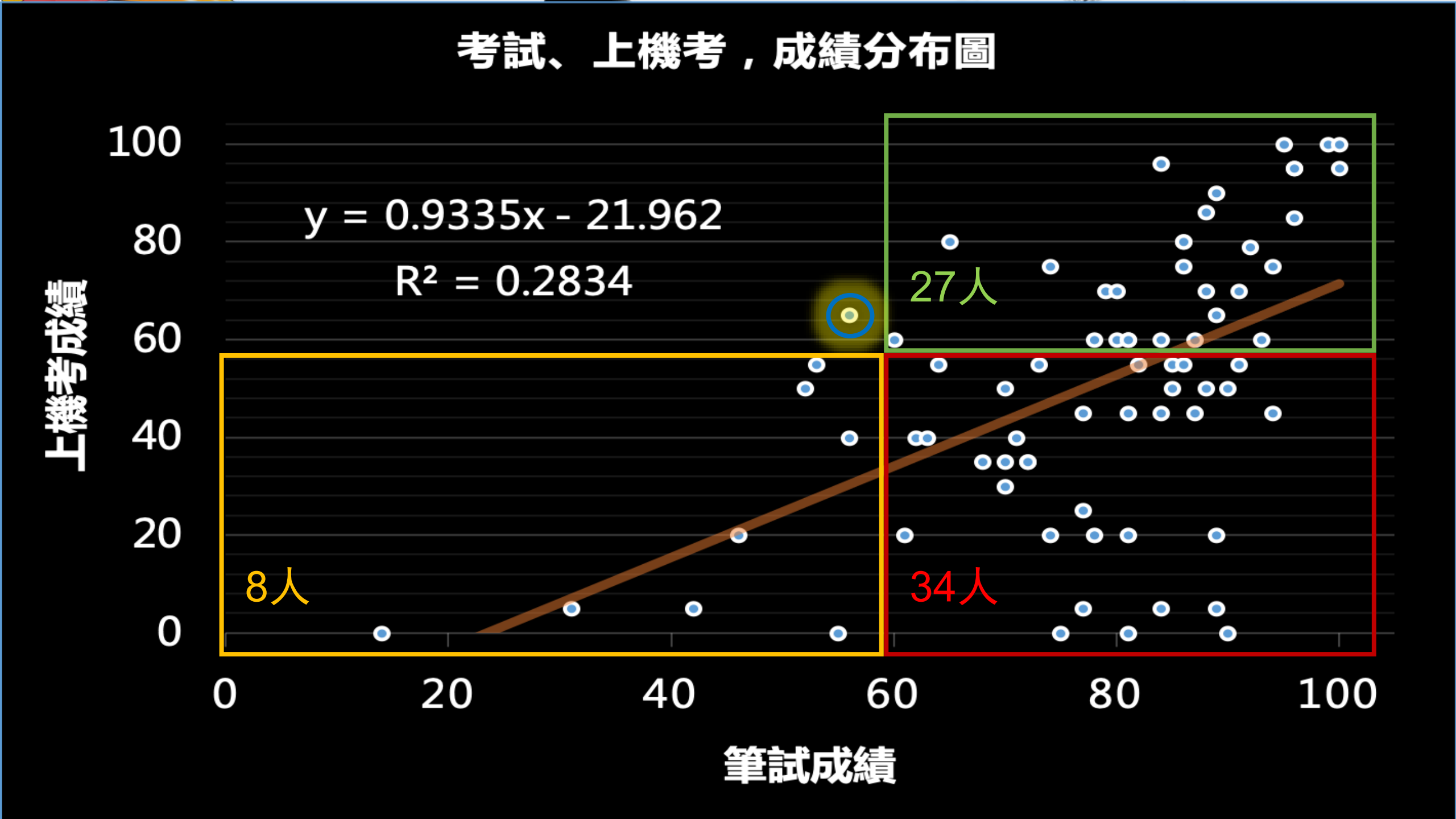
