操作系统课程设计实验报告

实验名称	内存监视		
学号	1120141831	姓名	朴泉宇

一、实验目的

在 Windows 以及 Linux 平台下对系统信息、执行信息、内存信息以及对单个进程的监视。

二、实验内容

1. 在 Windows 下编写监视器代码
2. 在 Windows 下编译运行，并监视各项信息
3. 在 Linux 下直接调用系统命令，监视各项信息

三、实验环境及配置方法

1. 利用 Visual Studio 编写 Windows 下的代码，并编译运行
2. 在 Ubuntu 中用 Terminal 调用系统命令

四、实验方法和实验步骤（程序设计与实现）

1. 编写 Windows 下的监视器代码

(1) 思路

此程序整体分成四个模块：

```
1.  void ShowSysInfo();
2.  void ShowPerformanceInfo();
3.  void ShowPhysicalMemoryInfo();
4.  void ShowTHISProcInfo();
```

以及其中 void ShowTHISProcInfo();模块调用的以下两个子模块：

```
1.  void ShowVirtualMemInfo(HANDLE hProcess);
2.  void ShowWorkingSetInfo(int PID);
```

以及为实时覆盖输出的有关控制台光标位置的函数：

```
1.  void GetConsoleCursorXY(int &x, int &y);
2.  void SetConsoleCursorTo(int x, int y);
```

1) `void ShowSysInfo();`模块：

创建 `SYSTEM_INFO` 结构体变量，并调用 `GetSystemInfo();`来实时获取操作系统的各项信息。

2) `void ShowPerformanceInfo();`模块：

创建 `PERFORMANCE_INFORMATION` 结构体变量，并调用 `GetPerformanceInfo();`来实时获取操作系统的执行信息。

3) `void ShowPhysicalMemoryInfo();`模块：

创建 `MEMORYSTATUSEX` 结构体变量，并调用 `GlobalMemoryStatusEx();` 来实时获取当前物理内存的使用情况。

4) `void ShowTHISProcInfo();`模块：

通过 Windows 自带的任务管理器获取某进程的 PID，通过 PID 获得此进程的 HANDLE。

`void ShowVirtualMemInfo();` 子模块创建 `SYSTEM_INFO` 和 `MEMORY_BASIC_INFORMATION` 结构体变量，并调用 `GetSystemInfo();`和 `VirtualQueryEx();`来实时获取次进程的虚拟内存使用情况。

`void ShowWorkingSetInfo();` 子模块创建 `PROCESSENTRY32` 和 `PROCESS_MEMORY_COUNTERS` 结构体变量，并调用 `Process32First();`和 `GetProcessMemoryInfo();`来实时获取此进程的工作集情况。

5) `void GetConsoleCursorXY();`函数：

获取当前控制台的光标位置。与 `kbhit()`配合，完成时事刷新信息的功能。

6) `void SetConsoleCursorTo();`函数：

将控制台的光标设定在指定位置。与 `kbhit()`配合，完成时事刷新信息的功能。

(2) 函数解释

1) `void ShowSysInfo();`模块：

```
1.  /*Show System Info*/
2.  void ShowSysInfo()
3.  {
4.      SYSTEM_INFO SystemInfo;
```

```
5.      int X = 0, Y = 0, flag = 0; //Cursor Position
6.
7.      ZeroMemory(&SystemInfo, sizeof(SystemInfo));
8.
9.      while (kbhit() == 0)
10.     {
11.         if (flag == 0)
12.         {
13.             GetConsoleCursorXY(X, Y);
14.             flag = 1;
15.         }
16.         else
17.         {
18.             SetConsoleCursorTo(X, Y);
19.         }
20.
21.         GetSystemInfo(&SystemInfo);
22.
23.         printf("Hardware information:\n");
24.         printf("OEM ID: %u\n", SystemInfo.dwOemId); //1. OEM ID
25.         printf("Processor Architecture: "); //2. Processor Architecture
26.         switch (SystemInfo.wProcessorArchitecture)
27.         {
28.         case 0:printf("x86\n");break;
29.         case 5:printf("ARM\n");break;
30.         case 6:printf("Intel Itanium-based\n");break;
31.         case 9:printf("x64 (AMD or Intel)\n");break;
32.         case 0xffff:printf("Unknown architecture.\n");break;
33.         }
34.         printf("Page Size(Bytes): %u\n", SystemInfo.dwPageSize);    //3. Page Size

35.         printf("Minimum Application Address: 0x%lx\n", SystemInfo.lpMinimumApplica
    tionAddress); //4. Minimum Application Address
36.         printf("Maximum Application Address: 0x%lx\n", SystemInfo.lpMaximumApplica
    tionAddress); //5. Maximum Application Address
37.         printf("Size of Application's usable virtual memory(Bytes): %u\n", ((DWORD
    )SystemInfo.lpMaximumApplicationAddress - (DWORD)SystemInfo.lpMinimumApplicationAd
    dress));    //Extend. Size of Application's usable virtual memory
38.         printf("Active Processor Mask: %u\n", SystemInfo.dwActiveProcessorMask);
      //6. Active Processor Mask
39.         printf("Number of Processors: %u\n", SystemInfo.dwNumberOfProcessors);  //
    7. Number of Processors
40.         printf("Processor Type: %u\n", SystemInfo.dwProcessorType); //8. Processor
     Type
```

```
        printf("Allocation Granularity: %u\n", SystemInfo.dwAllocationGranularity); //9. Allocation Granularity
        printf("Processor Level: %u\n", SystemInfo.wProcessorLevel);    //10. Processor Level
        printf("Processor Revision: %u\n", SystemInfo.wProcessorRevision);  //11. "Processor Revision

        printf("\n*****Press <ESC> to stop.*****\n");

        Sleep(1000);    //Refresh per second.
    }
}
```

2) void ShowPerformanceInfo();模块:

```
/*Show Performance Info*/
void ShowPerformanceInfo()
{
    PERFORMANCE_INFORMATION PerformanceInfo;
    int X = 0, Y = 0, flag = 0; //Cursor Position

    ZeroMemory(&PerformanceInfo, sizeof(PerformanceInfo));
    PerformanceInfo.cb = sizeof(PerformanceInfo);

    while (kbhit() == 0)
    {
        if (flag == 0)
        {
            GetConsoleCursorXY(X, Y);
            flag = 1;
        }
        else
        {
            SetConsoleCursorTo(X, Y);
        }

        GetPerformanceInfo(&PerformanceInfo, PerformanceInfo.cb);

        printf("CommitTotal(pages): %u\n", PerformanceInfo.CommitTotal);    //1. The number of pages currently committed by the system.
        printf("CommitLimit(pages): %u\n", PerformanceInfo.CommitLimit);    //2. The current maximum number of pages that can be committed by the system without extending the paging file(s).
        printf("CommitPeak(pages): %u\n", PerformanceInfo.CommitPeak);  //3. The m
```

```
    aximum number of pages that were simultaneously in the committed state since the l
    ast system reboot.
27.         printf("PhysicalTotal(pages): %u\n", PerformanceInfo.PhysicalTotal);    //
    4. The amount of actual physical memory, in pages.
28.         printf("PhysicalAvailable(pages): %u\n", PerformanceInfo.PhysicalAvailable
    );    //5. The amount of physical memory currently available, in pages.
29.         printf("SystemCache(pages): %u\n", PerformanceInfo.SystemCache);    //6. T
    he amount of system cache memory, in pages.
30.         printf("KernelTotal(pages): %u\n", PerformanceInfo.KernelTotal);    //7. T
    he sum of the memory currently in the paged and nonpaged kernel pools, in pages.
31.         printf("KernelPaged(pages): %u\n", PerformanceInfo.KernelPaged);    //8. T
    he memory currently in the paged kernel pool, in pages.
32.         printf("KernelNonpaged(pages): %u\n", PerformanceInfo.KernelNonpaged);  //
    9. The memory currently in the nonpaged kernel pool, in pages.
33.         printf("PageSize(Bytes): %u\n", PerformanceInfo.PageSize);  //10. The size
     of a page, in bytes.
34.         printf("HandleCount: %u\n", PerformanceInfo.HandleCount);   //11. The curr
    ent number of open handles.
35.         printf("ProcessCount: %u\n", PerformanceInfo.ProcessCount); //12. The curr
    ent number of processes.
36.         printf("ThreadCount: %u\n", PerformanceInfo.ThreadCount);   //13. The curr
    ent number of threads.
37.
38.         printf("\n*****Press <ESC> to stop.*****\n");
39.
40.         Sleep(1000);    //Refresh per second.
41.     }
42. }
```

