UCLA Computer Science 97 Midterm (Practice) - Spring 2020

100 points total. Open notes, open computer.Forewarning: **this midterm is very hard** (like the real one will probably be)!! You are **far** from expected to get perfect on it. Don't stress if you don't know everything - use it as a guide to see what topics you need to spend more time on. Answer what you know first before questions you aren't as sure about.

1. (1 min.) What is **one** main difference between a hard link and a symbolic link?

   Both point to a file, but hard links increment a file's reference count whereas symbolic links do not.

   Other answers (not a complete list) include:

   - **A file is deleted only when all hard links to it are deleted/deleting symbolic links does not affect the underlying file**
   - **A symbolic link can become dangling if the underlying file is deleted**

2. (1 min.) Whenever you create a new POSIX directory, it will always have two other special directories inside of it by default. What are they, and what do they do?

   1. '.' which points to the new directory itself.

   2. '..' which points to the parent directory of the new directory.

3. (4 mins.) JavaScript is one of the most popular scripting languages, and is an essential tool for any aspiring web developer. It is an interpreted language; every web browser has its own built-in engine that can understand and execute JavaScript commands line-by-line.

   Suppose we have a web browser named Zafari. Its JavaScript engine is simple, and

executes JavaScript instructions line-by-line **without** any fancy optimization techniques such as a bytecode compilation step. Compare the execution of JavaScript code in Zafari versus the execution of Python code using the CPython interpreter. Which do you expect to be faster/have better performance?

When CPython "executes" a Python script, it first converts it into bytecode in a .pyc file and then executes the bytecode line-by-line. This improves performance at run-time and makes the process of importing other Python files much faster (since we only need to compile the other file to bytecode once, then that bytecode can be reused every time). JavaScript in Zafari does not have this intermediary step and must parse every JavaScript line literally at run-time. For these reasons, we expect Python running using CPython to be faster.

4. (20 mins.) Internet Protocol version 4 (IPv4) is a core protocol that is used to uniquely address devices with networking capabilities. It performs routing, which is the task of forwarding packets (chunks of data, like text messages, streaming videos, and literally everything else sent over the Internet) from one device to another.

   An IPv4 address typically looks like: `[0-255].[0-255].[0-255].[0-255]`, where `[0-255]` is a decimal number from 0 to 255 inclusive. The periods are literal and separate the four numbers. Examples of valid addresses include:

   - 127.0.0.1
   - 192.168.0.255
   - 32.45.8.21

a. (6 min.) Write an extended regular expression that only matches numbers between 0 and 255. It should not match numbers like “02” or “095”.

```
^([0-9])|([1-9][0-9])|(1[0-9][0-9])|(2[0-4][0-9])|(25[0-5])$
```

b. (2 mins.) Write an extended regular expression that matches IPv4 addresses.

```
^(((([0-9])|([1-9][0-9])|(1[0-9][0-9])|(2[0-4][0-9])|(25[0-5]))\.(([0-9])|([1-9][0-9])|(1[0-9][0-9])|(2[0-4][0-9])|(25[0-5]))\.(([0-9])|([1-9][0-9])|(1[0-9][0-9])|(2[0-4][0-9])|(25[0-5])))$
```

Pasting the answer to a) 4 times separated by ‘\.’ also works. (with parentheses in the right places)

c. (12 mins.) The shell command `ifconfig`, invoked without any arguments, displays the current configuration for the network interface on your local machine. Write a shell script that looks at the output from `ifconfig`. For each line of output, if it contains any IPv4 addresses your script should output the first (i.e. leftmost) one. Each address you output should be on its own line.

Answer:

```
#!/bin/bash

REGEX='(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])'

ifconfig |
grep -E "$REGEX" |
sed -E "s/.* ($REGEX).*/\1/g"
```

5. (10 min.) Consider the shell command being executed on a device named “ubun3” by user “groot”:

```
ubun3:~/zoo-wee-mama groot$ a b < c | ./d > e
```

   a. (2 mins.) What program(s) is/are being executed by the command?

