whereas Clang does.

5. *Optimization and Reliability* : Clang surpasses GCC in optimization and reliability, largely owing to LLVM (Low Level Virtual Machine).

6. *Toolchain Compatibility* : Clang seamlessly replaces GCC in numerous scenarios without disrupting the toolchain, as it supports most commonly used GCC options. However, LLVM may still rely on GCC or another compiler front end for certain languages like Ada.

7. *Distribution Strategy* : Apple doesn't bundle a compiler with macOS or iOS and discourages the execution of user-compiled code. Nonetheless, Clang is included in Xcode's command-line developer tools.

I am running man clang because of the above explanations and gcc -v shows the clang version as provided in the above image.

Command : man clang

```
CLANG(1)                             Clang                            CLANG(1)
DESCRIPTION
NAME   clang is a C, C++, and Objective-C compiler which encompasses
       clang - the Clang C, C++, and Objective-C compilerion, assembly, and
       linking.  Depending on which high-level mode setting is passed, Clang
SYNOPSISill stop before doing a full link.  While Clang is highly integrated,
       clang [options] filename ...d the stages of compilation, to understand
       how to invoke it.  These stages are:
DESCRIPTION
       clang is a C, C++, and Objective-C compiler which encompassescontrols
       preprocessing, parsing, optimization, code generation, assembly, and
       linking.  Depending on which high-level mode setting is passed, Clang
       will stop before doing a full link.  While Clang is highly integrated,
       it is important to understand the stages of compilation, to understand
       how to invoke it.  These stages are:
       Preprocessing
       Driver The clang executable is actually a small driver which controls
              the overall execution of other tools such as the compiler,
              assembler and linker.  Typically you do not need to interact
              with the driver, but you transparently use it to run the other
              tools.

       Preprocessing
              This stage handles tokenization of the input source file, macro
              expansion, #include expansion and handling of other preprocessor
              directives.  The output of this stage is typically called a ".i"
              (for C), ".ii" (for C++), ".mi" (for Objective-C), or ".mii"
              (for Objective-C++) file.

       Parsing and Semantic Analysis
              This stage parses the input file, translating preprocessor
              tokens into a parse tree.  Once in the form of a parse tree, it
              applies semantic analysis to compute types for expressions as
              well and determine whether the code is well formed. This stage
              is responsible for generating most of the compiler warnings as
              well as parse errors. The output of this stage is an "Abstract
              Syntax Tree" (AST).

       Code Generation and Optimization
              This stage translates an AST into low-level intermediate code
              (known as "LLVM IR") and ultimately to machine code.  This phase
              is responsible for optimizing the generated code and handling
              target-specific code generation.  The output of this stage is
              typically called a ".s" file or "assembly" file.

              Clang also supports the use of an integrated assembler, in which
              the code generator produces object files directly. This avoids
              the overhead of generating the ".s" file and of calling the
              target assembler.

       Assembler
              This stage runs the target assembler to translate the output of
              the compiler into a target object file. The output of this stage
              is typically called a ".o" file or "object" file.

       Linker This stage runs the target linker to merge multiple object files
              into an executable or dynamic library. The output of this stage
              is typically called an "a.out", ".dylib" or ".so" file.

       Clang Static Analyzer

       The Clang Static Analyzer is a tool that scans source code to try to
       find bugs through code analysis.  This tool uses many parts of Clang
       and is built into the same driver.  Please see
       <https://clang-analyzer.llvm.org> for more details on how to use the
       static analyzer.
:
```