# Assignment 2

CPSC 525 Fall 2025
Due date is posted on D2L/Gradescope. Weight: 20% of your final grade.

## 1. Written answers (50%)

Complete this part of the assignment on Gradescope. You can find a link to it on D2L.

## 2. Coding question (50%)

You will write a C++ program that tries to determine whether a user has permissions to access a file. The program will be given 3 inputs: a username, a set of permissions to check, and a path. It will then inspect each component of the path, ensuring the given user has appropriate permissions on each component.

Start by downloading and compiling the starter code with the included `Makefile`:

```
$ git clone https://gitlab.com/cpsc525f25/checkp.git
$ cd chekp
$ make
```

This starter code contains incomplete solution, and your job is to finish it. The only file you can modify (and submit for grading) is `checkp.cpp`. The starter code contains a driver main.cpp, which parses command line arguments, and calls the unfinished function `checkp()` in `checkp.cpp`, which you need to write. The driver also prints the return value from the function on standard output.

The function you need to implement is defined in `checkp.cpp` file:

```
int checkp(const std::string & username,
           const std::string & perms,
           const std::string & path);
```

The function must try to determine whether user with `username` has permission `perms` to file or directory specified in `path`.

- If the function determines that the user DOES have the requested permissions, the function returns 1.
- If the function determines that the user does NOT have the requested permissions, the function returns 0.
- If the function encounters error conditions, and is thus UNABLE to determine whether the user does or does not have the requested permissions, it returns -1 and sets the global `errno` variable to the appropriate value.

The reason the function might not be able to determine whether the given user has the requested permissions is because the user that executes the code might not have the necessary permissions on some component of the path to obtain enough information about it. For example, consider what the function needs to do when invoked with parameters:

```
checkp("pfederl","rw","/etc/passwd")
```

This is asking the function to check whether user "pfederl" has both read and write access to file "/etc/passwd". The function needs to perform many checks, such as:

- If username "pfederl" is not valid, report error
- Determine the groups the user "pfederl" is member of
- If requested permissions "rw" are not valid, report error
- If path "/etc/passwd" is not absolute, report error
- Determine all ancestor directories (components) of the path, e.g. ancestors of "/etc/passwd" are "/" and "/etc"
- For each ancestor component, starting with root, check that check that user "pfederl" has execute permission
    - Retrieve the 9 permissions bits of the component, it's owner, and it's group owner
        - If this fails, report error
    - If the owner is pfederl:
        - If user permissions do not have execute bit set, return 0 immediately
    - Otherwise, if the group owner is one of "pfederl's" groups:
        - If group permissions do not have execute bit set, return 0 immediately
    - Otherwise:
        - If other permissions do not have execute bit set, return 0 immediately
- Final step is to check that user "pfederl" has both read and write permissions on the full path:
    - Retrieve the 9 permissions bits of the "/etc/passwd", it's owner, and it's group owner
        - If this fails, report error
    - If the owner is pfederl:
        - If user permissions do not have both read and write bits set, return 0, else return 1
    - Otherwise, if the group owner is one of "pfederl's" groups:
        - If group permissions do not have both read and write bits set, return 0, else return 1
    - Otherwise:
        - If other permissions do not have both read and write bits set, return 0, else return 1

This is actually quite similar to what the Linux kernel does, with some important simplifications:

- You must treat the case where username=root the same as any other user.
- You do not need to support symbolic links. We will not test your code on paths containing symbolic links.

Refer to the comments in `checkp.cpp` for instructions on how `checkp()` is expected to work. The comments contain information about the input parameters, return values, error codes, as well as a detailed description of the algorithm you must implement. You must follow these specifications and implement the algorithm as described in `checkp.cpp`. For example, the only system call you may use is lstat(2), and the only libc functions related to retrieving user IDs and group IDS are getpwnam(3) and getgrouplist(3).

## Submission

Submit `checkp.cpp` on Gradescope using the link posted on D2L. Do not modify nor submit any other files for grading. It is important that the file you submit works correctly with the rest of the original starter code.

Remove and/or disable any debugging output before you submit your code for grading. If you only use the provided dbg() function for debug output, you don't have to worry about disabling it.

## Marking

The Gradescope autograder will run several tests on your code, each graded independently. Many of the test cases will check for correctly reporting error conditions.

The autograder on Gradescope will be running Fedora 38. Unfortunately, in our department we have already upgraded all machines to Fedora 42, and we no longer have access to the older Fedora 38. However, you should still be able to do all your development and testing on our Linux computers (e.g. `cslinux.ucalgary.ca` or the Linux workstations on the main floor of MS), as the two versions of Fedora are sufficiently similar.

## Sample implementation

I made my own solution available for you to run on cslinux servers at this location:

```
~pfederl/public/cpsc525/a2/checkp-solution
```

You can use it to verify your own solution works correctly. For example, you can ask whether user "apache" can read file "/etc/passwd":

```
$ cd ~pfederl/public/cpsc525/a2
$ ./checkp-solution apache r /etc/passwd
Yes
```

The program reports a definite "yes". This is because both "/" and "/etc" are executable by all, and "/etc/passwd" is readable by all:

```
$ ls -ld / /etc /etc/fstab
dr-xr-xr-x.  23 root root  4096 Aug 20 06:56 /
drwxr-xr-x. 183 root root 12288 Oct  2 23:09 /etc
-rw-r--r--    1 root root   709 Dec 10  2023 /etc/fstab
```

But if you ask whether "apache" can both read and execute the same file:

```
$ ./checkp-solution apache rx /etc/passwd
No
```

Then the answer is "no", because nobody has execute permissions on "/etc/fstab".

