A Simple Build Server for C Programs

The goal of this exercise is to make a small proof-of-concept build server and artifact repository for C programs. (See "meta-requirements" below. You do *not* have to—and you're not advised to—write the server in C!)

For the purpose of this exercise, we assume that

- a "C program" is a program written in ISO C99 as a self-contained collection of `.c` and `.h` files, along with a `Makefile` with a canonical `all` and `clean` target. The C program will only depend on the C standard library. We have included a sample C program called `hello`.
- a "C program" builds into a single executable without requiring dynamic linkage to anything but the system's `libc`.
- a "C program" is tracked in a Git repository.

You are welcome to make assumptions and modifications to the above as you see fit (e.g., you may decide or change how and where artifacts are written, how they're named, etc.)

Goal #1: Registering & building C programs

Be able to register a C program with a server and build it. How one might register the projects is up to you (e.g., static configuration, HTTP API, interactive command loop, GUI). You should be able to detect build failures. How you report failures is left to your discretion. A failed build is considered to be a permanent failure, only fixable by additional commits.

Your server shall be able to support the registration and building of multiple independent C programs.

Goal #2: Simple artifact repository

Build a simple artifact repository. The artifact repository may be of any form at your discretion (e.g., a folder structure on the file system, a SQL database, etc.), so long as your build server understands it.

We may assume that

- artifacts are simple binaries tied to a specific commit from the program's Git repository.
- the Git repository from which the artifacts are built is "append only" (i.e., no history rewriting, no rebasing, etc.).

Goal #3: Periodic builds

Add functionality to allow the system to automatically fetch new commits periodically and build them. At your discretion, this functionality may be global, may be per-program, or whatever you choose.

The main point of this exercise is to allow the system to run autonomously without user interaction.

Goal #4: A web page showing C programs & their statuses

Build a simple webpage that allows one to look at the current and previous builds for the C projects known to the system, as well as the status of current builds (completed, failed, building, etc.). The web page does *not* need to be dynamically updated (i.e., one can rely on browser refreshes), and does not *need* to be interactive (i.e., commands to kick off builds are not necessary).

Meta-requirements

Terminology: Above, we have used programming terms like "function," "type," "constructor," etc. These terms are being used broadly, and may be interpreted however you please in your programming language. For example, something specified above to be a "function" may be implemented in your programming language as a method, procedure, subroutine, or otherwise.

Naming Conventions: You do *not* need to match the function and variable names exactly. Please feel free to choose whichever names you prefer and whichever names would be idiomatic in your programming language.

Language: You code can be written in any language of your choice. We prefer—but do not require—C++, Python, or Common Lisp. If it's not one of those, we'd like instructions on how to run your code as simply as possible.

Dependencies: Feel free to use any easily-accessible libraries that are utilitarian in nature. Do not use library functionality which directly solves any of the challenges. Avoid using third-party services and trial accounts.

Operating System: Your code should work on a UNIX system, like Linux or macOS. If this is not feasible for you, please let us know ahead of time.

Quality: We know that this is a take-home exercise, and time constrains one's ability to make such a project fully robust. We ask that you code to your usual standard, but if shortcuts shall be made, to clearly mark them.

Version Control: Do as you would in an ordinary, professional project. You may, if convenient, share a private GitHub repository with us.

Testing: Do as you see fit, noting the flexibility of the "quality" requirement above.

Documentation: Documentation should be done as you see fit, noting the flexibility of the "quality" requirement above.

Extensions and Scope: In some instances, we only ask for limited problem scope to be addressed. You are welcome—but not required—to go beyond the

scope of the question, by, for example, considering broader sets of inputs or more specific outputes.

The code you write is owned completely by you, and of course may be repurposed or used for anything you please. However, **we humbly request that the code you write as it pertains to this exercise remains private so others have a fair chance**.

Judgment criteria

You will be judged on overall software engineering quality, broad efficiency considerations (i.e., we are looking at big design choices, not micro-optimizations), and the extent to which your code satisfies the requirements outlined in this document.