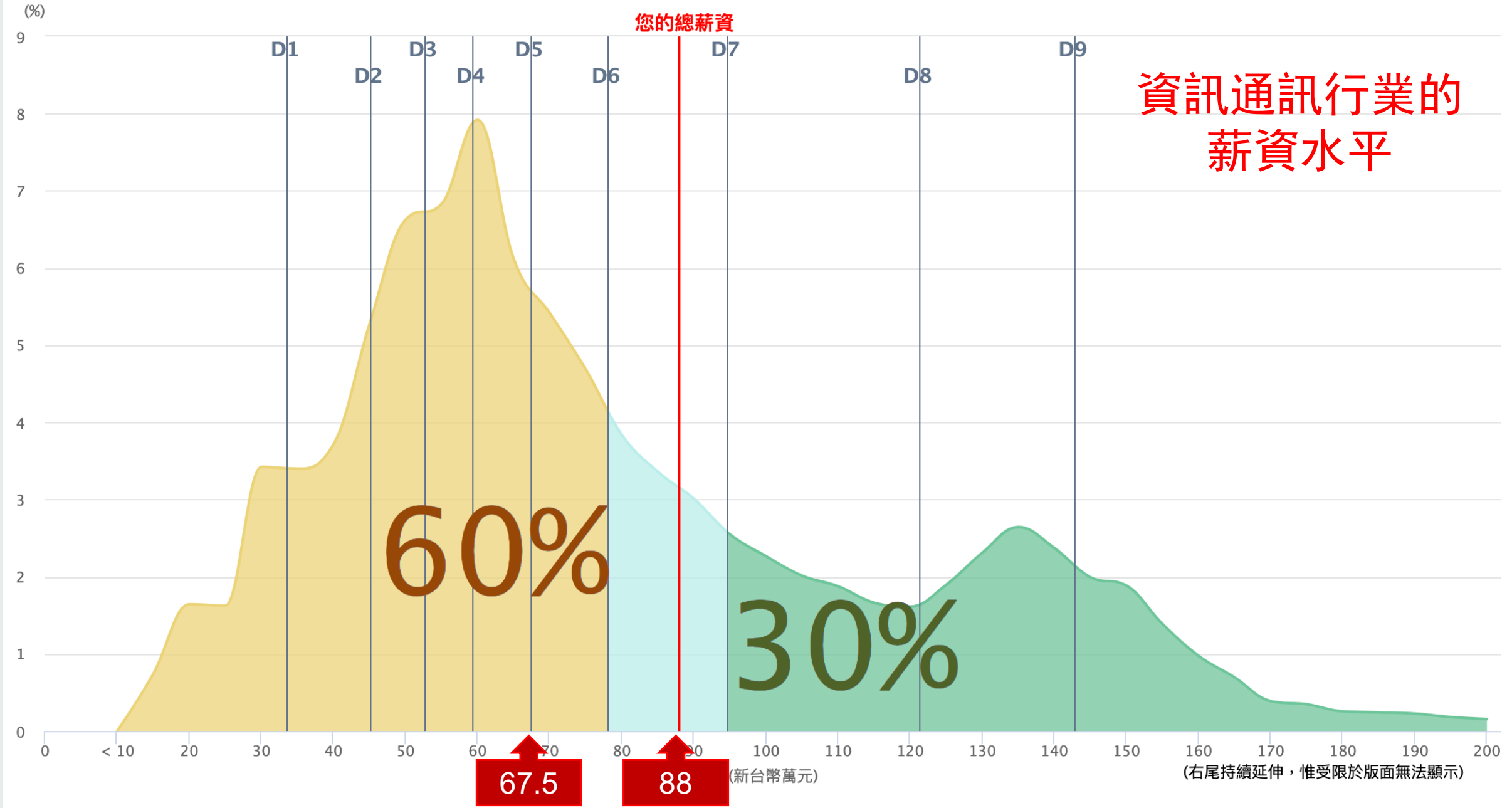
行程（process）

