**Dynamic vCPU Allocation Module**

**- Cloud Computing Project –**

2011140138

황 태림

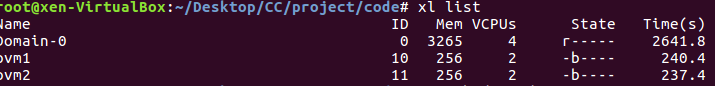
**개요**

**과제 구현 목표**

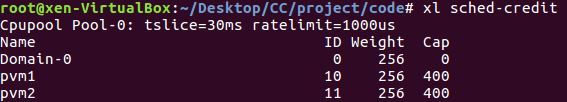
본 과제는 가상화 환경 구축과 가상 환경의 VCPU를 효율적으로 관리하여 최대 효율을 끌어내는 것을 목표로 한다. DVAM은 가상머신의 VCPU를 각 가상머신의 CPU 이용률에 따라 동적으로 재 할당하여 하이퍼바이저의 자원을 효율적으로 사용할 수 있도록 하는 모듈이다. 이러한 DVAM을 만들어 Linux 프로그래밍에 대한 지식 습득과 활용을 목표로 한다.

**가상머신 설정**

가상머신에 대한 설정은 다음과 같다.



[초기 VCPU의 수와 memory size]

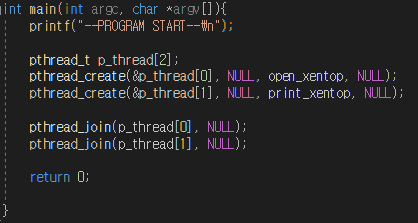
****

[가상머신의 weight 및 cap]

**본문**

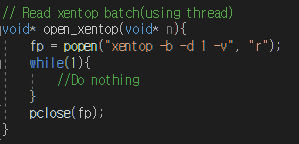
**코드 설명**

DVAM 코드에 대한 설명은 다음과 같다.



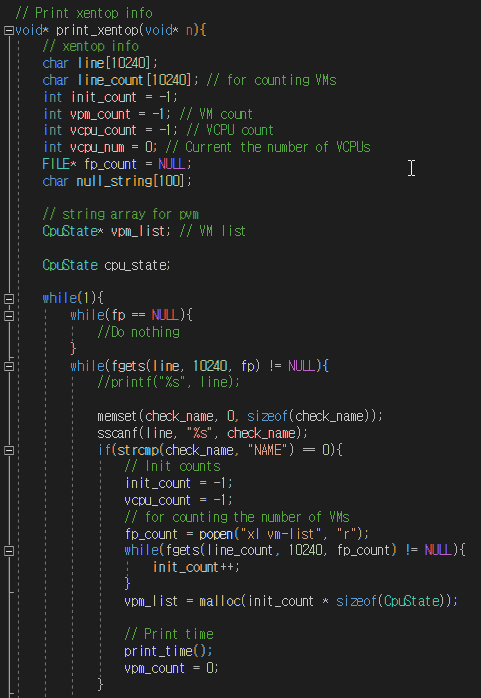
[main 함수]

* 쓰레드를 생성하여 실행한다.
* 쓰레드는 1)xentop batch를 읽어오는 것과 2)DVAM 처리 후 출력되는 함수를 실행하는 것 두 가지가 작동한다.



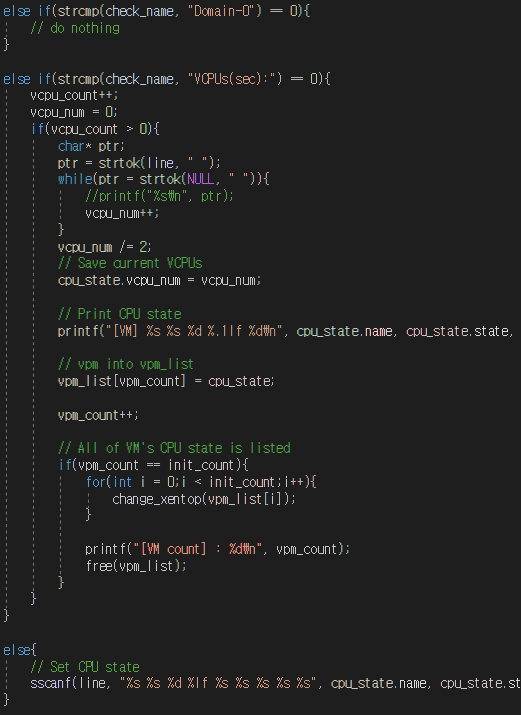
[open\_xentop 함수]

* xentop을 batch로 읽어온다.
* Delay는 1초로 주었으며, -v 옵션을 통해 가상머신의 현재 VCPU의 개수를 셀 수 있도록 하였다.