3) `void ShowPhysicalMemoryInfo();`模块：

```
1.  /*Show Physical Memory Info*/
2.  void ShowPhysicalMemoryInfo()
3.  {
4.      MEMORYSTATUSEX MemoryStatusEX;
5.      int X = 0, Y = 0, flag = 0; //Cursor Position
6.
7.      ZeroMemory(&MemoryStatusEX, sizeof(MemoryStatusEX));
8.      MemoryStatusEX.dwLength = sizeof(MemoryStatusEX);
9.
10.     while (kbhit() == 0)
11.     {
12.         if (flag == 0)
13.         {
```

```
            GetConsoleCursorXY(X, Y);
            flag = 1;
        }
        else
        {
            SetConsoleCursorTo(X, Y);
        }

        GlobalMemoryStatusEx(&MemoryStatusEX);

        printf("There is %ld percent of memory in use.\n", MemoryStatusEX.dwMemoryLoad);
        printf("There are %I64d total KB of physical memory.\n", MemoryStatusEX.ullTotalPhys / 1024);
        printf("There are %I64d free KB of physical memory.\n", MemoryStatusEX.ullAvailPhys / 1024);
        printf("There are %I64d total KB of paging file.\n", MemoryStatusEX.ullTotalPageFile / 1024);
        printf("There are %I64d free KB of paging file.\n", MemoryStatusEX.ullAvailPageFile / 1024);
        printf("There are %I64d total KB of virtual memory.\n", MemoryStatusEX.ullTotalVirtual / 1024);
        printf("There are %I64d free KB of virtual memory.\n", MemoryStatusEX.ullAvailVirtual / 1024);

        // Show the amount of extended memory available.
        printf("There are %I64d free KB of extended memory.\n", MemoryStatusEX.ullAvailExtendedVirtual / 1024);

        printf("\n*****Press <ESC> to stop.*****\n");

        Sleep(1000);    //Refresh per second.
    }
}
```

4) `void ShowTHISProcInfo();`模块：

```
/*Show THIS Process Info, includes Virtual Memory Info and Working Set Info*/
void ShowTHISProcInfo()
{
    int THISPID = MonitorPID;
    HANDLE hTHISProcess;
    int X = 0, Y = 0, flag = 0; //Cursor Position

```

```
    //THISPID = GetCurrentProcessId();
    hTHISProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, THISPID);

    while (kbhit() == 0)
    {
        if (hTHISProcess != NULL)
        {
            if (flag == 0)
            {
                GetConsoleCursorXY(X, Y);
                flag = 1;
            }
            else
            {
                SetConsoleCursorTo(X, Y);
            }

            ShowVirtualMemInfo(hTHISProcess);
            ShowWorkingSetInfo(THISPID);

            printf("\n*****Press <ESC> to stop.*****\n");
            printf("*****Close the monitor first, then close The Process.*****\n")
;

            Sleep(1000);    //Refresh per second.
        }
        else
        {
            printf("The Process was not exist. The monitor will close.\n");

            return;
        }
    }
}
```

5) `void ShowVirtualMemInfo();`子模块：

```
/*Show THE PROCESS's Virtual Memory Info*/
void ShowVirtualMemInfo(HANDLE hProcess)
{
    SYSTEM_INFO SI;
    MEMORY_BASIC_INFORMATION MBI;

    ZeroMemory(&SI, sizeof(SI));
```

```
    ZeroMemory(&MBI, sizeof(MBI));

    GetSystemInfo(&SI);
    LPCVOID pBlock = (LPVOID)SI.lpMinimumApplicationAddress;

    while (pBlock < SI.lpMaximumApplicationAddress)
    {
        if (VirtualQueryEx(hProcess, pBlock, &MBI, sizeof(MBI)) == sizeof(MBI))
        {
            LPCVOID pEnd = (PBYTE)pBlock + MBI.RegionSize;
            printf("Block starts at: 0x%lx, ", pBlock); //1. Show Block Starts Add
r.
            printf("Block Length(Bytes): %u, ", MBI.RegionSize);    //2. Show Bloc
k's length.
            switch (MBI.State)  //3. Show Block's State.
            {
            case MEM_COMMIT:
                printf("Committed, ");break;
            case MEM_FREE:
                printf("Free.\n");break;
            case MEM_RESERVE:
                printf("Reserved, ");break;
            }
            switch (MBI.Type)   //4. Show Block's Type.
            {
            case MEM_IMAGE:
                printf("Image.\n");break;
            case MEM_MAPPED:
                printf("Mapped.\n");break;
            case MEM_PRIVATE:
                printf("Private.\n");break;
            }

            pBlock = pEnd;
        }
        else
        {
            break;
        }
    }
}
```

6) `void ShowWorkingSetInfo();`子模块：

```
/*Show THE PROCESS's Working Set Info*/
void ShowWorkingSetInfo(int PID)
{
    PROCESSENTRY32 PE32;
    PROCESS_MEMORY_COUNTERS ProcMemCounter;
    HANDLE hProcessSnap;

    ZeroMemory(&PE32, sizeof(PE32));
    ZeroMemory(&ProcMemCounter, sizeof(ProcMemCounter));
    PE32.dwSize = sizeof(PE32);
    ProcMemCounter.cb = sizeof(ProcMemCounter);

    hProcessSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

    BOOL bNext = Process32First(hProcessSnap, &PE32);
    printf("Process Working Set Information:\n");
    while (bNext)
    {
        if (PID == PE32.th32ProcessID)
        {
            HANDLE hProcess;
            hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, PID);
            GetProcessMemoryInfo(hProcess, &ProcMemCounter, sizeof(ProcMemCounter)
);
            printf("Commited(Bytes): %u\n", ProcMemCounter.PagefileUsage);
            printf("Working Set Size(Byte): %u\n", ProcMemCounter.WorkingSetSize);

            printf("Peak Working Set Size(Byte): %u\n", ProcMemCounter.PeakWorking
SetSize);

            break;
        }

        bNext = Process32Next(hProcessSnap, &PE32);
    }
}
```

7） `void GetConsoleCursorXY();`函数:

```
/*Get console cursor's position*/
void GetConsoleCursorXY(int &x, int &y)
{
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
```

```
    CONSOLE_SCREEN_BUFFER_INFO CSBI;

    GetConsoleScreenBufferInfo(hConsole, &CSBI);

    x = CSBI.dwCursorPosition.X;
    y = CSBI.dwCursorPosition.Y;
}
```

8) `void SetConsoleCursorTo();`函数:

```
/*Set console cursor to (x, y)*/
void SetConsoleCursorTo(int x, int y)
{
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD SetCursorPosition = { x, y };

    SetConsoleCursorPosition(hConsole, SetCursorPosition);
}
```

