# junpkim

github: https://github.com/junppyo/exem

▼ Week1

## info

### cpu

/proc/stat, 첫 번째 줄에 총 cpu 정보가 나옴.

필요한 data: usr, sys, idle, iowait

첫 번째 줄 2, 4, 5, 6번째 칸이 각각 cpu usr, sys, idle, iowait 에 대한 정보.

https://medium.com/@yogita088/how-to-calculate-cpu-usage-proc-stat-vs-top-e74f99f02d08

### memory

/proc/meminfo

필요한 데이터: free, total, used, swap

total: 첫 번째 줄

free: 두 번째 줄

used:

```
used = total - free - buffer - cache
```

**하지만 buffer/cache 데이터가 proc를 통해 수집한 값과 free나 top 명령어로 얻은 값이 달라 used 값 역시 다르게 나온다.**

swap: 15번째 줄

## packet

/proc/net/dev

필요한 데이터: send packet, byte, receive packet, byte

데이터는 3번째 줄에서 시작되고 2, 3, 10, 11번째 열에서 얻을 수 있다.

## process

/proc 폴더를 읽어 데이터 확인. 읽은 데이터의 유형이 디렉터리가 아니거나 이름이 숫자가 아니면 패스하여 pid를 얻음.

/proc/[pid]/stat 을 읽어 name, ppid, cpu 점유율 계산을 위한 데이터를 얻음.

/proc/[pid]/attr 을 stat 함수를 이용해 uid를 얻고, getpwuid 함수를 이용해 username을 얻음.

sysconf 함수를 이용해 cpu clock을 얻음.

/proc/uptime 을 읽어 uptime을 얻음.

cputime = (utime + stime) / clock

cpu 사용량 = ((utime + stime + cutime + cstime) / clock) / (uptime - (starttime / clock)) * 100

```
Pid %d: process ID
Comm %s: 괄호 안은 실행 가능한 파일이름.
State %c: "RSDZTW"중 한 개의 문자. R은 running, S는 interrupt가능한 sleeping, D interrupt
 불가능한 disk sleep 상태의 waiting, Z zombie, T is traced or stopped (on a signal), and
 W is paging
Ppid %d: 부모의 PID
Pgrp %d: process의 process group ID
Session %d: process의 session ID
Tty_nr %d: The controlling terminal of the process.
tpgid %d:
flags %u (%lu before Linux 2.6.22):
minflt %lu:
cminflt %lu:
```

```
majflt %lu:
cmajflt %lu:
utime %lu: user mode에 scheduled 되는 process의 시간의 양. Clock ticks로 언급됨. 이것은 guest time을 포함한다. 따라서 application은 계산에서 손실되는 guest time 부분의 시간을 인식하지 않아도 된다.
stime %lu: kernel mode에서 scheduled 되는 process의 시간의 양. Clock ticks로 언급됨.
Cutime %ld: user mode에서 process가 children을 기다리는 시간의 양. Clock ticks로 언급됨.
Cstime %ld: kernel mode에서 process가 children을 기다리는 시간의 양. Clock ticks로 언급됨.
Prority %ld:
Nice %ld:
Num_threads %ld: process의 thread 수.
 Iteralvalue %ld:
Starttime %llu:
Vsize %lu:
Rss %ld:
Rsslim %lu:
Startcode %lu: The address above which program text can run.
Endcode %lu: The address below which program text can run.
Startstack %lu: the address of the start (i.e. bottom) of the stack.
Kstkesp %lu: The current value of ESP (stack pointer), as found in the kernel stack page for the process.
Kstkeip %lu: The current EIP (instruction pointer)
Signal %lu:
Blocked %lu:
Sigignore %lu:
Sigcatch %lu:
Wchen %lu:
Nswap %lu:
Cnswap %lu:
Exit_signal %d:
Processor %d:
Rt_priority %u:
Policy %u:
Delayacct_blkio_ticks %llu:
Guest_time %lu:
Cguest_time %ld:
```

# struct

## osinfo

unsigned long cpu_usr

unsigned long cpu_sys

unsigned long cpu_iowait

unsigned long cpu_idle

unsigned long mem_free

unsigned long mem_total

unsigned long mem_used

unsigned long mem_swap

unsigned long packet_in_cnt

unsigned long packet_out_cnt

unsigned long packet_in_byte

unsigned long packet_out_byte

## procinfo

char name[256]

char uname[32]

int pid

int ppid

float cpuusage

float cputime

char cmdline[4096]

struct procinfo *next

**문자열, 특히 cmdline이 최대 4096byte까지 존재 하는데 용량을 줄일 방법 없을까?**

⇒ int형 변수 cmdline_len 을 멤버변수로 추가. server 에서 구조체 수신 후 cmdline_len 만큼 cmdline을 동적할당 해 준 뒤 cmdline을 한 번 더 수신 하도록 변경.