There are cases where the checkp() fails to definitely answer with "yes" or "no", because the user that is running the program lacks sufficient permissions to get the information about the components of the path. For example, I have a file which I can access:

```
pfederl $ cat tests/private/joke.txt
Chuck Norris counted to infinity, twice!
pfederl $ ls -ld tests tests/private tests/private/joke.txt
drwxr-xr-x 4 pfederl profs 4096 Oct  3 00:01 tests
drwx------ 2 pfederl profs 4096 Oct  3 00:09 tests/private
-rw------- 1 pfederl profs   41 Oct  3 00:10 tests/private/joke.txt
```

A student cannot read the file or even establish whether the file exists:

```
$ cat tests/private/joke.txt
cat: tests/private/joke.txt: Permission denied
$ ls -l tests/private/
ls: cannot open directory 'tests/private/': Permission denied
```

This is because the parent directory "tests/private" does not have execute bit set for anyone except the owner ("pfederl"). This will cause checkp() to fail if a student runs it, because it will be unable to call lstat(2) on "tests/private/joke.txt".

```
$ ./checkp-solution pfederl r tests/private/joke.txt
error: EOPNOTSUPP (errno=95)
```

Oops, the program is not supposed to work on relative paths… Let's try with absolute path:

```
./checkp-solution pfederl r $(pwd)/tests/private/joke.txt
error: EACCES (errno=13)
```

My solution supports printing extra debug information, controlled by setting CHECKP_DEBUG=1:

```
$ env CHECKP_DEBUG=1 ./checkp-solution pfederl r $(pwd)/tests/private/joke.txt
getpwnam(pfederl) succeeded: uid=1436065 gid=110003
getgrouplist(pfederl) failed: ngroups=9
getgrouplist(pfederl) succeeded: returned value=9 110003 115250 110000 115269 110062
110057 110054 110047 115260
Requested permissions: 'r' -> mode=4
lstat(/) succeeded: mode=555 (r-xr-xr-x) uid=0 gid=0
lstat(/home) succeeded: mode=755 (rwxr-xr-x) uid=0 gid=0
lstat(/home/profs) succeeded: mode=755 (rwxr-xr-x) uid=0 gid=0
lstat(/home/profs/pfederl) succeeded: mode=711 (rwx--x--x) uid=1436065 gid=110003
lstat(/home/profs/pfederl/public) succeeded: mode=711 (rwx--x--x) uid=1436065
gid=110003
lstat(/home/profs/pfederl/public/cpsc525) succeeded: mode=755 (rwxr-xr-x) uid=1436065
gid=110003
lstat(/home/profs/pfederl/public/cpsc525/a2) succeeded: mode=755 (rwxr-xr-x)
uid=1436065 gid=110003
lstat(/home/profs/pfederl/public/cpsc525/a2/tests) succeeded: mode=755 (rwxr-xr-x)
uid=1436065 gid=110003
lstat(/home/profs/pfederl/public/cpsc525/a2/tests/private) succeeded: mode=700 (rwx--
----) uid=1436065 gid=110003
lstat(/home/profs/pfederl/public/cpsc525/a2/tests/private/joke.txt) failed: EACCES -
Permission denied
error: EACCES (errno=13)
```

You may find this useful when debugging your own code.

## Collaboration

As we discussed in our first lecture, limited collaboration is allowed on this assignment. You may discuss this assignment with one other classmate, and you may help each other with high-level design. However, you must write your answers by yourself (both written answers and coding questions), and you must declare your collaborator. Some amount of similarity between your submission and your collaborator's submission will be tolerated, but identical answers or answers with only cosmetic differences will not be accepted (we may ask you to resubmit if we detect too much similarity).

If you collaborated on the assignment with another student, you must put their name on the written answers Part of the assignment on Gradescope (first question), and on top of the `checkp.cpp` file in a comment.

## General information about all assignments

- All assignments are due on the date listed on D2L or Gradescope. Refer to the outline for late submission policy and relevant penalties.
- You can submit as many times as you like.
- If you have questions about the assignment, post them on Ed Discussion. If possible, make your questions public, so that the answers will be visible to everyone. If you need to include personal/sensitive information in your post, you can mark your post private, and only the TAs and the instructor will be able to see the question.
- Other than the limited collaboration allowed (as discussed during lectures), assignments must reflect your own work. Here are some examples of what you are not allowed to do for assignments: you are not allowed to copy code or written answers (in part, or in whole) from anyone else; you are not allowed to share your solutions (including code or pseudocode) with anyone; you are not allowed to sell or purchase a solution; you are not allowed to make your solutions available publicly, you are not allowed to use AI tools to generate answers. This list is not exclusive. For further information on plagiarism, cheating and other academic misconduct, check the information at this link: http://www.ucalgary.ca/pubs/calendar/current/k.html.
- We may use automated similarity detection software to check for plagiarism. Your submission will be compared to other students (current and previous), as well as to any known online sources. Any cases of detected plagiarism or any other academic misconduct will be investigated and reported.