中正大學，作業系統實驗室
羅習五 陽春副教授
shiwulo@gmail.com


考試、上機考，成績分布圖
y = 0.9335x - 21.962
R² = 0.2834
27人
8人
34人
上機考成績
筆試成績
100
80
60
40
20
0
0
20
40
60
80
100

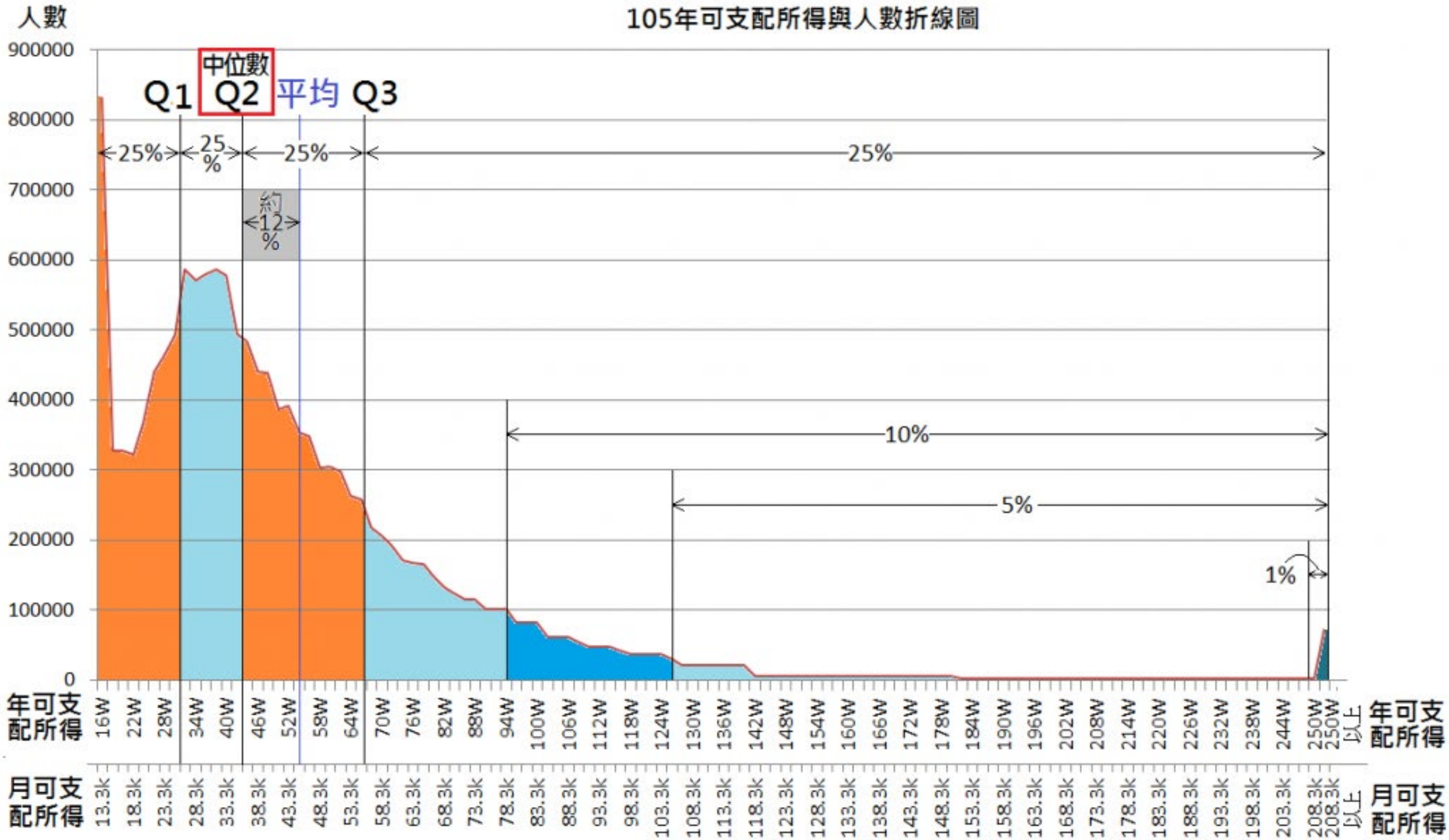
台北市「每人所得」