(3) main 函数

main 函数起到引导用户选择并使用 Monitor 的作用。

```
int main()
{
Restart:
    printf("Please Input the PID to monitor a process.\n");
    printf("If don't want to monitor any process, please input -1.\n");
    scanf("%d", &MonitorPID);

    if (MonitorPID == -1)
    {
        int SelectNum;

        printf("1. Show System Information.\n");
        printf("2. Show Performance Infomation.\n");
        printf("3. Show Physical Memory Information.\n");
        printf("Please input the number to select.\n");

        scanf("%d", &SelectNum);
        switch (SelectNum)
        {
        case 1:ShowSysInfo();break;
        case 2:ShowPerformanceInfo();break;
```

```
22.         case 3:ShowPhysicalMemoryInfo();break;
23.         default:printf("Input Error!\n");break;
24.         }
25.
26.         printf("\nPlease input the command:\n");
27.         printf("[0] Restart the monitor.\n");
28.         printf("Other key to close the monitor.\n");
29.         int command = -1;
30.         scanf("%d", &command);
31.         if (command == 0)
32.         {
33.             goto Restart;
34.         }
35.         else
36.         {
37.             printf("Thank you for using.\n");
38.         }
39.     }
40.     else
41.     {
42.         int SelectNum;
43.
44.         printf("1. Show System Information.\n");
45.         printf("2. Show Performance Infomation.\n");
46.         printf("3. Show Physical Memory Information.\n");
47.         printf("4. Show This Process Information, includes Virtual Memory Informat
    ion and Working Set Information.\n");
48.         printf("Please input the number to select.\n");
49.
50.         scanf("%d", &SelectNum);
51.         switch (SelectNum)
52.         {
53.         case 1:ShowSysInfo();break;
54.         case 2:ShowPerformanceInfo();break;
55.         case 3:ShowPhysicalMemoryInfo();break;
56.         case 4:ShowTHISProcInfo();break;
57.         default:printf("Input Error!\n");break;
58.         }
59.
60.         printf("\nPlease input the command:\n");
61.         printf("[0] Restart the monitor.\n");
62.         printf("Other key to close the monitor.\n");
63.         int command = -1;
64.         scanf("%d", &command);
```

```
65.         if (command == 0)
66.         {
67.             goto Restart;
68.         }
69.         else
70.         {
71.             printf("Thank you for using.\n");
72.         }
73.     }
74. }
```