      Two programs named “a” and “d”.

   b. (3 mins.) For each program you listed above, where will the shell look to find their executables?

      “a”: The shell will use the PATH environment variable and search each path specified in order, and will use the first “a” executable it finds.

      “d”: The shell will look in the current directory, i.e. ~/zoo-wee-mama

   c. (2 mins.) For each program you listed above, how many arguments is each being invoked with? What is the value of each argument?

      “a”: One argument with value ‘b’.

      “d”: No arguments.

   d. (2 mins.) What standard input does each program you listed above receive (if any)?

      “a”: The contents of the file in the current directory named “c”.

      “d”: The output of the “a” command.

   e. (1 min.) Where is the standard output of the above command written to?

      A file in the same directory named “e”.

6. (25 mins.)

   a. (20 mins.) Write a Python function named `py_spell` whose first argument is a string named `document`, and whose second argument is a **set** of words named `dictionary`. Let us define a *word* as a string with all non-alphabetic

characters removed.

Your function should return a **set** of all words in `document` that do not appear in `dictionary`. Words in the dictionary are case-sensitive. If a word in the dictionary contains any non-alphabetic characters, you should remove them before making any comparisons.

For example, if `document =`

`"Carole Baskin kill-ed her husband!"`

and `dictionary` contains

`{ Carole, baskin, killed, h-u-s-b-a-n-d888 }`

then your function should output a set containing `{ her, Baskin }`.

Answer:

```
def strip_nonalpha(string):
  return ''.join([char for char in string if char.isalpha()])

def py_spell(document, dictionary):
  # Create a dictionary where all punctuation is removed
  dict_no_punctuation = set(map(strip_nonalpha, dictionary))

  # Parse document
  blocks = document.split()
  words = set(map(strip_nonalpha, blocks))
  return set(filter(lambda word: word not in dict_no_punctuation, words))
```

BN: map and filter are fancy built-in functions (standard across languages) that can be easily implemented using for loops.

The `strip_nonalpha` function was created by Googling "remove non-alphabet characters from string python" and using a StackOverflow article.

b. (5 mins.) Compare `py_spell` to the `myspell` shell script that you implemented in Homework 1. Given that they are both supposed to be actual spell-checkers, how well does each work, i.e. what words don't work for `py_spell` that do for `myspell`, and vice versa?

They are both far from perfect. `myspell`'s main issue stems from the fact that the dictionary contains punctuation, so even if the word "dog" existed in the dictionary, "dog!" would count as a misspelled word. `py_spell`, on the other hand, can properly handle these cases. Its shortcoming is that there can be no dictionary word containing punctuation. "usergenerated" is certainly not a word but if "user-generated" is in the dictionary then it is treated as a properly spelled word.

7. (14 mins.) Consider the following keystrokes that are typed by a user in the shell in their home directory. In the transcript, a new line means the user typed "Enter"/"return" on the previous line. There is also an "Enter"/"return" after the final line. (Ignore the a, b, c, d, e; those are intended as markers.)

```
echo '#!/bin/bash' > sh
echo 'echo Hello, World!' >> sh            (a)
chmod 420 sh
emacs sh                                   (b)
Texttexttext                               (c)
M-x shell                                  (d)
./sh                                       (e)
```

a. (2 mins.) What does the file named "sh" contain after the keystrokes on line (a) are typed?

#!/bin/bash

echo Hello, World

b. (2 mins.) What happens when the keystrokes on line (b) are typed? Why?

emacs opens the file and its contents can be read and displayed to the user. This is because the chmod command grants the user read permissions. In fact, the only important number is "4" (binary 100) which gives the user read permissions, but no write or execute permissions.

c. (2 mins.) What happens when the keystrokes on line (c) are typed? Why?