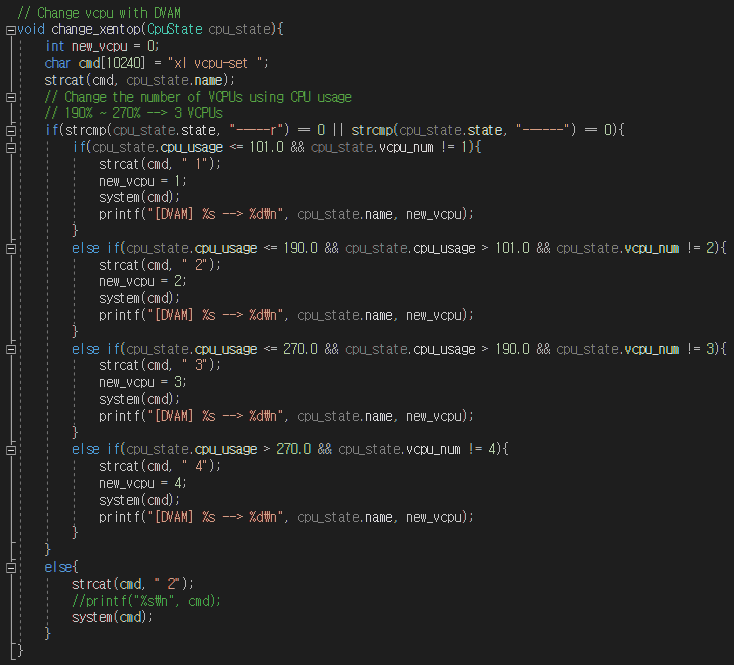
[print\_xentop 함수] - 1

* 초기화된 변수 중에서 count 부분이 모두 -1인 이유는 최초 column 부분을 셀 때 +1 해서 초기화 하려는 의미로 설정한다.
* CPU state를 저장하는 구조체와 이 구조체를 저장하는 동적 할당된 배열을 통해 모든 VM에 대한 CPU state를 처리한다.
* 프로그램 실행 후 쓰레드 처리 시 최초 File pointer가 아무것도 읽지 못하는 동안의 처리를 추가한다.
* terminal에서 한 줄씩 받아온 후 memset을 통해 첫 번째 단어를 check하여 필요한 line만 처리한다.
* NAME으로 시작하는 라인은 column 이므로 해당 라인에서 카운트 초기화한다.
* xl vm-list 명령어를 통해 가상머신이 총 몇 개인지를 init\_count로 계산한다(-1인 이유는 column 부분 때문).
* print\_time() 으로 읽어 들인 시간을 출력한다.



[print\_xentop 함수] – 2

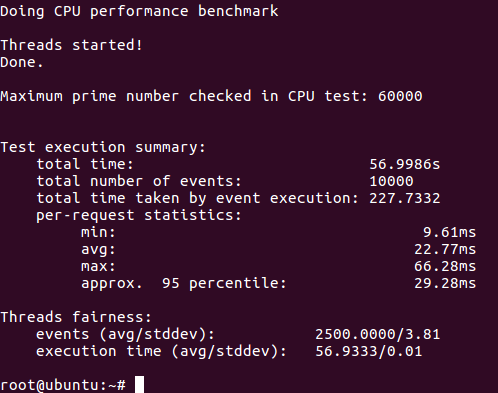
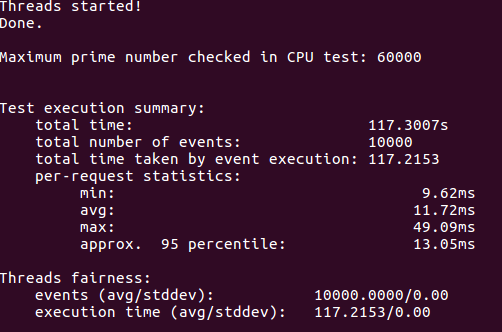
* line의 앞부분이 Domain-0이면 아무 기능 안한다.
* line의 앞부분이 VCPUs(sec):이면 line을 읽어온 후에 strtok()으로 tokenize 한다.
* 이후 vcpu의 개수를 계산하여 이를 CPU state 구조체 변수에 저장한다.
* Line의 앞부분이 위의 명시된 것이 아니라면 CPU state를 나타내므로 이를 CPU state에 저장한다.
* 가상머신 개수 만큼 필요한 값을 저장했다면 change\_xentop() 함수를 통해 DVAM 기능을 수행한다.



