ITP 30002-{01, 02} Operating System, Spring 2021

# Homework 1. PCTest: Multi-processed Programming Assignment Testers

Shin Hong

## 1. Overview

This homework asks you to construct `pctest` which builds, and tests a given C program to assist an instructor of a C programming courses in evaluating programming homework submissions of the students. With each of given test inputs, `pctest` runs both a target program (i.e., a student's submission) and a solution program (i.e., a correct program offered by the instructor), and then compares their results to determine whether the target program returns a correct result or a failure (i.e., wrong answer) or not. In this homework, you need to find proper system library functions for different purposes of process control and use them properly in constructing `pctest`. As a report of your result, you need to produce a video demo (Section 4) to show your program working as specified in the requirements.

## 2. Program Design

This section describes the basic designs of the structure and the behaviors of a programming assignment testing program `pctest`.

### 2.1. Workflow

Figure 1 shows the overall process of `pctest`. `pctest` receives as input (1) two source code files, *target* and *solution*, and (2) a set of program inputs *Tests*. *target* is a C source code file (i.e., submitted by a student) subject to evaluation. *solution* is another C source code file assumed to be correct (i.e., offered by the instructor). We can see that *target* and *solution* aim to solve the same problem, but *target* may or may not have errors. *Tests* consists of one to ten text files each of which is an input of *target* and *solution*.

In this homework, we assume that every program given to `pctest` consists of only a single C source code file and does not have any dependencies. Another assumption is that a *target* (or *solution*) program is programmed to receive inputs only via the standard input and produce output only to the standard output.

Given source code files, `pctest` uses a compiler (e.g., `gcc`) to build two executable files respectively, and then runs these two executables with each of the test input file in *Tests*. `pctest` determines whether the execution result of *target* for an input is correct or not by checking if the program terminated successfully and the texts produced to the standard output is identical to one that produced by *solution*.

After running all given test inputs (i.e., *Tests*), `pctest` displays the summary of the test results (see Section 2.3) via the standard output. The summary must include (1) whether the compilation was succeeded or not, (2) the number of the correct test executions and the number of failed test executions, (3) the maximum and the minimum time of a single test execution, and the sum of all test execution time of *target*. Meanwhile, what *target and* solution print to the standard output must not be shown in the display.

### 2.2. Running Tests

`pctest` launches *target* and *solution* with each input file in *Tests* one by one. To better utilize a multicore CPU, for each test input, `pctest` must create one process for *target* and the other for *solution* and run them concurrently.

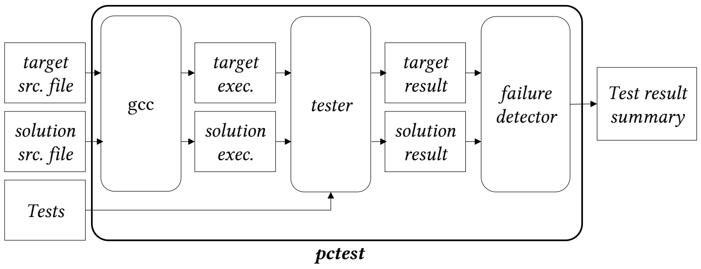

**Figure 1. overall process of `pctest`**

`pctest` must kill a process running target if its execution exceeds a predefined time limit, because it may run indefinitely. Such a test execution should be recognized as a failed one. In addition, when running *target* on a new process, `pctest` must disallow the process to open a new file to limit the danger of executing untrusted student's code on the instructor's computer (see Section 3.2).

### 2.3. Checking Test Results

`pctest` determines that *target* fails for a given test input if the corresponding test execution falls into one of the following cases:

- a crash (runtime/fatal error) occurs, or
- the program returns an error code at termination, or
- the program does not terminate within a certain amount of execution time (i.e., time over), or
- the text printed to the standard output is not identical to that of *solution.*

In addition to the functionality checking, `pctest` also measures the test execution time to maintain the following three values:

- the maximum execution time of a single test run in milliseconds, and
- the minimum execution time of a single test run in milliseconds, and
- the sum of the execution time of all test runs in milliseconds.