- 全台北市平均「 880 960 」
- 內湖區平均「 830 262 」

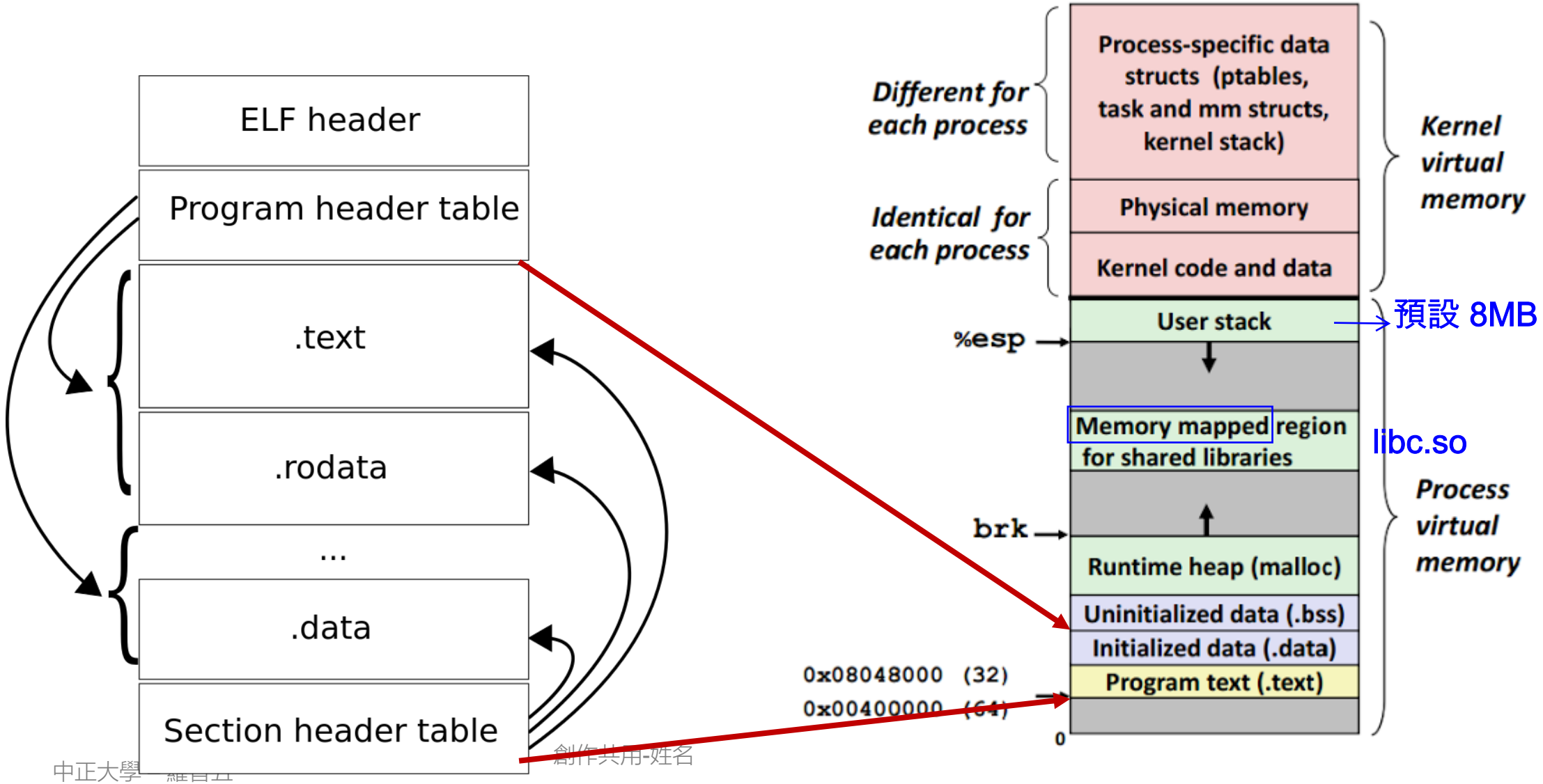


D1：第1十分位數 33.6 萬元 **D2**：第2十分位數 45.2 萬元 **D3**：第3十分位數 52.8 萬元 **D4**：第4十分位數 59.4 萬元 **D5**：第5十分位數(中位數) 67.5 萬元