[change\_xentop 함수]

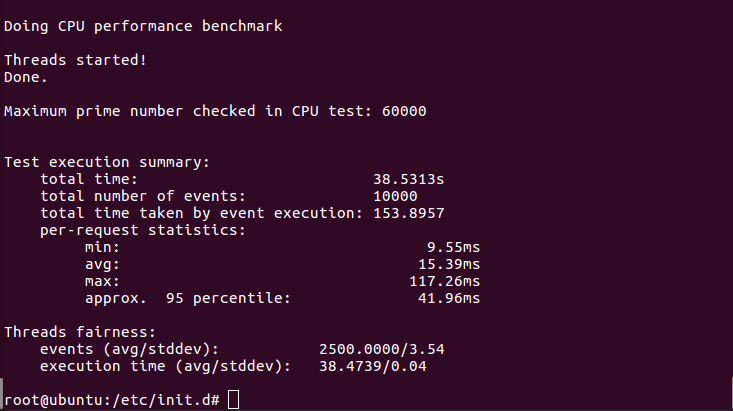
* 이 프로그램의 핵심 기능을 수행한다.
* xl vcpu-set “가상머신” “vcpu 개수”의 명령어를 만들어 system()함수로 terminal에서 실행한다.
* 이 때, CPU state가 -----r 또는 ------인 경우 CPU의 작동임을 알 수 있다.
* CPU가 작업을 수행하는 동안 CPU usage에 따라 vcpu의 수를 변경한다.
* 101%이하는 1개, 101~190%은 2개, 190~270%은 3개, 270%이상은 4개로 할당한다.
* 100 단위로 나눌 수도 있지만, 사용량 패턴 증가율이 점진적으로 줄어드는 경향을 보여 위와 같이 설정한다(101인 이유는 100으로 설정 시 vcpu 개수가 경계에서 자주 바뀌므로).
* CPU state가 -b----로 바뀌면 유휴상태 이므로 초기값인 vcpu 2개로 변동한다.

**시뮬레이션 결과**

****

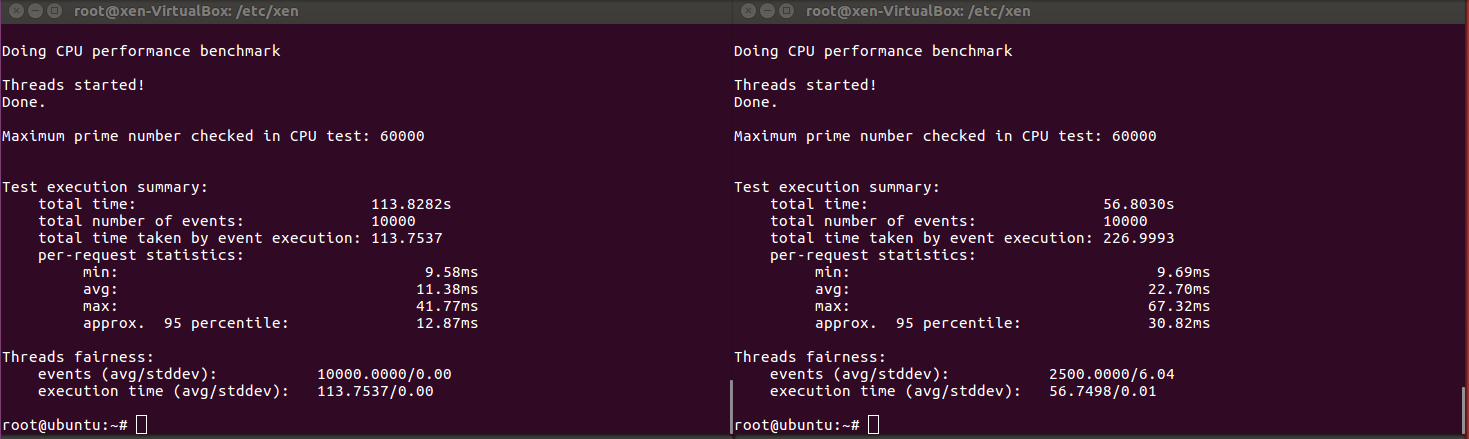
[DVAM 없이 pvm1만 실행] [DVAM 없이 pvm2만 실행]

* DVAM을 작동하지 않을 경우 각각 실행 시 2배 정도 pvm2가 빠른 것을 알 수 있다.
* Sysbench 시 쓰레드를 1개 사용하는 pvm1보다 4개 사용하는 pvm2의 병렬처리 작업 수행 때문이다.