▼ Week2

packet

{

osinfo *osinfo

```
    int proc_len    // 전송할 프로세스 갯수
    plist *proc
    {
        procinfo *HEAD
        procinfo *TAIL
    }
    packet *next
}
```

▼ Week3

# 구조체

구조 대대적 변경.

osinfo → cpuinfo, meminfo, netinfo로 분리

모든 info 구조체 연결리스트로 구현

packet 구조체를 queue로 사용 → packet 안에 각 info 구조체의 queue를 넣음.

packet 구조체에 각 info 수집 스레드의 mutex 추가.

전송 전 보낼 header 추가. (packethead 구조체. 현재는 id와 type만 존재)

# 스레드

스레드 계속 살아있도록 변경.

queue에 append, pop시 mutex lock, unlock 추가.

client

main thread - tcp 패킷 전송

- thread1 - cpu 정보 수집
- thread2 - memory 정보 수집
- thread3 - network 정보 수집

thread4 - process 정보 수집

server

main thread - tcp 수신

thread1 - 받은 패킷 DB 저장

# DB

mariaDB 사용.

이유: 익숙하고 편해서 ㅎㅎ;

DB 구조: 테이블과 동일.

unixodbc 시도중.

## DDL

Data Definition Language. 데이터를 생성하거나 수정, 삭제 등의 역할을 하는 언어.

ex) CREATE, ALTER, DROP, TRUNCATE

## DML

Data Manipulation Language. DB에 저장된 레코드를 조회하거나 수정 삭제 등의 역할을 하는 언어.

ex) SELECT, INSERT, UPDATE, DELETE

▼ Week4

```
MariaDB [exem]> show tables;
+----------------+
| Tables_in_exem |
+----------------+
| cpuinfo        |
| diskinfo       |
| meminfo        |
| netinfo        |
| procinfo       |
| udp            |
| udpmatric      |
+----------------+
7 rows in set (0.000 sec)

MariaDB [exem]> desc cpuinfo;
+--------------+------------------+------+-----+---------------------+-----------------------------+
| Field        | Type             | Null | Key | Default             | Extra                       |
+--------------+------------------+------+-----+---------------------+-----------------------------+
| id           | int(11)          | YES  |     | NULL                |                             |
| usr          | int(10) unsigned | YES  |     | NULL                |                             |
| sys          | int(10) unsigned | YES  |     | NULL                |                             |
| iowait       | int(10) unsigned | YES  |     | NULL                |                             |
| idle         | int(11)          | YES  |     | NULL                |                             |
| collect_time | timestamp        | NO   |     | current_timestamp() | on update current_timestamp() |
| idx          | int(11)          | NO   | PRI | NULL                | auto_increment              |
| deltausage   | float            | YES  |     | NULL                |                             |
+--------------+------------------+------+-----+---------------------+-----------------------------+
8 rows in set (0.001 sec)
```

DB table은 코드 내 구조체를 그대로 구현함.

▼ Week5

# Preload

https://blog.dork94.com/145

Preload는 실행될 때 호출되는 constructor와 함수를 후킹하는 단, 종료될 때 호출되는 destructor로 구성된다.

후킹하려는 함수와 동일한 함수명으로 함수를 구현하면, 해당 함수가 호출 되기 전에 후킹 한다.

## dlysm

```
void *dlsym(void *handle, const char *symbol);
```

loading한 shared object에서 함수 또는 변수의 pointer를 반환해주는 함수.

포인터 변수의 형태는 함수의 원형과 동일해야만 한다.

preload 파일에서 후킹을 위해 구현한 함수가 아닌 원본 함수를 가져와야하기 때문에 shared object에서 두 번째에 있는 함수의 주소를 가져와야 한다.

때문에 handle 매개변수에는 처음에 찾은 주소가 아닌 2번째 주소를 의미하는 RTLD_NEXT 를 적어주고, symbol에는 찾으려는 함수명을 적어준다.