D6：第6十分位數 78.3 萬元 **D7**：第7十分位數 94.7 萬元 **D8**：第8十分位數 121.4 萬元 **D9**：第9十分位數 142.9 萬元


105年可支配所得與人數折線圖
人數
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
Q1
中位數
Q2
平均
Q3
25%
25%
25%
約12%
25%
10%
5%
1%
年可支配所得
16W
22W
28W
34W
40W
46W
52W
58W
64W
70W
76W
82W
88W
94W
100W
106W
112W
118W
124W
130W
136W
142W
148W
154W
160W
166W
172W
178W
184W
190W
196W
202W
208W
214W
220W
226W
232W
238W
244W
250W
250W以上
年可支配所得
月可支配所得
13.3k
18.3k
23.3k
28.3k
33.3k
38.3k
43.3k
48.3k
53.3k
58.3k
63.3k
68.3k
73.3k
78.3k
83.3k
88.3k
93.3k
98.3k
103.3k
108.3k
113.3k
118.3k
123.3k
128.3k
133.3k
138.3k
143.3k
148.3k
153.3k
158.3k
163.3k
168.3k
173.3k
178.3k
183.3k
188.3k
193.3k
198.3k
203.3k
208.3k
208.3k以上
月可支配所得

什麼是行程



什麼是行程

沒有做真正的複製，有需要讀到該資料時才複製過去

- 作業系統先用一對一的「對應」將一個ELF（Extensible Linking Format，一種執行檔格式）映射到記憶體中
- 再由作業系統依照實際的需求，擴增data session（透過brk或mmap等系統呼叫）及stack（自動長大，預設值最大為8MB）
- 動態連結庫（shared object, so）則由作業系統依照當時是否已經把shared object（so）已經載入到記憶體，決定是否要載入該so或只要將該so映射到這個行程的記憶體空間
- so通常位於stack與heap之間

Kernel 也有 KASLR

早期所有的執行檔都有一個固定的開始位置

Linux和多數的OS都使用ASLR（Address space layout randomization），隨機配置so和data section的位置，因此程式執行，每次的memory layout會有所不同

Stack size

```
$ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 128067
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time                (seconds, -t) unlimited
max user processes              (-u) 128067
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

程式的參數

- 啟動行程時，可以帶上參數，這些參數如下一頁投影片所示

echo.c 程式的參數

一個標準的GNU C的主程式應該下底下這樣

```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

/*argc，代表在呼叫這個執行檔的時候，總共有幾個參數*/
/*請注意，第一個參數一定是執行檔的檔名，這樣的設計，有助於除錯*/
/*argv則是字串陣列，最後一個字串是NULL*/

int main(int argc, char**argv) {
    int i=0;

    while(argv[i]!=NULL) {
        printf("%s\n", argv[i]);
        i++;

    }

    /*依照程式的屬性，設定適當的return value*/
    return 1;
}
```

argv

執行結果

```
$ ./echo para1 para2 para3
./echo
para1
para2
para3
```

程式的執行環境：環境變數

在command模式的設定法

myname=shiwulo

- 注意，等號的二邊不可以加上空白字元
- 變數的開頭不可以是數字，變數名稱不可以是英數以外的東西，例如：@mail是不可以的
- 變數的內容如果有空白，可以用單引號「'」或雙引號「"」包起來
- 可以使用跳脫字元「\」將特殊的字元（空白、@等）加入變數
- 如果要附加（append）到一個變數，可以用下列形式：
  - 例：PATH="PATH":~~myname=shiwulo~~ **這裡應該要是 path**
- 取消用unset，例如：unset myname
- 如果要將該變數傳給子行程，要加上export，例如：export myname

課堂作業

- 將你的英文名字export成環境變數
- 將「./」加到PATH內
  - 試試看，如果執行目前目錄的執行檔，還需要加上「./myexe」或者「myexe」就好
  - 這可能會造成安全性的漏洞

範例