2. Linux 下直接调用系统命令

(1) 思路

1) 了解 Linux 的 top 命令以及打开 top 之后的 P（按 CPU 使用率）、T（按进程时间累计）、M（按内存占用率）等排序命令。

2) 使用 ps -A 查看所有进程，并找到指定进程的 PID。

3) 使用 top -p PID 查看指定进程的情况。

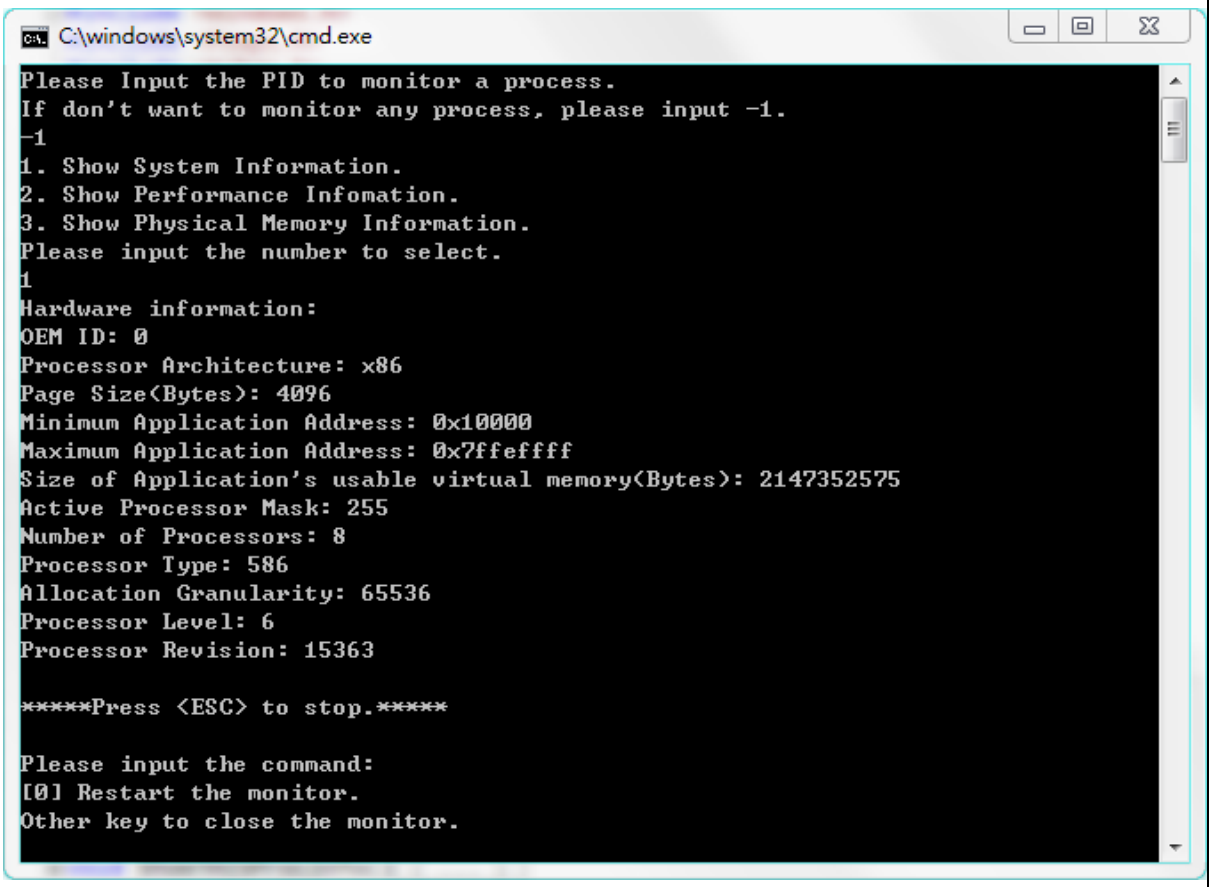
4) 使用 pmap -d PID 查看指定进程的内存使用情况。

5) 额外了解 htop 命令，并对比、使用 htop 完成如 Windows 下任务管理器的操作。

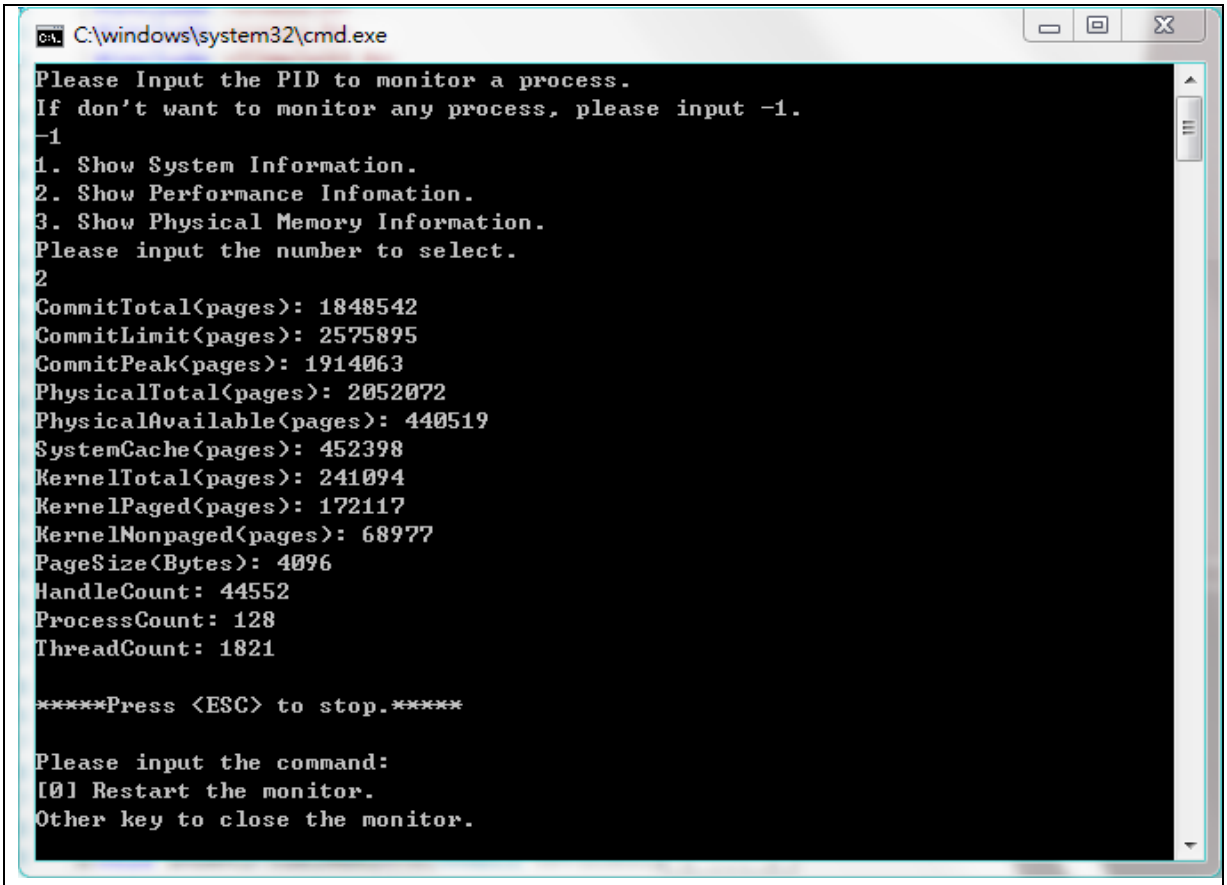
(2) 实验过程略

五、 实验结果和分析

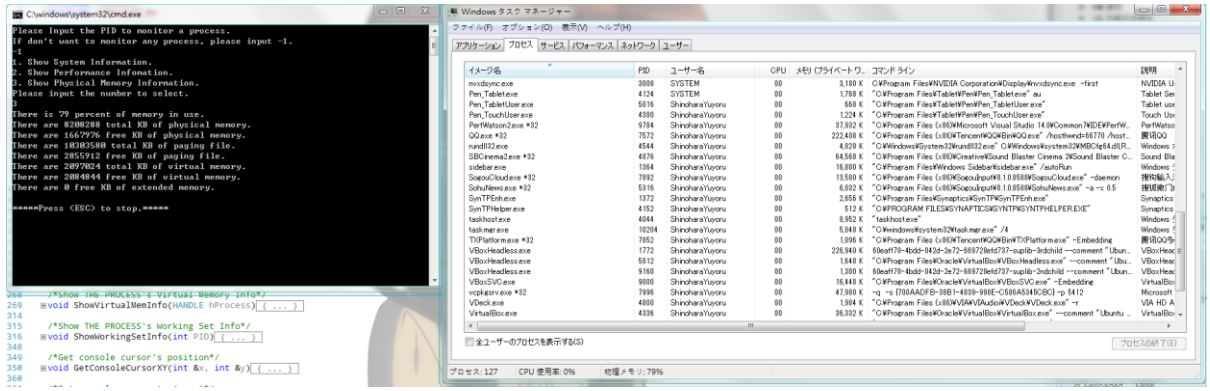
1. Windows 下的实验结果



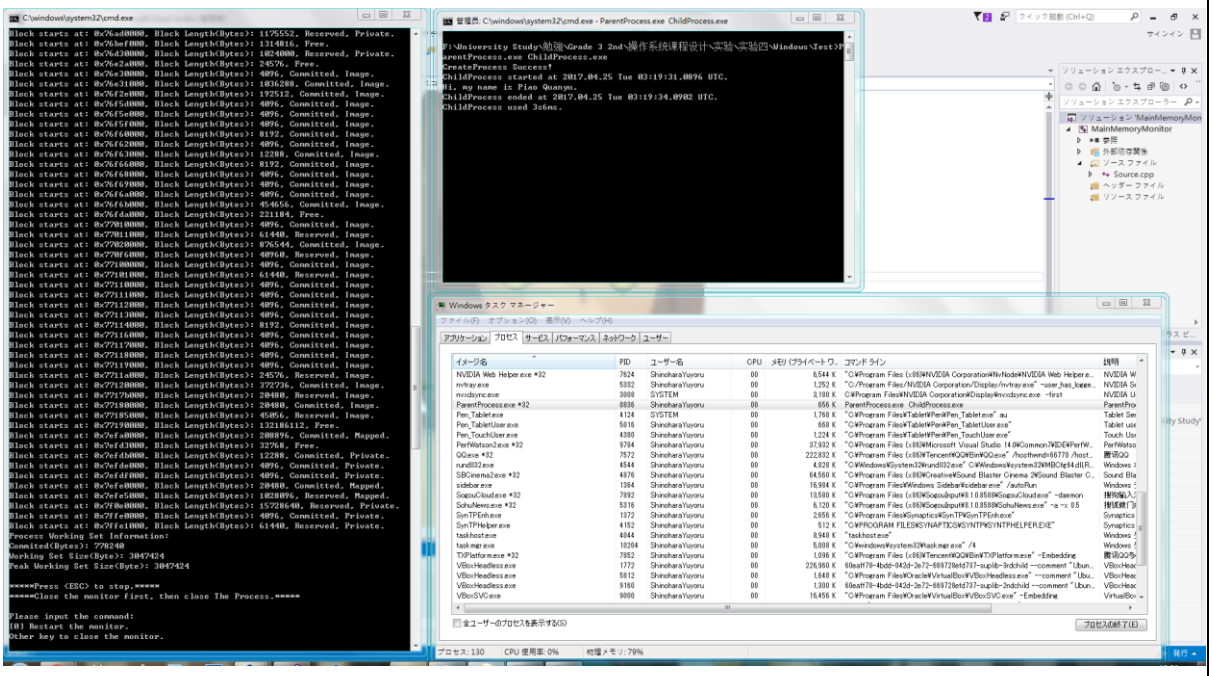
（图 1）`void ShowSysInfo();`模块



（图 2）`void ShowPerformanceInfo();`模块

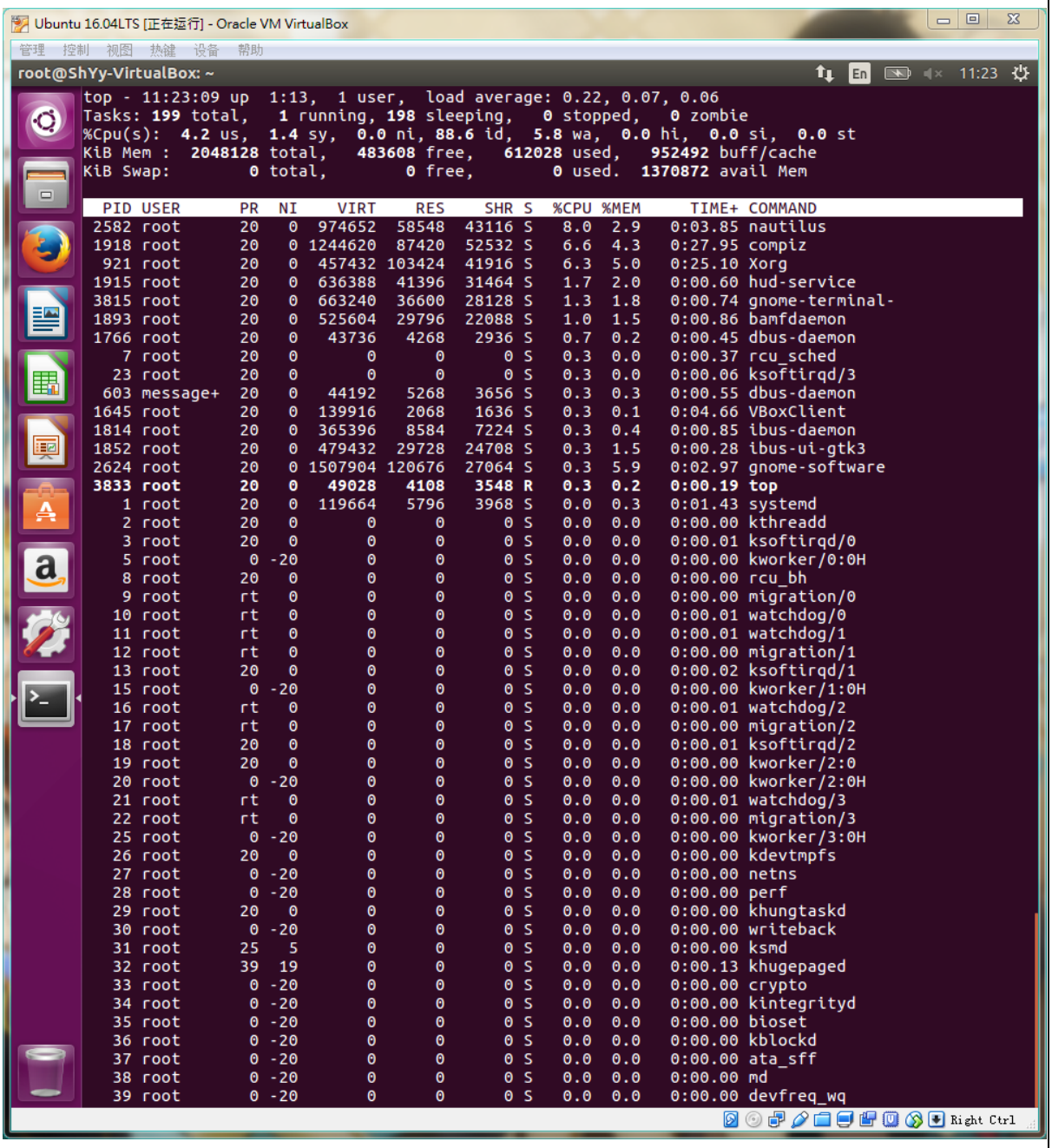


（图 3）`void ShowPhysicalMemoryInfo();`模块

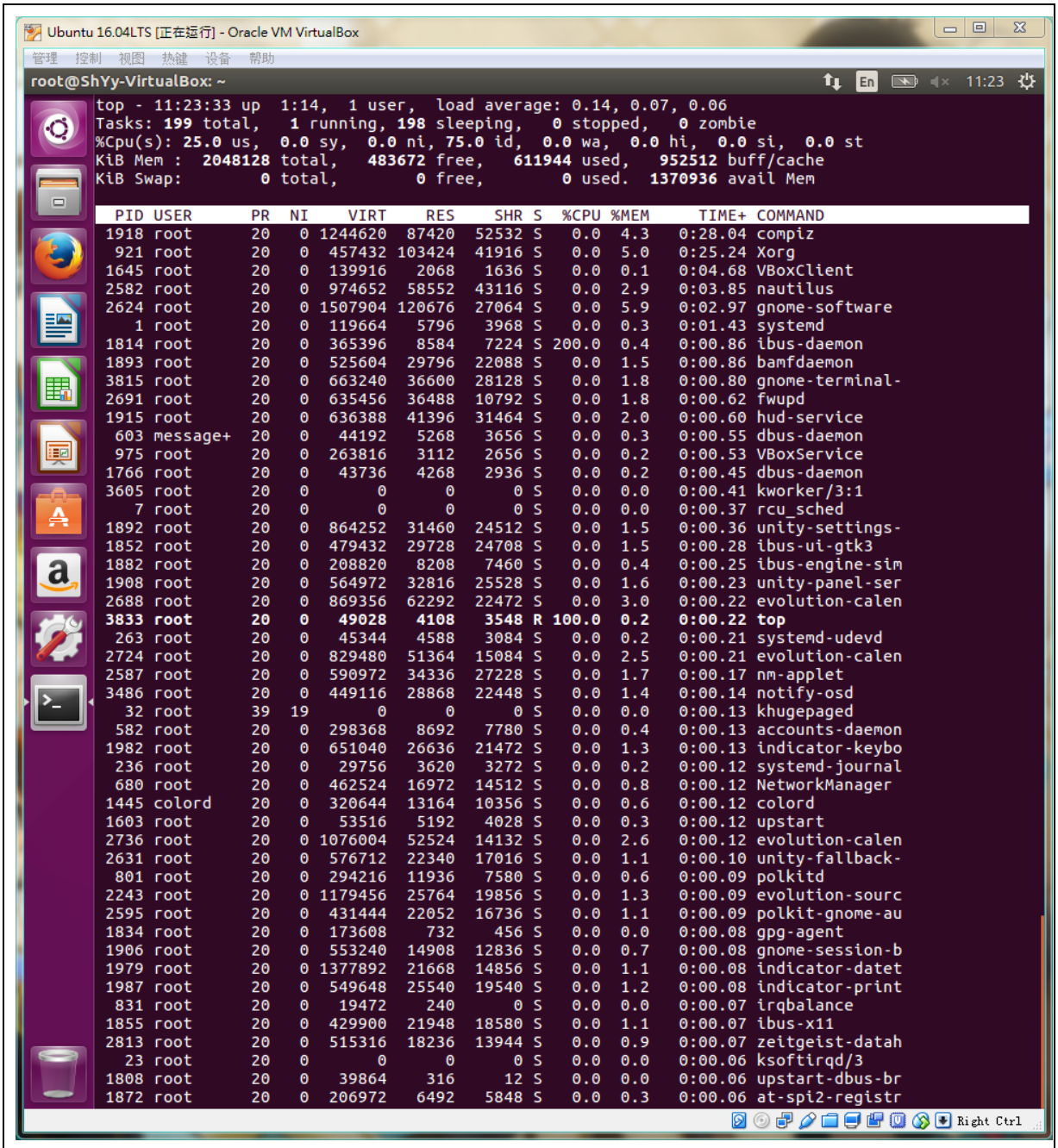


（图 4）`void ShowTHISProcInfo();`模块

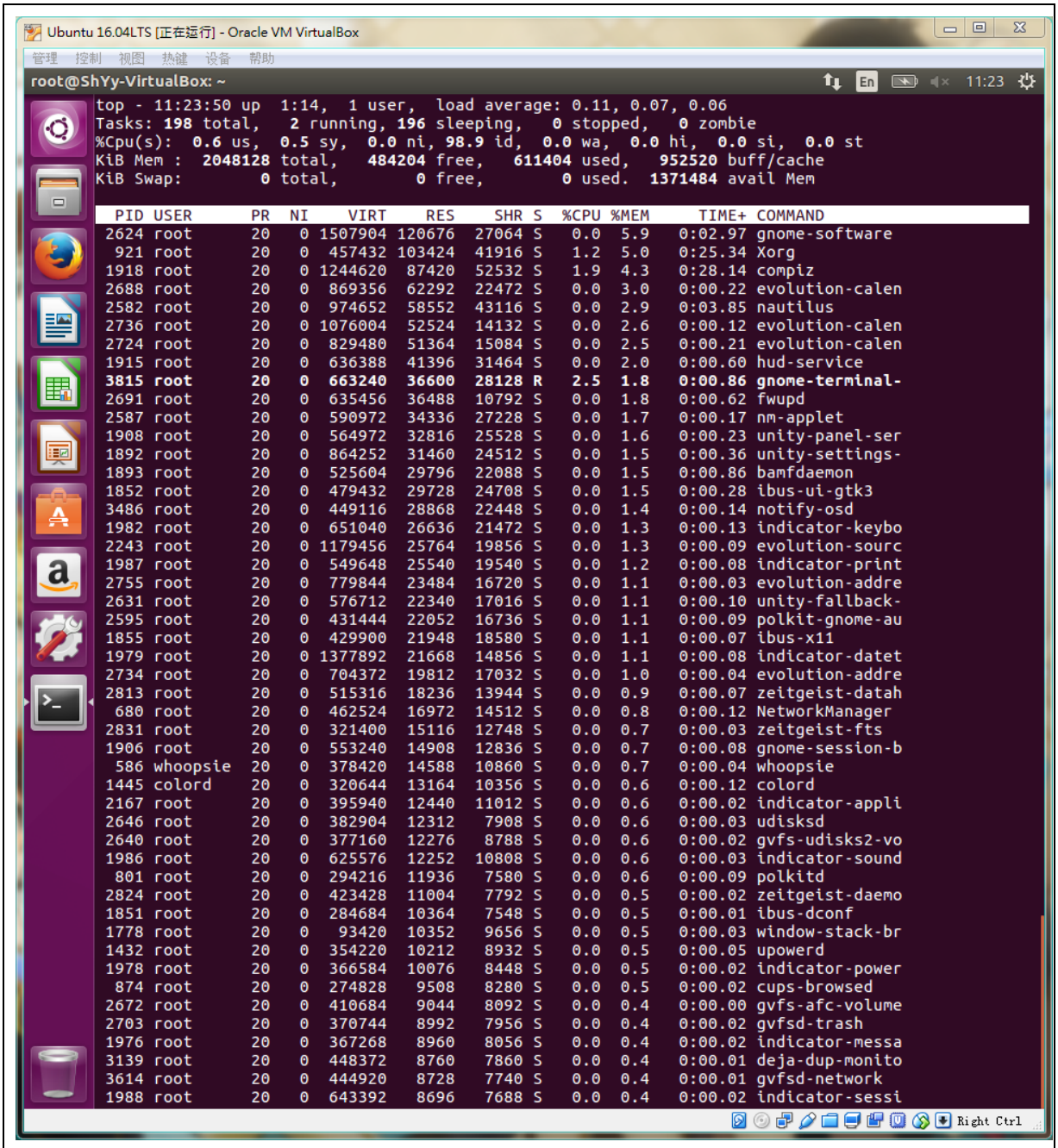
2. Linux 下的实验结果



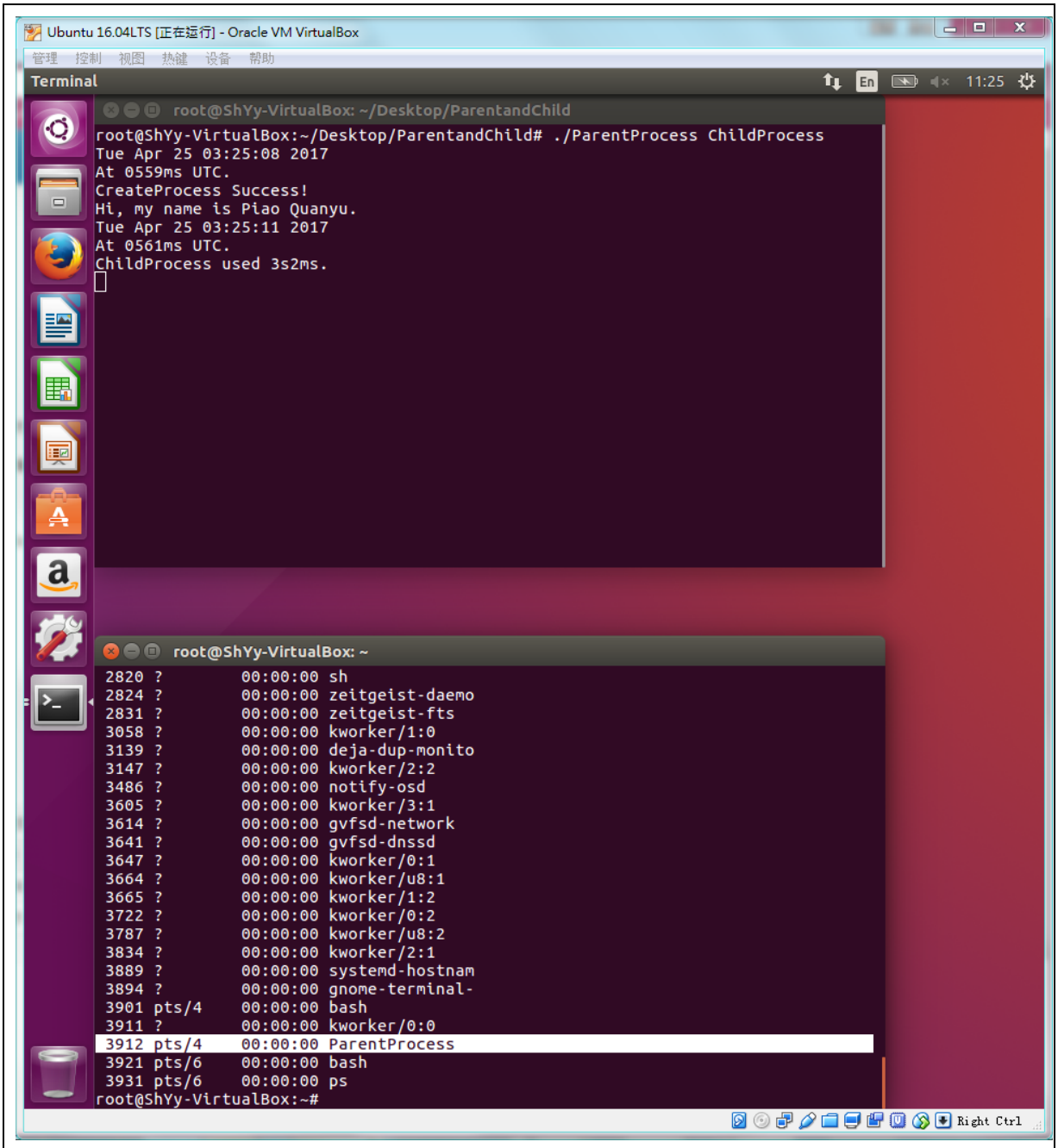
（图 1）top 命令，P（按 CPU 使用率）排序



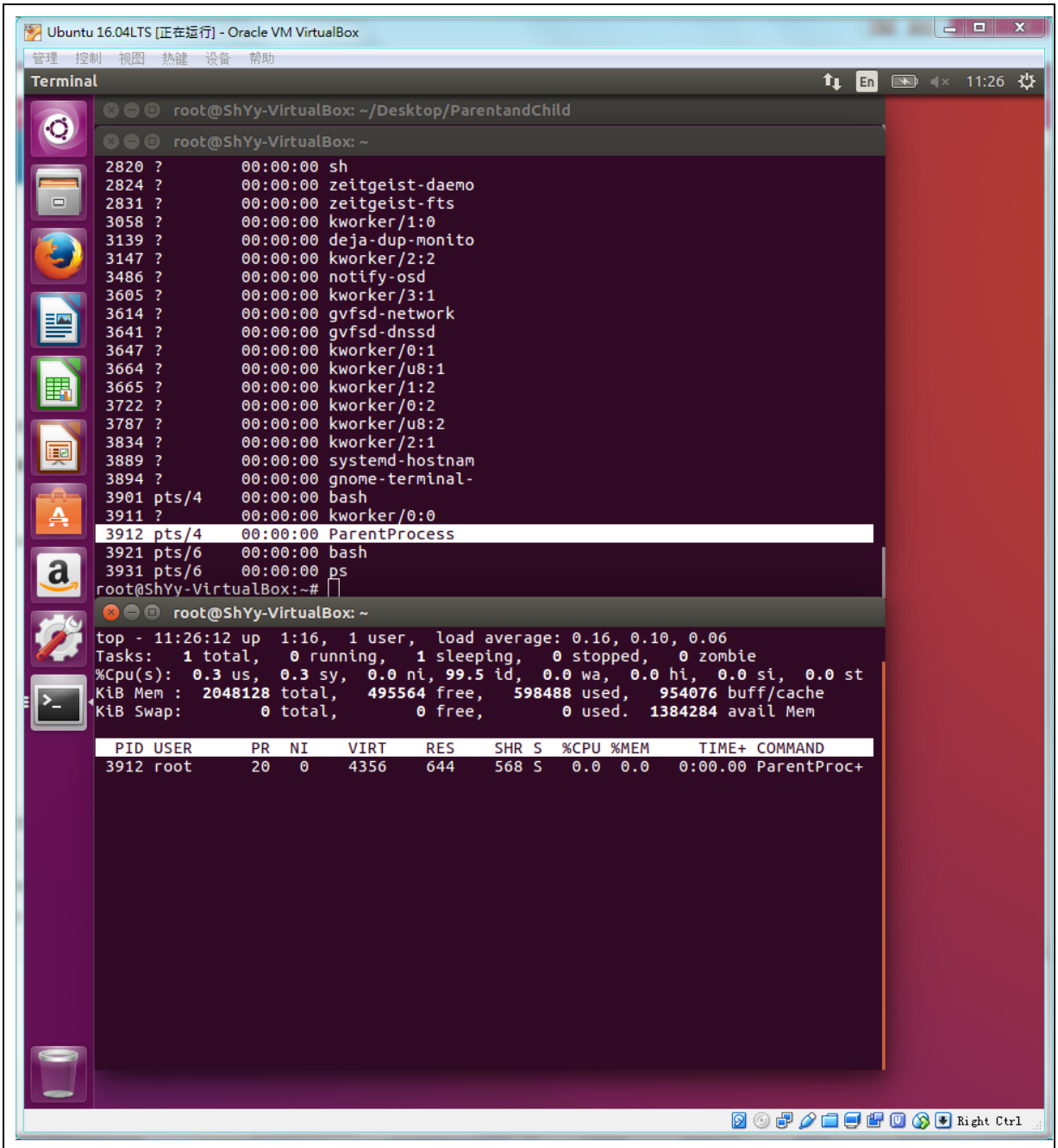
（图 2）top 命令，T（按进程时间累计）排序