The text cannot be entered because we don't have write permissions, as explained above.

d. (6 mins.) What happens when the keystrokes on line (d) are typed? In your answer, define what a "process" is and explain exactly what's happening.

A process is an instance of a program in execution. emacs itself is a process; the command M-x shell causes emacs to execute the shell program, which becomes a child process of emacs.

e. (2 mins.) What happens when the keystrokes on line (e) are typed? Why?

The "sh" script cannot be executed because we don't have execution permissions, as explained above.

8. (10 mins.) Jeve Stobs is a nerd who has been tinkering away on his own new operating system named FijiOS. He claims it's extremely fast, but since he's a one-man team he didn't have the time to make his OS POSIX-compliant. What are the benefits and drawbacks of Stobs' approach with FijiOS? If FijiOS takes off, what are the implications for software developers and FijiOS users alike?

Not conforming to POSIX definitely makes the development process faster and allows you to create your own interfaces for process management and file system operations. These interfaces could be simpler and perhaps even more performant than POSIX. However, there is a huge drawback in portability. The reason POSIX exists is to provide users, software developers, and hardware designers alike with a guaranteed interface that everyone can agree on; now, software developers will have to learn the functions and programs unique to FijiOS. The implications are that it will take much more work and time for developers to build versions of their programs that also work on FijiOS. Small/independent developers likely won't have time for this, and as a result FijiOS users will probably have a more limited set of programs that work on their devices.

9. (5 mins.) Suppose you're working on a project that uses Git for version control. You want to create and switch to a new branch named `feature/localization`, create an empty file named `strings.txt`, then add and commit **only** the new file to this branch, using the commit message `"added strings.txt"`. Write a list of shell commands that, when executed in order, will achieve the above.

git checkout -b feature/localization (1)

touch strings.txt

git add strings.txt (2)

git commit -m “added strings.txt” (3)

There are alternative commands.

(1) can be replaced with:

git branch feature/localization

git checkout feature/localization

(2) and (3) can be replaced with:

git commit -m “added strings.txt” strings.txt

10. (10 mins.) You’re in an alternate universe where Ariana Grande didn’t co-write and sing the song NASA, she became the chief scientist of a highly secretive NASA team instead. You work under her and help manage terabytes of sensitive data. It is extremely important that this data isn’t lost for whatever reason.

    Ariana Grande, as a scientist rather than an engineer, doesn’t know anything about failure models, and uses a simple policy: Only half of the digital storage available to your NASA team is in use at a time. Every Monday night, the entire contents of those machines is copied exactly as is to the other half of digital storage available. Whatever was in those machines is simply overwritten. Evaluate the effectiveness of Ariana’s approach.

    I didn’t watch the lecture on failures but I’m able to write this answer based on

lecture notes. Let's call this system AGS. AGS is a big thank u, next (i.e. it's really bad). Discuss possible scenarios where this approach is insufficient (it's insufficient for pretty much every single one) and provide ways you could improve things, e.g.:

a. Deleting files by mistake
    - If you delete a file by mistake on Sunday you need to restore the backup from last Monday, which might delete a lot of other work!!
    - Backups should happen much more frequently.
b. An outside attacker
    - AGS backups are not encrypted; data is copied as is.
    - Encrypt backups.
c. An inside attacker
    - AGS backups are accessible by everyone on the team
    - Use passwords or even physical security measures to ensure that only trusted team members have access to backups.
d. Redundancy
    - If the data is REALLY that important then we should have more than just one backup available at a time.
    - More copies/backups

You can also discuss space costs:

- Half of the total available storage to the team is being wasted because backups are not compressed.
- Use compression for backups.
- Talk about any of the other optimizations discussed in lecture (which you have access to)

    - **Data grooming**
    - **Deduplication**
    - **Multiplexing**
    - **Staging**

Finally, you can also talk about testing the system regularly:

- **Testing Recovery**
- **Checksumming**