```
shiwulo@vm:~$ shiwu=shiwulo
shiwulo@vm:~$ echo $shiwu
shiwulo
shiwulo@vm:~$ bash
shiwulo@vm:~$ echo $shiwu /*子行程看不到環境變數shiwu*/

shiwulo@vm:~$ export shiwu
shiwulo@vm:~$ bash
shiwulo@vm:~$ echo $shiwu /*子行程看得到環境變數shiwu*/
shiwulo
```

user environment相關的函數及變數

- #include <stdlib.h>

- char *getenv(const char *name);
- int putenv(char *string);
- extern char **environ;

listEnv

```
1.  #include <stdlib.h>
2.  #include <stdio.h>
3.  extern char **environ;

4.  int main(int argc, char**argv) {
5.      int i;
6.      while(environ[i] != NULL) {
7.          printf("%s\n", environ[i++]);
8.      }
9.  }
```

執行結果

```
$listEnv
XDG_VTNR=7
XDG_SESSION_ID=c2
…
…
…
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
GTK_IM_MODULE=ibus
LESSCLOSE=/usr/bin/lesspipe %s %s
TEXTDOMAINDIR=/usr/share/locale/
COLORTERM=gnome-terminal
XAUTHORITY=/home/shiwulo/.Xauthority
_=./listEnv
```

getenv

```
1.  #include <stdio.h>
2.  #include <stdlib.h>

3.  int main(int argc, char **argv) {
4.      int i;
5.      char* value;
6.      for (i=1; argv[i]!=NULL; i++) {
7.          value = getenv(argv[i]);
8.          printf("%s=%s\n", argv[i], value);
9.      }
10.
11.     return 0;
12. }
```

執行結果

沒有 ./

```
./getEnv PATH
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/games:/usr/local/games
```

程式碼的結束呼叫

程式的結束

- 程式結束時，作業系統會回收幾乎所有的資源，並且讓這個程式進入殭屍模式（zombie，下一個章節會介紹）。
- 我們可以「註冊」一些函數，讓程式碼「正確的結束時」，將執行這些函數。
  - 例如：釋放shared memory
  - 例如：釋放網路的socket

atexit

- #include <stdlib.h>
- int atexit(void (*function)(void)); **可跨平台使用，但是沒有帶參數**
- int on_exit(void (*function)(int , void *), void *arg); **不可跨平台使用，但是可以帶參數**

  通常 libc 會幫我們呼叫
- atexit 在程式正常結束時，作業系統（OR compiler）會幫我們呼叫函數：function
- on_exit在程式正常結束時，將呼叫所註冊的函數，該函數的第一個參數是「該程式的回傳值」，第二個變數則是一個指標（使用者可以自行應用）
- on_exit目前只能在Linux上使用

atexit()

```
1.  #include <stdio.h>
2.  #include <stdlib.h>


3.  void myName() {
4.      printf("shiwulo\n");
5.  }

6.  int main(int argc, char **argv) {
7.      atexit(myName);
8.      return 0;
9.  }
```

執行結果

```
$ ./atexit
shiwulo
```

on_exit()

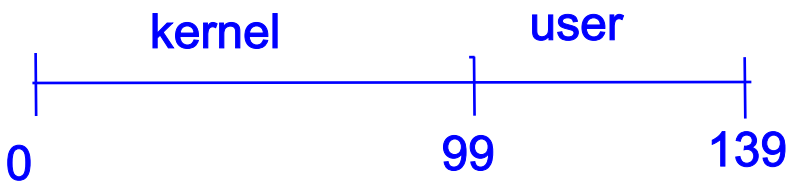
```
1.  #include <stdio.h>
2.  #include <stdlib.h>

3.  void myName(int ret, void *arg) {
4.          printf("%s shiwulo\n", (char*)arg);
5.  }
6.

