SE-227::CSE

Assignment #4

End-to-end Argument

Q1: Description In my word, the end-to-end argument is a high-level strategy selection based on non-100%-reliable low level system.

It didn't consider every detail error corner cases at low level or try to recover from them. Oppositely, it just watches if the operation goes well at a higher level, and raises a simple retrial when somehow error is noticed.

It is simple to implement an end-to-end argument system for both high level and low level: High level programs doesn't care about the internal operations, and it just choose accepting or asking for a retrial. Meanwhile, low level programs doesn't care if the current operation is reliable or not. Anyway, high level programs will check its result and correct it if necessary.

Notice that this technique isn't suitable if the average error ratio is significant. Or you may fall into the "Retrying Trap" and suffer from great performance loss. Nor this technique is suitable to those operation that doesn't support redos or costs a lot.

Q2: Three Suitable Cases Here are three cases that meets the requirements of the "End to end" Argument.

Large File Copy According to Raymond Chan's *The Old New Things* that reveals many ideas about internal Windows implementation, we can learn that when we copy large files from local disks to removable devices, Windows will use the "End to end" Technique when copying large chunks.

Windows will first slice the big file into chunks, and calculate each chunk's checksum. Then Windows copies each chunk without caring about the correctness. After all copy operations are finished, Windows will check the checksum again to see if it matches the origin checksum. If any chunks has a invalid checksum, it will be erased and perform the copy operation again.

Analysis

- **Low error rate?** Yes. Under current hardware, it's extremely rare when a copy operation could fail. So it's quite suitable for the "End-to-end" error handling technique.
- **Sophisticated Internals?** Somehow yes. Copying files must adapt to different file systems that based on various hardwares. So it's nearly impossible to conclude a recovery manual at all.
- **Redoable?** Absolutely! If you can copy a file once, you can copy it any times you want. When you noticed the error, the copy operation isn't

completed yet. So you can declare a retry (for one single block or the whole file) with confidence.

Network APIs As a client that requires server to handle its API calling, the "End-to-end" technique also applies to this circumstance. In order to figure out if the server did its job correctly, the best way is to check its `status` state or calculates the checksum.

Clients are neither responsible nor qualified to handle or recover the servers' faults. The best way and only way for a client application is to check the response, and determines to accept it, or refuse it.

Analysis

- **Low error rate?** Not really, but when error pops up, there's nothing more you could do than a retrial.
- **Sophisticated Internals?** Totally yes! Internet is so large and it's impossible to trace where does the error occurs. The best way is do it again and again, until it goes right.
- **Redoable?** Absolutely! If you can copy a file once, you can copy it any times you want. When you noticed the error, the copy operation isn't completed yet. So you can declare a retry (for one single block or the whole file) with confidence.

Boss and employees It's not strictly an example in the computer field, but somehow it matches the idea quite well.

As a boss (higher level hierarchy consuming high salary), you must want to be bossy and order your employees high at top.

However your employees make mistakes sometimes, mainly based how much you pay them. So when you notices that a command from you didn't get accomplished well, what will you do?

Will you digest into the whole procedure, see where it goes wrong, and reconsider your order?

Or, will you just simply order the same thing again without even thinking?

Or, will you give your disappointing employees an angry denounce and fire them all?

Q3: Three Unsuitable Cases

Download Checksums As a download tool, Thunder was once very popular. But it used to have a big disadvantage: Downloading big files will cause long-time hanging at 99.9%.

Sometimes it will even fail after long-time wait, which annoys many users.

That's because almost all files that has a calibrating checksum is a whole one. That means it can't determine which part of this file goes wrong even it lies at the very beginning. And all later downloading operations are all in vain.

Now WinRAR has a better chunk-sliced checksum that could determine each part's integrity. And if your download goes wrong on the half way, it could recognize this immediately.

Compilers If we use the end-to-end idea into the compiler developing, that would be a stupid idea. That means the only error message you can acknowledge is "Passed!" or "Failed!". Totally no internal error handling.

That must be very frustrating for programmers who are always making mistakes.

Lab Grading Once upon a time, TAs use carefully written shell scripts to grade our tests that could give scores in stairway based on which step you've conquered. However if we use the "end-to-end" idea, students could get only 0 or 100 as their result, which is quite unfair for those who completed the job partially who are failed to be distinguished from those who did nothing.

Q4: inode FS Improvements In my mind, the meta-data system shouldn't be limited in the file system. The only thing that the file system should do is to create inodes, allocate blocks and ensure the integrity when writing and reading blocks.

More than that, the directory's structure is also hard-encoded in the file system, which I don't think quite necessary. It should be flexibly implemented in user state.

Q5: OS Improvements I think the base line for an OS is the system call. OS should preserve the secrets that happens when a system call is operated.

If a user-state application finds something goes wrong when making a system call, the only thing it could do is to make the system call again.

All these things that goes above the system, like cstdlib or something like that should provide their own node and use their favorite error handling method. That's not something an OS should consider.