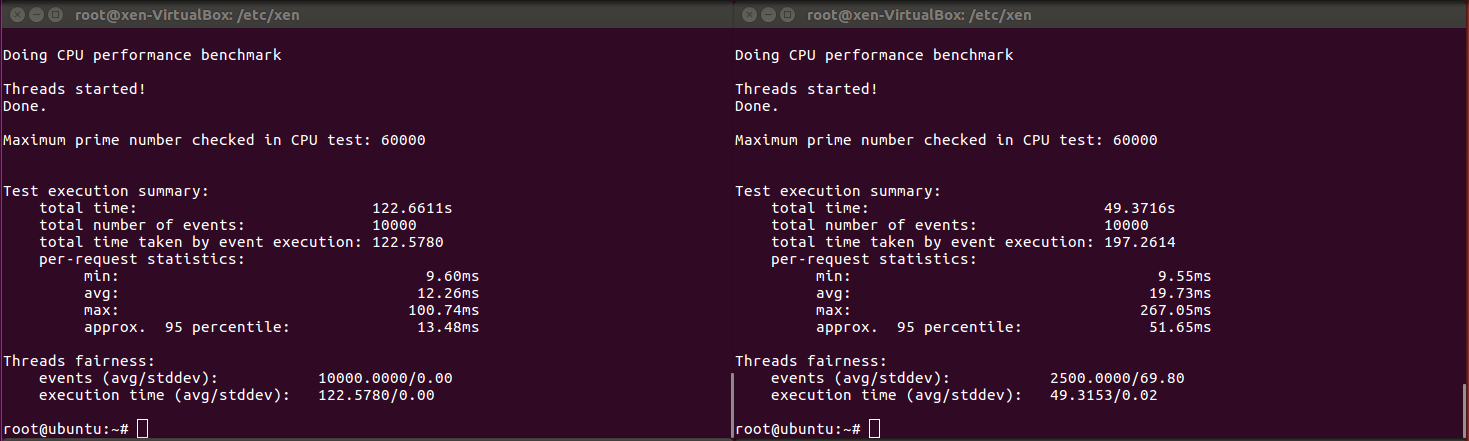


[DVAM 작동 후 pvm2만 실행]

* 가장 빠른 결과를 나타낸다.
* 이 경우, 270 이상의 CPU 사용률을 보이며 VCPU를 최대인 4개까지 할당한다.

****

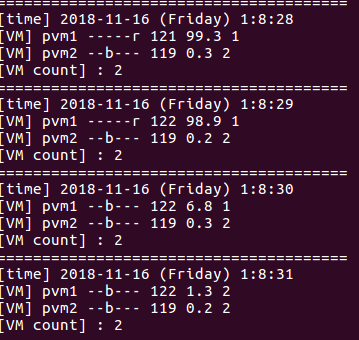
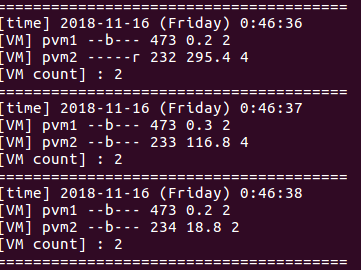
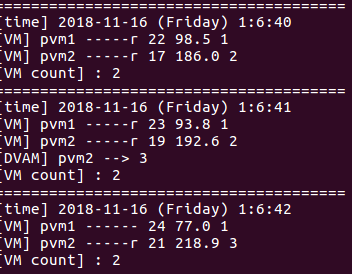
[DVAM 없이 pvm1, pvm2 동시 실행]

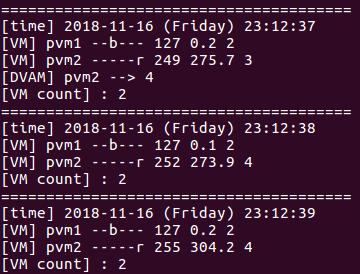


[DVAM 실행 후 pvm1, pvm2 동시 실행]

* Pvm1은 느려지고, vpm2는 빨라졌다.
* 이는 pvm1에 대해서는 100 이하의 사용률을 보여 VCPU를 1개 할당하고, vpm2에 대해서는 230 대의 사용률을 보여 VCPU를 3개 할당하게 된 결과이다.
* Pvm2의 처리 속도는 VCPU의 개수에 비례함을 알 수 있다.