7.  int main(int argc, char **argv) {
8.          char* p = "professor";
9.          on_exit(myName, p);
10.         return 0;
11. }
```

執行結果

```
shiwulo@vm:~/sp/ch08$ gcc on_exit.c
shiwulo@vm:~/sp/ch08$ ./a.out
professor shiwulo
```

控制行程的
優先權及多核心排程

改變行程的優先權



- 對大部分的系統而言（包含Linux）優先權越小，就能拿到更高的優先權（或者是提升優先權）
  - 例如：（絕對）優先權0是系統中最高的優先權
  - 例如：（絕對）優先權139是系統最低的優先權
- 通常只有root可以提升優先權，其他使用者只能降低優先權
- 在Linux中，撰寫程式時不可以假設底層的優先權處理機制
  - 例如：為了避免競賽問題（race condition），假設高優先權的工作做完以後，才執行低優先權的工作。

變更優先權

- Linux指令
  - nice

- 函數呼叫
- #include <unistd.h>
- int nice(int inc);　　/*回傳值為先的優先權，只有超級使用者可以設定inc為負數*/

myNice.c

```
1.    int main(int argc, char **argv) {
2.       int niceVal, newNiceVal, i, ret=0;
3.       sscanf(argv[1], "%d", &niceVal);
4.       errno =0;
5.       newNiceVal = nice(niceVal);
6.       if (newNiceVal == -1 && errno !=0)
7.          perror("Error: nice");
8.       else {
9.          printf("new val = %d\n",
10.            newNiceVal);
11.      }
```

```
1.       for (int i=0; i<=500000000; i++) {
2.          if (i%50000000 == 0)
3.          fprintf(stderr, "*");
4.          ret+=i;
5.       }
6.       return ret;
7.    }
```

執行結果

Linux 的 scheduling algorithm 是 CFS，每提高一級，速度可提升 1.25 倍，這個指的是 CPU 的部分，不包含 I/O

```
./myNice 10
new val = 10
shiwulo@ubuntu:~/Desktop/sp/ch9$ ./myNice -10
nice: Operation not permitted
shiwulo@ubuntu:~/Desktop/sp/ch9$ sudo ./myNice -10
new val = -10
```

設定task可以在哪些CPU上執行

相關的巨集指令

