CS 240
Computer Systems

#14: API Programming and Virtualizations

March 8, 2022 · Wade Fagen-Ulmschneider

Sending HTTP Requests:

In Python, the **`requests`** library provides us the ability to make HTTP requests to external APIs:

14/api.py

```python
import requests

r =
   requests.get("https://www.colr.org/json/color/random")
print(f"Status Code: {r.status_code}")
print(f"Character Encoding: {r.encoding}")
```

- **`requests.get(...)`** sends a GET request,
- **`requests.post(...)`** sends a POST request,
- **`requests.put(...)`** sends a PUT request,
- ...etc...

The requests library is just a wrapper around the request and response from any HTTP web service:

14/api.py

```python
print("== Headers ==")
for header in r.headers:
  print(header + ": " + r.headers[header])

print("== Payload (text) ==")
print(r.text)

print("== Payload (json) ==")
data = r.json()
print(data["colors"][0]["hex"])
```

Note that **`r.text`** returns the response as a string (at attribute). **`r.json()`** parses it for us into a dictionary for us to index into quickly (it's a function, requires the parameters)!

Receiving HTTP Requests:

The flask library allows us to receive HTTP requests:

14/app.py

```python
from flask import Flask
app = Flask(__name__)

@app.route('/', methods=["GET"])
def index():
  return "index function!"

@app.route('/', methods=["POST"])
def post():
  return "post function!"

@app.route('/hello', methods=["GET"])
def hello():
  return "hello function!"

@app.route('/hello/<id>')
def with_id(id):
  return f"with_id function: {id}"

@app.route('/hello')
def mystery():
  return "mystery function!"
```

What happens with the following requests:

1. `GET /`
2. `POST /`
3. `PUT /`
4. `GET /hello/`
5. `GET /hello`
6. `POST /hello`
7. `PUT /hello`
8. `GET /hello/42`
9. `GET /hello/world`

Operating Systems: A Great Illusionist

Throughout this entire course, we have discussed how the operating system abstracts away the complexity of real systems:

- As a process, it appears that we have __________________.
- ...and has ______________________________________!

Do we need additional abstractions?

Virtualization

Big Idea:

- All states $\mathbf{S_X}$ can be represented on a host system $\mathbf{H(S_X)}$.
- For all sequences of transitions between $\mathbf{S_1} \Rightarrow \mathbf{S_2}$, there is a sequence of transitions between $H(\mathbf{S_1}) \Rightarrow H(\mathbf{S_2})$.

What is a "machine"?

- Language Virtualization:
- Process Virtualization:
- System Virtualization:

Language Virtualization: Example w/ a JVM

Initial State ($\mathbf{S_1}$):		
Transition ($\mathbf{S_1} \Rightarrow \mathbf{S_2}$):		

System #1
`COPY r1 1`
`SHIFTL x 2`
`ADD x r1`
	System #2
	`COPY r1 x`
	`SHIFTL x`
	`SHIFTL x`
	`ADD x r1`
		System #3
		`COPY r1 x`
		`ADD r1 x`
		`ADD r1 x`
		`ADD r1 x`
		`ADD r1 x`

Final State ($\mathbf{S_2}$):		

Process Virtualization: Example w/ Rosetta and the M1 chip

Initial State ($\mathbf{S_1}$):
Transition ($\mathbf{S_1} \Rightarrow \mathbf{S_2}$):
Final State ($\mathbf{S_2}$):

System Virtualization: Your CS 240 Virtual Machine / EC2

- Type 1 Hypervisor:
- Type 2 Hypervisor:

Q: How has this changed the deployment of software?