**DVAM 출력 결과**



[DVAM에서 pvm2 VCPU 증가] [sysbench 종료 후 VCPU 복귀]

[VCPU 최대로 증가(pvm2만 DVAM 작동 후 sysbench run)]

* 가상머신은 name, state, running time, usage, vcpu 순으로 표시된다(sample code 활용).
* DVAM의 기능이 잘 작동하여 VCPU가 CPU usage에 따라 늘어나고, 작업 종료 후 복귀하는 것을 알 수 있다.

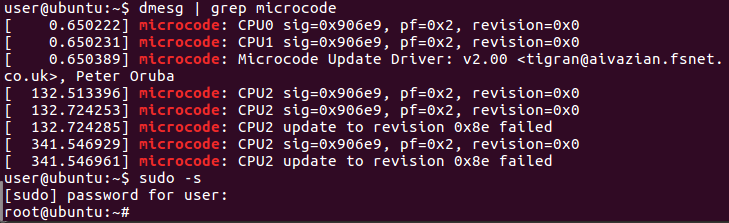
**맺음**

**DVAM을 만들면서 얻은 것…**

실습을 통해 xen의 가상환경 구축을 배웠고, 이를 응용한 프로젝트 수행을 하면서 xen의 원리와 사용에 대해 더 친숙해졌다. Domain OS에서 하위의 가상환경 VCPU를 조절하며 여러 가상머신이 작동할 경우, 효율적인 VCPU 분배를 통해 전체적인 퍼포먼스 향상을 가져올 수 있음을 확인하였다.

추가적으로, 리눅스 환경에서의 C Programming과 Terminal 사용에 대한 숙련도가 증가할 수 있는 시간이었다. 특히, popen(), system() 등의 함수를 통해 프로그램에서 Terminal 명령어 수행, Terminal 결과 읽기 등으로 프로그래밍을 하는 경험을 할 수 있었다.

**Microcode 문제…**



위와 같이 CPU의update to revision 0x8e failed 라는 메시지와 함께 VCPU의 변경이 안되는 문제가 있었다. 해당 문제는 가상머신을 작동한 상태에서 DVAM 프로그램을 오래 작동시키면 발생하였다. 이에 대해서는 여러 해결 방법을 적용해 보았으나 좋은 결과를 얻지는 못하였다. 이 부분이 아쉬움으로 남는다.

**부록**

**소스코드**

// gcc -o CCProject CCProject.c -lpthread

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

// Load xentop command line into file pointer

FILE \* fp = NULL;

char check\_name[20];

// CpuState

typedef struct \_CpuState{

char name[10];

char state[8];

int cpu\_time;

int vcpu\_num;

double cpu\_usage;

} CpuState;

void print\_time();

void\* open\_xentop(void\*);

void\* print\_xentop(void\*);

void dvam\_xentop();

int main(int argc, char \*argv[]){

printf("--PROGRAM START--\n");

pthread\_t p\_thread[2];

pthread\_create(&p\_thread[0], NULL, open\_xentop, NULL);

pthread\_create(&p\_thread[1], NULL, print\_xentop, NULL);

pthread\_join(p\_thread[0], NULL);

pthread\_join(p\_thread[1], NULL);

return 0;

}

// Print time

void print\_time(){

time\_t timer;

struct tm \*t;

char\* day\_kor[] = { "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday" };

timer = time(NULL);

t = localtime(&timer);

printf("=======================================\n");

printf("[time] %d-%d-%d (%s) %d:%d:%d\n", t->tm\_year + 1900, t->tm\_mon + 1, t->tm\_mday, day\_kor[t->tm\_wday], t->tm\_hour, t->tm\_min, t->tm\_sec);

}

// Change vcpu with DVAM

void change\_xentop(CpuState cpu\_state){

int new\_vcpu = 0;

char cmd[10240] = "xl vcpu-set ";

strcat(cmd, cpu\_state.name);