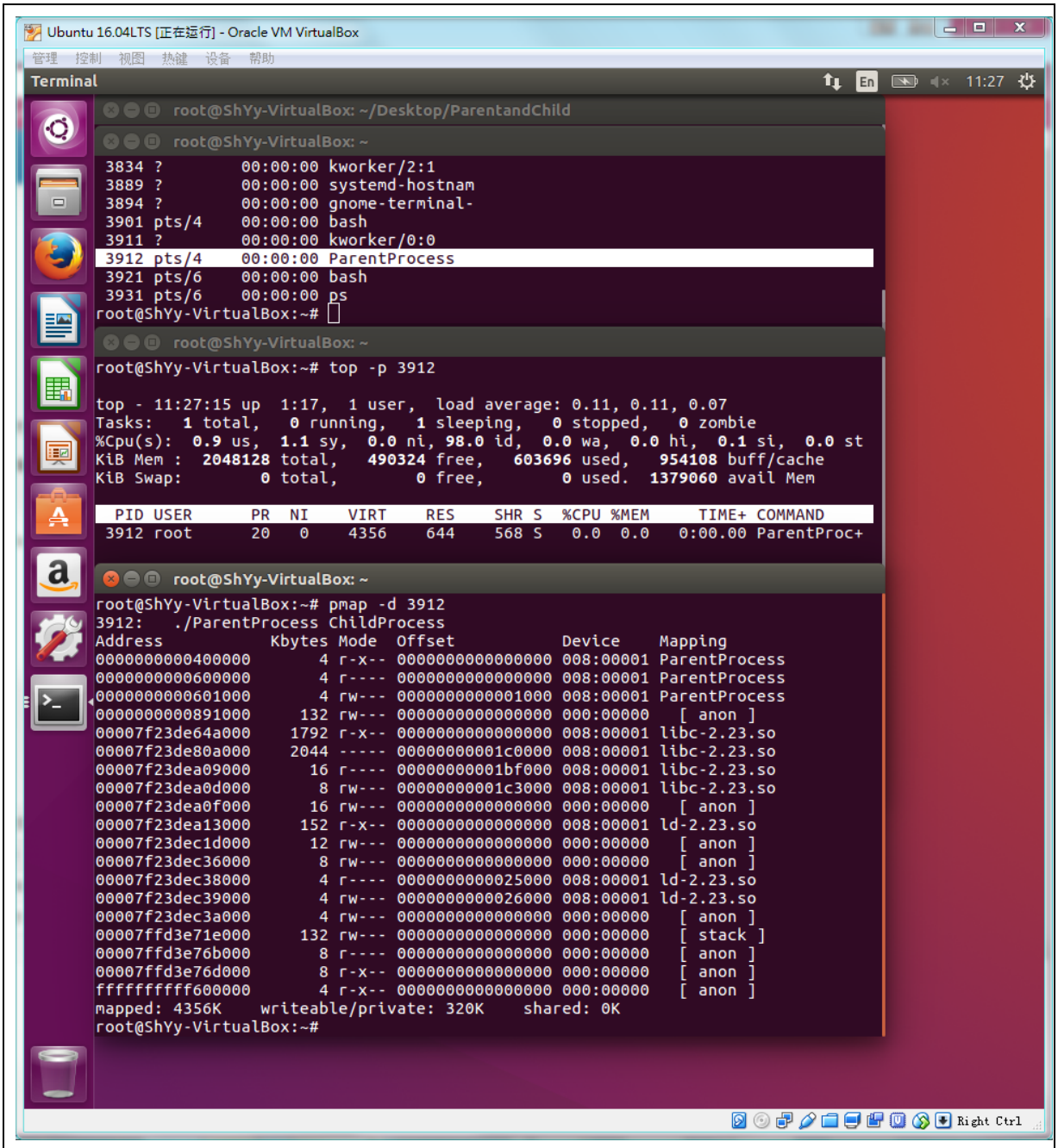
（图 3）top 命令，M（按内存占用率）排序



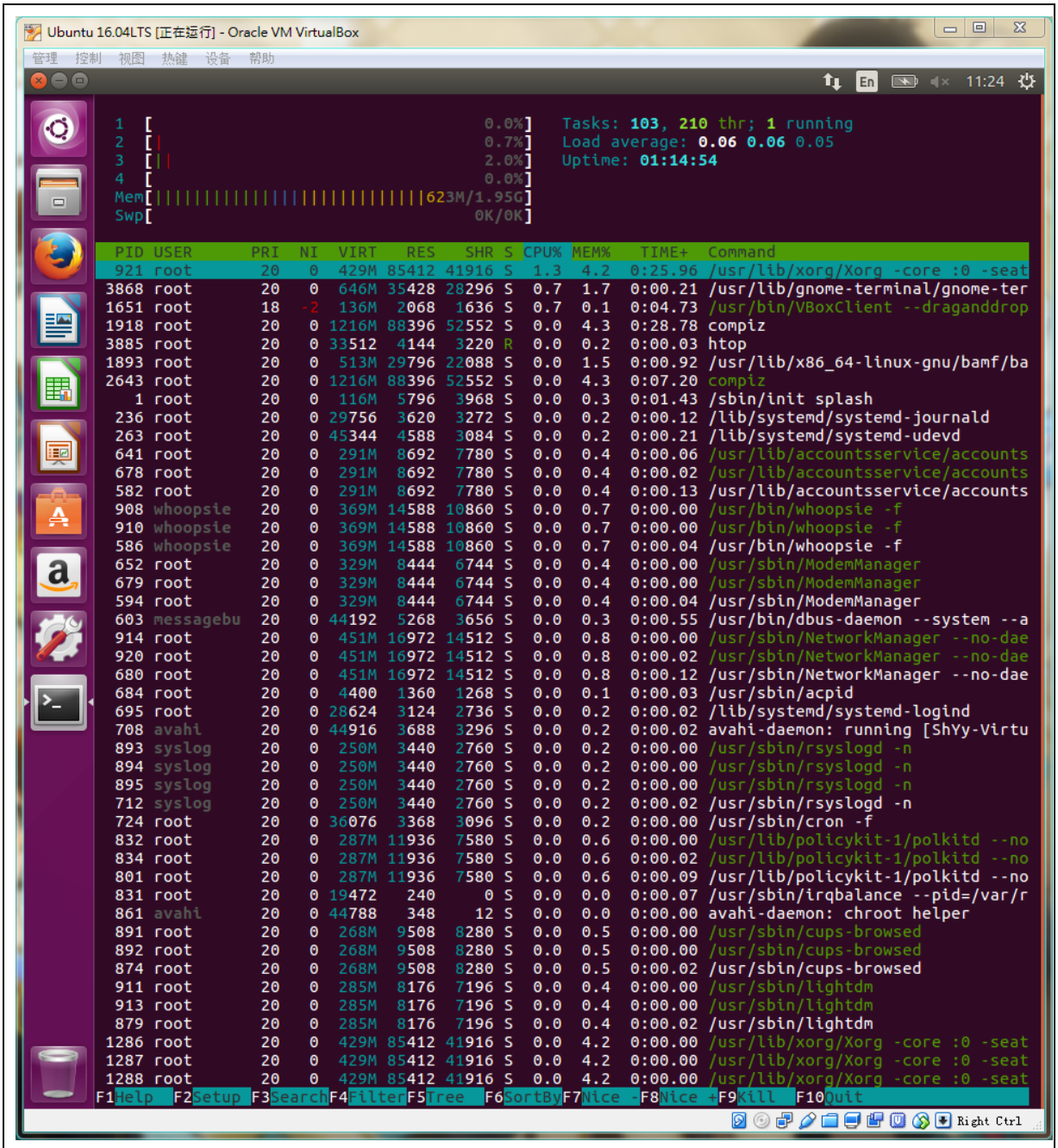
（图 4）ps -A 命令，查看所有进程，获取到 ParentProcess 进程的 PID



（图 5）top -p PID 命令，查看 ParentProcess 进程的情况



（图 6）pmap -d PID 命令，查看 ParentProcess 进程的内存使用情况


Ubuntu 16.04LTS [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
11:24
1 [0.0%]
2 [| 0.7%]
3 [|| 2.0%]
4 [0.0%]
Mem[|||||||||||||||||||||623M/1.95G]
Swp[0K/0K]
Tasks: 103, 210 thr; 1 running
Load average: 0.06 0.06 0.05
Uptime: 01:14:54
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
921 root 20 0 429M 85412 41916 S 1.3 4.2 0:25.96 /usr/lib/xorg/Xorg -core :0 -seat
3868 root 20 0 646M 35428 28296 S 0.7 1.7 0:00.21 /usr/lib/gnome-terminal/gnome-ter
1651 root 18 -2 136M 2068 1636 S 0.7 0.1 0:04.73 /usr/bin/VBoxClient --draganddrop
1918 root 20 0 1216M 88396 52552 S 0.0 4.3 0:28.78 compiz
