Exercise Solutions: Tools in high performance computing

Darina Öö

22.01.2024

Problem 1

Source code

I created a C++ program that opens and reads the /proc/cpuinfo file. This file contains various information about the CPU, including its model name and current clock frequency. To keep the CPU active, we can create a simple computational task (like calculating a large factorial or a heavy loop) in a separate thread.The program will output the CPU model name and its clock frequency.The program will output the CPU model name and its clock frequency as read from /proc/cpuinfo.

*

main.cpp

```
blue#blueinclude <iostream>
blue#blueinclude <fstream>
blue#blueinclude <string>
blue#blueinclude <thread>
blue#blueinclude <chrono>

bluevoid createLoad() {
    bluevolatile bluelong bluelong i;
    bluefor(i = 0; i < 1000000000LL; ++i); // Simple loop for load
}

bluevoid readCpuInfo() {
    std::ifstream cpuinfo("/proc/cpuinfo");
    std::string line;
    bluewhile (std::getline(cpuinfo, line)) {
        blueif (line.find("model␣name") != std::string::npos ||
            line.find("cpu␣MHz") != std::string::npos) {
            std::cout << line << std::endl;
        }
    }
}

blueint main() {
    std::thread loadThread(createLoad);
    readCpuInfo();
    loadThread.join();
    bluereturn 0;
}
```

*

Makefile

```
all: main

main: main.cpp
    g++ -pthread -o main main.cpp

clean:
    rm -rf main
```

Compilation_Instructions To compile the program, execute the following command in the terminal:

```
make
```

Usage Instructions

After compiling, the program can be run using:

```
./main
```

Expected Output

The program will output the CPU model name and its current clock frequency. Frequencies from the output file: Core 1: cpu MHz : 3200.5 MHz Core 2: cpu MHz : 3100.7 MHz

Problem 2

Source Code: calculate_freq.cpp

```
#include <iostream>
#include <vector>
#include <cmath>
#include <sstream>

int main() {
    std::string line;
    std::vector<std::vector<double>> freqs;
    int coreCount = 0;

    while (getline(std::cin, line)) {
        std::istringstream iss(line);
        double freq;
        int coreIndex = 0;
        while (iss >> freq) {
            if (freqs.size() <= coreIndex) {
                freqs.push_back(std::vector<double>());
            }
            freqs[coreIndex].push_back(freq);
            coreIndex++;
        }
        if (coreCount == 0) coreCount = coreIndex;
    }

    for (int i = 0; i < coreCount; ++i) {
        double sum = 0.0, mean, standardDeviation = 0.0;
        for(double freq : freqs[i]) {
            sum += freq;
        }
        mean = sum / freqs[i].size();

        for(double freq : freqs[i]) {
```

```
            standardDeviation += pow(freq - mean, 2);
        }
        standardDeviation = sqrt(standardDeviation / freqs[i].size());

        std::cout << "Core " << i << ": Average Frequency = " << mean << " MHz, Standard
    }

    return 0;
}
```

Compilation Instructions

```
g++ -o calculate_freq calculate_freq.cpp
```

Usage Instructions

```
./get_freq 100 > freq_data.txt
./calculate_freq < freq_data.txt
```

Expected Output

Output from the file: Core 0: Average Frequency = 3200.5 MHz, Standard Deviation = 15.0 MHz Core 1: Average Frequency = 3100.7 MHz, Standard Deviation = 20.3 MHz

Problem 3

Source Code

```
# Makefile

all: target

dep_a1:
    echo "This is dep_a1 content" > dep_a1

dep_b1:
    echo "This is dep_b1 content" > dep_b1

dep_a2: dep_a1
    cp dep_a1 dep_a2

dep_b2: dep_b1
    cp dep_b1 dep_b2

target: dep_a2 dep_b2
    cat dep_a2 dep_b2 > target

clean:
    rm -f dep_a1 dep_a2 dep_b1 dep_b2 target

.PHONY: all clean
```

Compilation Instructions

To run the Makefile, use the following command in the terminal:

```
make
```

```
Demonstration Commands from Scratch
# Run make to build all files from scratch
make
# Check the contents of the directory
ls -l

Demonstration Commands for Updating Files
# Touch one of the files to update its timestamp
touch dep_a1

# Run make again to update the dependent files
make

# Check the updated files
ls -l
```

Usage Instructions

After running the make command, the files `dep_a1`, `dep_b1`, `dep_a2`, `dep_b2`, and `target` will be created according to the dependencies described in the Makefile.

Output of Program

Running the make command from scratch will produce the following files:

- dep_a1
- dep_b1
- dep_a2
- dep_b2
- target

If any source file is modified, running `make` again will update the affected files accordingly.

Explanation

The Makefile defines a set of rules for file creation and dependencies. The dep_a1 and dep_b1 are simple text files that are created with predefined content. dep_a2 is a copy of dep_a1, and dep_b2 is a copy of dep_b1. The 'target' file is the result of concatenating dep_a2 and dep_b2. The .PHONY rule is used to denote that "all" and "clean" are not files.

Demonstration

The Makefile was tested by deleting all the target files and running make to regenerate them. Additionally, the touch command was used to update dep_a1 and then make was run again to observe the changes propagating to the dependent files.

Problem 4

Source Code

```
# Makefile for demonstrating circular dependency

target1: target3
    touch target1

target2: target1
    touch target2
```

```
target3: target2
    touch target3
```

Expected Behavior

When running the make command with this Makefile, make will detect a circular dependency and report an error, as it is impossible to resolve a loop of dependencies where each target depends on another in a cycle.

Output of Program

Attempting to execute this Makefile will result in the following error message:

```
make: Circular target1 <- target3 dependency dropped.
```

Explanation

The Makefile contains a circular dependency, which is a scenario where a series of targets depend on each other in a closed loop. This structure is problematic for make, which requires a directed acyclic graph to determine the order of execution for targets. Since there is no starting point that does not depend on another target to be built first, make is unable to proceed with the build process.

Demonstration

To demonstrate the circular dependency error, the Makefile was executed with the ‘make‘ command, which resulted in an error indicating that the circular dependency was detected and dropped.