컴파일 할 때는

```
gcc -fPIC -shared -o preload.so preload.c -ldl
```

# UDP

## packet

원본 패킷을 보내기 전에 전송하는 begin 패킷과 원본 패킷을 보낸 후에 전송하는 end 패킷으로 구성되어 있다.

### begin packet

clientid

preload가 호출된 pid

전송하려는 소켓의 ip와 port

전송이 시작된 시간

### end packet

clientid

preload가 호출된 pid

전송된 byte

전송에 걸린 시간

udpmatric을 보내기 전에 활성화되는 flag

server에서 받은 뒤에는 하나의 패킷으로 처리.

▼ Week6

# 추가 기능 구현

## Disk 사용량 정보 수집

df 명령어를 사용하기 위해 popen 사용.

popen 사용시 SIGCHLD 시그널이 발생하는 것을 발견. 무시하도록 설정.

패킷과 DB 테이블은 기존과 같은 방식으로 구현.

## Delta, Average 계산

자료구조를 양방향 연결리스트로 구현.

연결리스트에는 노드들의 갯수와 값 합계를 저장하는 변수를 가지고 있음.

Delta 데이터의 경우 새로운 데이터를 push 할 때 가장 앞에 들어있던 데이터와의 차이를 반환하도록 함.

Delta 데이터는 각 DB 테이블에 Delta 컬럼을 추가하여 저장.

Average 데이터의 경우 리스트에 저장되어 있는 합계와 노드 갯수를 이용해 계산.

데이터가 수집된 지 한 시간이 지나면 리스트에서 삭제.

## 임계치 이상 데이터 별도 저장

DB 테이블에 index를 추가. 임계치를 넘어간 데이터의 index를 파일로 저장하도록 함.

# 수정사항

- 서버와 연결이 끊길 경우 수집 스레드 종료 하도록 변경. 재연결시 수집 스레드 다시 실행.

- 서버에 의해 클라이언트와의 통신이 끊긴 경우(잘못된 패킷 수신)에도 같은 클라이언트에서 재접속할 수 있도록 변경.
- mysql 재연결 코드 추가.

# 알려진 오류

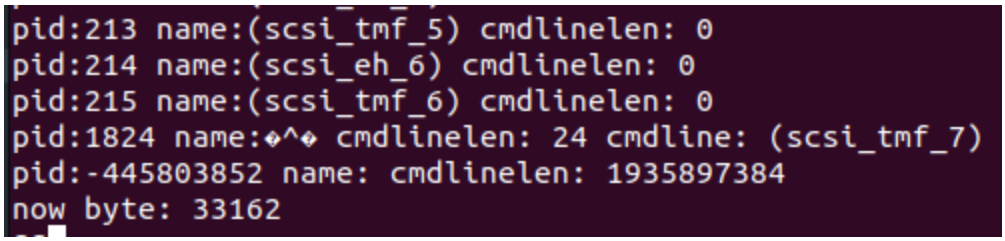
- 랜덤하게 프로세스의 cmdline을 받는 과정에서 잘못된 패킷을 받는 것으로 로그가 남음.

  자세한 확인 결과 cmdline을 잘못 받는 것이 로그상 에러가 발생하기 전 패킷을 잘못받아 잘못된 cmdline_len을 얻음.


~~오류가 나기 전까지의 byte 합계를 구해보니 항상 65000 byte 안팎에서 발생하는 것으로 확인되어 tcp 패킷 길이 제한인 65535byte에 걸려 잘린 것이 아닐까 하는 추측.~~

→ 6만byte 단위로 끊어서 전송해보았으나 해결되지 않음.

→ 6만byte 마다 헤더를 새로 붙여서 전송. ~~해결됨~~ 해결되지 않음.




6만 byte씩 끊어서 전송하니 33000 안팎에서 에러가 발생하는 것으로 확인됨.

VM이 아닌 윈도우 os에 실행되어있는 게임을 종료하니 빈도가 줄어들은 듯한 체감이 듦.

- 아주 가끔

```
double free or corruption ( prev)
aborted
```

에러 발생.

→ 앞의 에러와 연관이 있을 것으로 추정됨.

- 서버 재실행시 클라이언트가 재접속 할 때 랜덤하게 seg fault 발생.

- 수집주기를 0.1초로 약 2분이상 돌리면 memtotal의 값에서 이상이 생김.