3885 root 20 0 33512 4144 3220 R 0.0 0.2 0:00.03 htop
1893 root 20 0 513M 29796 22088 S 0.0 1.5 0:00.92 /usr/lib/x86_64-linux-gnu/bamf/ba
2643 root 20 0 1216M 88396 52552 S 0.0 4.3 0:07.20 compiz
1 root 20 0 116M 5796 3968 S 0.0 0.3 0:01.43 /sbin/init splash
236 root 20 0 29756 3620 3272 S 0.0 0.2 0:00.12 /lib/systemd/systemd-journald
263 root 20 0 45344 4588 3084 S 0.0 0.2 0:00.21 /lib/systemd/systemd-udevd
641 root 20 0 291M 8692 7780 S 0.0 0.4 0:00.06 /usr/lib/accountsservice/accounts
678 root 20 0 291M 8692 7780 S 0.0 0.4 0:00.02 /usr/lib/accountsservice/accounts
582 root 20 0 291M 8692 7780 S 0.0 0.4 0:00.13 /usr/lib/accountsservice/accounts
908 whoopsie 20 0 369M 14588 10860 S 0.0 0.7 0:00.00 /usr/bin/whoopsie -f
910 whoopsie 20 0 369M 14588 10860 S 0.0 0.7 0:00.00 /usr/bin/whoopsie -f
586 whoopsie 20 0 369M 14588 10860 S 0.0 0.7 0:00.04 /usr/bin/whoopsie -f
