Debugging and Profiling

2021 Winter ABC Mentoring Program

Team 5 Mentor 정현준

2021 / 02 / 01

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

지난 시간에...

- SSH key를 이용하기 위해서는 각 파일들이 적절한 permission을 가지고 있어야 합니다.
- 지난 시간에서도 언급하였듯, private key는 유출이 될 경우 보안에 문제가 발생하기 때문에 각 파일들은 아래와 같은 permission을 유지해야 합니다.

```
chmod 700 ~/.ssh
chmod 600 ~/.ssh/id_rsa
chmod 644 ~/.ssh/id_rsa.pub
chmod 644 ~/.ssh/authorized_keys
chmod 644 ~/.ssh/known_hosts
```

chmod

- 특정 파일이나 폴더의 이용 권한을 바꾸는 명령어 입니다.
- chmod [변경할 권한] [파일 또는 폴더]
- 변경할 권한은 숫자 3자리로 나타내며, 각 자릿수는 소유자, 그룹, 기타 사용자를 나타냅니다.

Ex) chmod 754 test.py

파일의 소유자는 7 (읽기, 쓰기, 실행) 권한
그룹은 5 (읽기, 실행) 권한
기타 사용자는 4 (읽기) 권한

r : 읽기 w : 쓰기 x : 실행

0 = --- = 0 + 0 + 0
1 = --x = 0 + 0 + 1
2 = -w- = 0 + 2 + 0
3 = -wx = 0 + 2 + 1
4 = r-- = 4 + 0 + 0
5 = r-x = 4 + 0 + 1
6 = rw- = 4 + 2 + 0
7 = rwx = 4 + 2 + 1

Debugging

- 프로그램 내부에서 Fault, Error, Failure를 찾아내고 고치는 것을 디버깅이라고 합니다.
- Failure: 프로그램의 잘못된 결과 (잘못된 output이나 slow running 등을 의미)
- Error: 올바르지 않은 프로그램의 내부 상태
- Failure: 잘못된 코드 오타 같은 결함을 의미
- 일반적으로 프로그램 개발에서 가장 많은 시간을 차지합니다.

Print debugging and Logging

- Print debugging은 가장 간단한 방법이자 많이 사용하는 방식 입니다.
- 코드 중간에 터미널에 출력을 할 수 있는 print(Python)나 printf(C언어)를 사용합니다.
- Logging은 파이썬에서 print와 같은 standard output 대신 프로그램에서 발생하는 이벤트를 추적하는 내장 모듈입니다.
- import logging을 통해 프로그램에서 사용할 수 있습니다.
- Logging의 큰 장점은 각 이벤트가 포맷(debug, info, warning, error, critical)에 따라 log의 레벨이 구분된다는 점입니다.

Third Party Logs

- UNIX 계열의 OS 경우, 프로그램을 실행하면서 시스템은 프로그램에서 발생한 log를 특정 장소에 기록하기도 합니다.
- 일반적으로 이러한 log들은 /var/log 위치에 기록됩니다.
- Linux 계열의 경우, systemd라는 시스템에 관련된 로그를 수집하여 /var/log/journal에 저장하는 기능이 존재합니다.
- 저장된 log는 journalctl 등의 명령어를 이용해 확인할 수 있습니다.

PDB debugger

- GCC의 GDB와 비슷한 Python 전용 debugger 입니다.
- python –m pdb [디버깅을 하고 싶은 py 파일]

```
[cs20151509@uni06 pdb_example]$ python -m pdb bubble_sort.py
> /students_home/old_students/cs20151509/ABC_program/pdb_exam
1)<module>()
-> def bubble_sort(arr):
(Pdb)
```

PDB debugger

- PDB는 내부에서 다양한 명령어를 지원합니다.

  l: 현재 line을 기준으로 11줄 정도의 코드를 보여줍니다.

  s: 현재 line을 실행합니다. (함수를 만나면 함수 내부로 진입)

  n: 현재 함수에서 다음 line을 실행합니다.

  b: break point를 설정합니다.

  p: 주어진 값을 계산하고 화면에 출력합니다.

  r: 현재 함수에서 return을 만날 때까지 실행을 진행합니다.

  q: 디버깅을 종료합니다.

