Debugging and Profiling

Debugging

Printf debugging and Logging

> "The most effective debugging tool is still careful thought, coupled with judiciously placed print statements" — Brian Kernighan, *Unix for Beginners*.

Approaches for Debugging

1. Add print statements around the problem area
2. Use **logging** instead of ad hoc print statements

Why Logging?

- Log to files, sockets, or remote servers
- Supports severity levels (INFO, DEBUG, WARN, ERROR, etc.)
- Logs can contain information for new issues

Example Code for Logging

```
$ python logger.py
$ python logger.py log
$ python logger.py log ERROR
$ python logger.py color
```

Coloring Logs

- Terminals use colors for readability
- Uses ANSI escape codes
- Example for color coding: `echo -e "\e[38;2;255;0;0mThis is red\e[0m"`

Script for Printing RGB Colors

```
#!/usr/bin/env bash
for R in $(seq 0 20 255); do
    for G in $(seq 0 20 255); do
        for B in $(seq 0 20 255); do
            printf "\e[38;2;${R};${G};${B}m \e[0m";
        done
    done
done
```

Third Party Logs

- Dependencies such as web servers and databases generate their own logs
- Logs are often stored under `/var/log` in UNIX systems
- Use `systemd` or `log show` on macOS to view system logs

Using `logger` for System Logs

```
logger "Hello Logs"
log show --last 1m | grep Hello
journalctl --since "1m ago" | grep Hello
```

Debuggers

- Allows interaction with program execution
- Features: halt, step through, inspect variables, set breakpoints

Python Debugger (`pdb`)

- **l**(ist)
- **s**(tep)
- **n**(ext)
- **b**(reak)
- **p**(rint)
- **r**(eturn)
- **q**(uit)

Buggy Python Code Example

```python
def bubble_sort(arr):
    # ... buggy code ...
    return arr

print(bubble_sort([4, 2, 1, 8, 7, 6]))
```

Specialized Tools

- Tools for debugging black box binaries
- Tracing system calls with `strace` or `dtrace`

Using `strace` or `dtruss`

```
sudo strace -e lstat ls -l > /dev/null
sudo dtruss -t lstat64_extended ls -l > /dev/null
```

Profiling

Timing

- Measure time between code points
- Difference between *Real*, *User*, and *Sys* time

Python Timing Example

```python
import time, random

start = time.time()
# ... some work ...
print(time.time() - start)
```

Profilers

CPU Profilers

- **Tracing** vs **Sampling** profilers
- `cProfile` in Python for function call profiling

Python `cProfile` Example

```
$ python -m cProfile -s tottime grep.py
```

Line Profilers

- Shows time taken per line of code
- Example with `line_profiler`

Memory Profilers

- Identify memory leaks with tools like Valgrind
- Use memory profiler for languages like Python

Python Memory Profiler Example

```
@profile
def my_func():
    # ... code ...
    return a
```

Event Profiling

- Black box profiling with `perf`
- Reports system events related to programs

Visualization

- Flame Graphs for hierarchy of function calls
- Call graphs for function relationships

Resource Monitoring

- `htop`, `iotop`, `df`, `du`, `free`, `lsof`, `ss`, `nethogs`
- Artificial loads with `stress`

Specialized tools

- Black box benchmarking with `hyperfine`
- Profiling webpage loading with browser developer tools