652 root 20 0 329M 8444 6744 S 0.0 0.4 0:00.00 /usr/sbin/ModemManager
679 root 20 0 329M 8444 6744 S 0.0 0.4 0:00.00 /usr/sbin/ModemManager
594 root 20 0 329M 8444 6744 S 0.0 0.4 0:00.04 /usr/sbin/ModemManager
603 messagebu 20 0 44192 5268 3656 S 0.0 0.3 0:00.55 /usr/bin/dbus-daemon --system --a
914 root 20 0 451M 16972 14512 S 0.0 0.8 0:00.00 /usr/sbin/NetworkManager --no-dae
920 root 20 0 451M 16972 14512 S 0.0 0.8 0:00.02 /usr/sbin/NetworkManager --no-dae
680 root 20 0 451M 16972 14512 S 0.0 0.8 0:00.12 /usr/sbin/NetworkManager --no-dae
684 root 20 0 4400 1360 1268 S 0.0 0.1 0:00.03 /usr/sbin/acpid
695 root 20 0 28624 3124 2736 S 0.0 0.2 0:00.02 /lib/systemd/systemd-logind
708 avahi 20 0 44916 3688 3296 S 0.0 0.2 0:00.02 avahi-daemon: running [ShYy-Virtu
893 syslog 20 0 250M 3440 2760 S 0.0 0.2 0:00.00 /usr/sbin/rsyslogd -n
894 syslog 20 0 250M 3440 2760 S 0.0 0.2 0:00.00 /usr/sbin/rsyslogd -n