```
1.  #define _GNU_SOURCE
2.  #include <sched.h>

3.  void CPU_ZERO(cpu_set_t *set);
4.  void CPU_SET(int cpu, cpu_set_t *set);
5.  void CPU_CLR(int cpu, cpu_set_t *set);
6.  int  CPU_ISSET(int cpu, cpu_set_t *set);
7.  int  CPU_COUNT(cpu_set_t *set);
```

初始化

設定哪些 CPU 可以跑

設定哪些 CPU 不可以跑

這裡會跟後面要介紹的 pthread 有點重複

cgroup 可以設定得更精細

相關巨集的意義

- CPU_ZERO()　　Clears set, so that it contains no CPUs.
- CPU_SET()　　Add CPU cpu to set.
- CPU_CLR()　　Remove CPU cpu from set.
- CPU_ISSET()　　Test to see if CPU cpu is a member of set.
- CPU_COUNT()　　Return the number of CPUs in set.

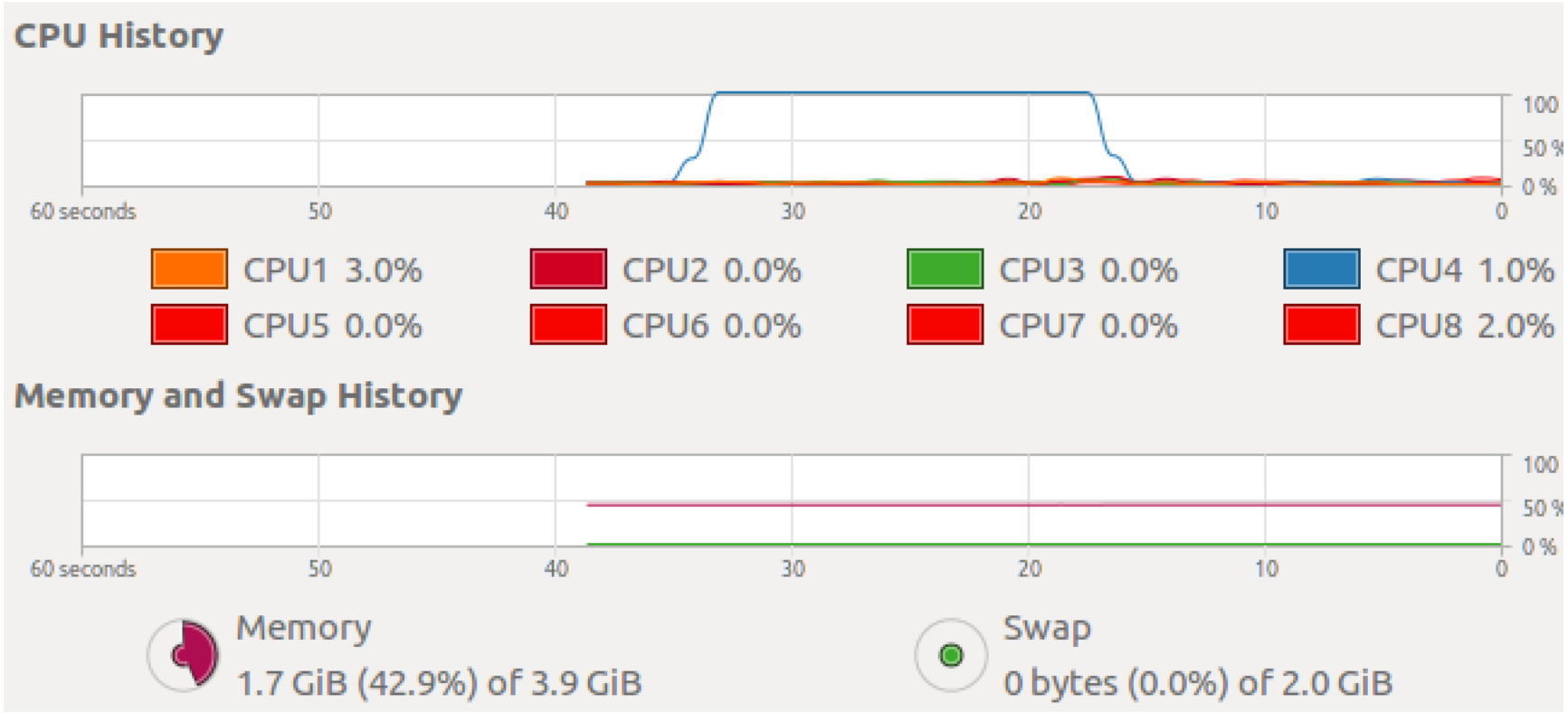
相關的函數

```
1.  #define _GNU_SOURCE
2.  #include <sched.h>

3.  int sched_setaffinity(pid_t pid, size_t cpusetsize,
4.                        cpu_set_t *mask);
5.  /*如果pid為0，表示設定目前的process*/

6.  int sched_getaffinity(pid_t pid, size_t cpusetsize,
7.                        cpu_set_t *mask);
8.  /*如果pid為0，表示設定目前的process*/
```

cpu_set

```
1.  int main(int argc, char **argv) {
2.      cpu_set_t set;
3.      CPU_ZERO(&set);
4.      int i,j,ret;
5.      CPU_ZERO(&set);
6.      CPU_SET(3, &set);
7.      ret=sched_setaffinity(0, sizeof(cpu_set_t), &set);
8.      if (ret==-1)
9.          perror("sched_setaffinity");
10.     for(i=0; i<1000; i++) { /*busy loop*/
11.         for(j=0; j<10000000; j++) {
12.             j=j+1;
13.         }
14.     }
15.     return j;
16. }
```

執行結果（system monitor）



小結

- 瞭解什麼是環境變數，如何獲得環境變數
- 瞭解如何設定優先權，設定優先權的原則
- 設定程式「正常」結束執行時，「自動」執行的函數
- 在多核心、多處理器環境中，如何設定程式在哪一顆處理器上執行