### 2.4. Command-line interface

The usage of `pctest` in command-line interface is as follows:

```
$ pctest -i <testdir> -t <timeout> <solution> <target>
```

A user gives <*testdir*> a path to a directory where the test input files are stored. All files under <*testdir*> will be recognized as input files. <*timeout*> specifies the time limit of a program execution in seconds. It should be an integer between 1 and 10. <*solution*> and <*target*> give a filename of the source code of the correct version and a filename of the source code of a student's program under test, respectively. `pctest` should exit if it receives an invalid command-line argument.

## 3. Implementation

This section gives the requirements, the assumption, and useful information on implementing `pctest`.

### 3.1. Constraints, assumptions, and requirements

This homework is designed to give you a chance to exercise various system functions for creating, executing, and killing processes would be used for implementing different features of `pctest`. For this reason, you are not allowed to execute a shell (or shell script) like `bash` from `pctest`, since a shell can be used as workarounds of the system libraries. Note that whether suitable system functions were selected and exercised properly in a `pctest` implementation will be evaluated.

Another requirement is that `pctest` must use `pipe` and `dup2` to construct an unnamed pipe for transferring data between a parent process with a child process[1]. Also, function `kill` must be used for a parent process to terminate a child process. For many exceptional cases, `pctest` must properly handle them so that it works reliably even with an invalid input.

Last, let us assume in this homework that a given solution program does not have any problem. Also, let us assume that all files under a given test input directory are of test inputs and there exists no subdirectory in this directory.

### 3.3. Useful information

The list of the names of files in a certain directory can be obtained by calling function `opendir`[2]. For querying the current time in milliseconds, you may use function `clock_gettime`[3]. It would be better not to use `gettimeofday`. since the use of time zone structures is now obsolete.

By calling function `setrlimit`[4], we can change the limits of various resource uses of the current process, including the maximum number of file descriptors that the process opens (i.e., this resource is referred as `RLIMIT_NOFILE`). Interestingly, `setrlimit` can decrease the limit of a resource use from the current value but cannot increase it. Regarding `setrlimit`, note that if a process is to execute `exec` (or its variants), it requires a new file descriptor to open the target binary file to load the content to the memory.

## 4. Your Tasks

You are asked to write a C program `pctest` following to the program design (Section 2) while considering the constraints on program implementation (Section 3). Your program must have a build script (e.g, `Makefile`) by which TAs can build your program without much effort. Your program must have a `README` file in addition to describe instructions for building and running `pctest`.

To present your results of this homework, you are asked to produce a video demo which must contain the following aspects:

- You must code review your program to show that you use suitable system library functions to implement different features of `pctest`.
- You must prepare realistic program inputs and then show the actual execution of `pctest`. This demonstration should cover several major scenarios of `pctest`. You need to explain the result of your program in detail to prove that your program fulfills all requirements.
- You must demonstrate how to build and run your program in a step-by-step manner. These steps should be the same as the instructions at the `README` file.

Your video must not exceed 8 minutes. Narration and subtitles must be in English. Your homework will be evaluated primary based on your video demo. Therefore, you must carefully arrange all important things that you need to deliver within a limited amount of time. Following a demo video review, TAs will try reproducing some scenarios as shown in the demo to validate whether the submitted source code is consistent with the one that presented in the video demo. Note that this reproduction step will be made at `peace.handong.edu`, thus, you would confirm that your program runs successfully on this Linux server before submission.

## 5. Submission

Turn in an archive (e.g., as a `tar` or `zip` file) of all results through Hisnet by **11:00 PM, 3rd April (Sat)**. Your submission must include (1) a link to a video demo and (2) all source code files needed for reproducing what you reported in the video. You must upload the video to YouTube or Google Drive (or other platforms) and put a link to the video at the `README` file.

[1] https://sites.google.com/handong.edu/system-programming/unit-4-inter-process-communication

[2] https://man7.org/linux/man-pages/man3/opendir.3.html

[3] https://linux.die.net/man/3/clock_gettime

[4] https://man7.org/linux/man-pages/man2/prlimit.2.html