참고: https://docs.python.org/ko/3.7/library/pdb.html

strace

- Application (응용 프로그램)이 사용하는 system call과 signal등을 추적하는 명령어 입니다.
- macOS의 경우 dtrace가 있습니다.
- Ex) strace open ls
- 우리가 ls 명령어를 사용할 때, ls 명령어의 실행 도중에 open이라는 system call을 어디서 사용하는지 추적해줍니다.

Profiling

- 프로파일링 혹은 성능 분석이라고도 합니다.
- 최적화 문제와 연결되기 때문에 시스템 분야에서 매우매우매우매우 중요합니다.
- 일반적으로 프로그램을 사용할 때, CPU의 utilization이나 memory 사용량 같은 컴퓨터의 resource가 얼마나 사용되는지를 분석합니다.
- 프로그램 자체에서 사용자가 코드를 통해 Profiling을 하기도 하지만 일반적으로 외부의 tool에 도움을 받아 하는 경우가 많습니다.

Time 측정

- Linux에는 time이라는 명령어를 통해 프로그램이 실행되고 종료될 때 걸리는 시간을 측정할 수 있습니다.
- time [실행할 명령] 으로 사용합니다.

```
[cs20151509@uni06 pdb_example]$ time python bubble_sort.py
Traceback (most recent call last):
  File "bubble_sort.py", line 10, in <module>
    print(bubble_sort([4,2,1,8,7,6]))
  File "bubble_sort.py", line 5, in bubble_sort
    if arr[j] > arr[j+1]:
IndexError: list index out of range

real    0m0.051s
user    0m0.033s
sys     0m0.017s
```

Real : 프로그램 시작에서 종료까지 모든 요소들이 실행되는데 걸린 시간

User : CPU가 프로그램에서 유저 코드를 실행하는데 사용한 시간

Sys: CPU가 프로그램에서 커널 코드 (시스템 코드)를 실행하는데 사용한 시간

Time 측정

- 앞서 time 명령어는 프로그램 전체를 측정하는데 사용됩니다.
- 만약 프로그램 내부에서 어떤 하나의 명령어를 실행하는데 걸리는 시간을 측정하고 싶다면, 각 언어에 존재하는 time library를 사용해야 합니다.
- 예를 들면, Python에는 time이라는 모듈이 존재합니다.

```
1 import time
2
3 start = time.time()
4 print("Hello World!")
5 end = time.time()
6
7 print(end - start)
```

Print 함수가 실행되는데 얼마나 시간이 걸리는지를 측정합니다.

cProflie

- Python에서 사용되는 CPU Profiler 입니다.
- CPU를 통해 프로그램의 어떤 부분이 얼마나 자주, 오래 사용되었는지 분석할 수 있도록 통계를 보여줍니다.
- python –m cProfile [프로파일링을 하고 싶은 py 파일]
- 참고: https://docs.python.org/ko/3/library/profile.html

```
[cs20151509@uni06 ABC_program]$ python -m cProfile time_profiling.py
Hello World!
3.79085540771e-05
         4 function calls in 0.000 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 time_profiling.py:1(<module>)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        2    0.000    0.000    0.000    0.000 {time.time}
```

memory_profiler

- pip install memory_profiler 명령어로 설치 가능합니다.
- Decorator(@)를 통해 profiling을 원하는 함수를 선택 가능합니다.

```
[cs20151509@uni06 pdb_example]$ python -m memory_profiler memory_test.py
Filename: memory_test.py

Line #    Mem usage    Increment   Line Contents
================================================
     1   24.492 MiB   24.492 MiB   @profile
     2                             def test_func():
     3   32.125 MiB    7.633 MiB       a = [1] * (10 ** 6)
     4  184.715 MiB  152.590 MiB       b = [2] * (2 * 10 ** 7)
     5   32.125 MiB    0.000 MiB       del b
     6   32.125 MiB    0.000 MiB       return a
```

top, free, df + 그 외

- Linux에서 resource monitoring을 할 수 있는 유용한 명령어들입니다.
- top : 프로세스별로 CPU 사용량과 할당중인 코어에 대해 볼 수 있습니다.
- free : memory 사용량과 swap memory 사용량에 대해 볼 수 있습니다.
- df : disk storage 사용량에 대해 볼 수 있습니다.
- nvidia-smi : GPU 사용량에 대해 볼 수 있습니다. (주로 딥러닝에 사용)

Assignment

- 다음 수업인 Metaprogramming에 대해 강의를 보고 옵시다.
- https://missing.csail.mit.edu/2020/metaprogramming/

Thank you


UNIST
ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY
2009