// Change the number of VCPUs using CPU usage

// 190% ~ 270% --> 3 VCPUs

if(strcmp(cpu\_state.state, "-----r") == 0 || strcmp(cpu\_state.state, "------") == 0){

if(cpu\_state.cpu\_usage <= 101.0 && cpu\_state.vcpu\_num != 1){

strcat(cmd, " 1");

new\_vcpu = 1;

system(cmd);

printf("[DVAM] %s --> %d\n", cpu\_state.name, new\_vcpu);

}

else if(cpu\_state.cpu\_usage <= 190.0 && cpu\_state.cpu\_usage > 101.0 && cpu\_state.vcpu\_num != 2){

strcat(cmd, " 2");

new\_vcpu = 2;

system(cmd);

printf("[DVAM] %s --> %d\n", cpu\_state.name, new\_vcpu);

}

else if(cpu\_state.cpu\_usage <= 270.0 && cpu\_state.cpu\_usage > 190.0 && cpu\_state.vcpu\_num != 3){

strcat(cmd, " 3");

new\_vcpu = 3;

system(cmd);

printf("[DVAM] %s --> %d\n", cpu\_state.name, new\_vcpu);

}

else if(cpu\_state.cpu\_usage > 270.0 && cpu\_state.vcpu\_num != 4){

strcat(cmd, " 4");

new\_vcpu = 4;

system(cmd);

printf("[DVAM] %s --> %d\n", cpu\_state.name, new\_vcpu);

}

}

else{

strcat(cmd, " 2");

//printf("%s\n", cmd);

system(cmd);

}

}

// Read xentop batch(using thread)

void\* open\_xentop(void\* n){

fp = popen("xentop -b -d 1 -v", "r");

while(1){

//Do nothing

}

pclose(fp);

}

// Print xentop info

void\* print\_xentop(void\* n){

// xentop info

char line[10240];

char line\_count[10240]; // for counting VMs

int init\_count = -1;

int vpm\_count = -1; // VM count

int vcpu\_count = -1; // VCPU count

int vcpu\_num = 0; // Current the number of VCPUs

FILE\* fp\_count = NULL;

char null\_string[100];

// string array for pvm

CpuState\* vpm\_list; // VM list

CpuState cpu\_state;

while(1){

while(fp == NULL){

//Do nothing

}

while(fgets(line, 10240, fp) != NULL){

//printf("%s", line);

memset(check\_name, 0, sizeof(check\_name));

sscanf(line, "%s", check\_name);

if(strcmp(check\_name, "NAME") == 0){

// Init counts

init\_count = -1;

vcpu\_count = -1;

// for counting the number of VMs

fp\_count = popen("xl vm-list", "r");

while(fgets(line\_count, 10240, fp\_count) != NULL){

init\_count++;

}

vpm\_list = malloc(init\_count \* sizeof(CpuState));

// Print time

print\_time();

vpm\_count = 0;

}

else if(strcmp(check\_name, "Domain-0") == 0){

// do nothing

}

else if(strcmp(check\_name, "VCPUs(sec):") == 0){

vcpu\_count++;

vcpu\_num = 0;

if(vcpu\_count > 0){

char\* ptr;

ptr = strtok(line, " ");

while(ptr = strtok(NULL, " ")){

//printf("%s\n", ptr);

vcpu\_num++;

}

vcpu\_num /= 2;

// Save current VCPUs

cpu\_state.vcpu\_num = vcpu\_num;

// Print CPU state

printf("[VM] %s %s %d %.1lf %d\n", cpu\_state.name, cpu\_state.state, cpu\_state.cpu\_time, cpu\_state.cpu\_usage, cpu\_state.vcpu\_num);

// vpm into vpm\_list

vpm\_list[vpm\_count] = cpu\_state;

vpm\_count++;

// All of VM's CPU state is listed

if(vpm\_count == init\_count){

for(int i = 0;i < init\_count;i++){

change\_xentop(vpm\_list[i]);

}

printf("[VM count] : %d\n", vpm\_count);

free(vpm\_list);

}

}

}

else{

// Set CPU state

sscanf(line, "%s %s %d %lf %s %s %s %s %s", cpu\_state.name, cpu\_state.state, &cpu\_state.cpu\_time, &cpu\_state.cpu\_usage, null\_string, null\_string, null\_string, null\_string, null\_string);

}

}

}

}