Program Execution Model: C/C++ vs Python

This document explains, in a structured and academic manner, how programs written in C/C++ and Python are processed and executed by a computer system. The explanation follows standard compiler and interpreter design concepts used in Computer Science.

1. Execution of C / C++ Programs

- Source Code: The programmer writes human-readable code in a file such as demo.c or demo.cpp.
- Preprocessing: The preprocessor handles directives like #include and #define, expanding macros.
- Compilation: The compiler checks syntax and converts source code into object code (.o).
- Assembly: Object code is translated into low-level machine instructions.
- Linking: The linker combines object files with required libraries to create an executable file.
- Loading: The operating system loads the executable into main memory (RAM).
- Execution: The CPU executes machine code directly, resulting in fast performance.

2. Execution of Python Programs

- Source Code: The programmer writes Python code in a file such as hello.py.
- Compilation to Bytecode: Python internally compiles the source code into bytecode (.pyc).
- Bytecode Storage: Bytecode files are stored in the __pycache__ directory.
- Python Virtual Machine (PVM): The PVM reads bytecode and converts it to machine code at runtime.
- Interpretation: Execution occurs line by line, which is why Python is considered an interpreted language.

3. Comparative Analysis

- C/C++ programs are fully compiled before execution, while Python programs are compiled to bytecode and interpreted.
- C/C++ execution is faster because machine code runs directly on hardware.
- Python offers greater portability because bytecode runs on any system with a PVM.
- Errors in C/C++ are mostly detected at compile time, whereas Python errors often appear at runtime.

Conclusion: C/C++ follows a traditional compiled execution model focused on performance, whereas Python uses a hybrid approach combining compilation and interpretation to emphasize simplicity and portability.