895 syslog 20 0 250M 3440 2760 S 0.0 0.2 0:00.00 /usr/sbin/rsyslogd -n
712 syslog 20 0 250M 3440 2760 S 0.0 0.2 0:00.02 /usr/sbin/rsyslogd -n
724 root 20 0 36076 3368 3096 S 0.0 0.2 0:00.00 /usr/sbin/cron -f
832 root 20 0 287M 11936 7580 S 0.0 0.6 0:00.00 /usr/lib/policykit-1/polkitd --no
834 root 20 0 287M 11936 7580 S 0.0 0.6 0:00.02 /usr/lib/policykit-1/polkitd --no
801 root 20 0 287M 11936 7580 S 0.0 0.6 0:00.09 /usr/lib/policykit-1/polkitd --no
831 root 20 0 19472 240 0 S 0.0 0.0 0:00.07 /usr/sbin/irqbalance --pid=/var/r
861 avahi 20 0 44788 348 12 S 0.0 0.0 0:00.00 avahi-daemon: chroot helper
891 root 20 0 268M 9508 8280 S 0.0 0.5 0:00.00 /usr/sbin/cups-browsed
892 root 20 0 268M 9508 8280 S 0.0 0.5 0:00.00 /usr/sbin/cups-browsed
874 root 20 0 268M 9508 8280 S 0.0 0.5 0:00.02 /usr/sbin/cups-browsed
911 root 20 0 285M 8176 7196 S 0.0 0.4 0:00.00 /usr/sbin/lightdm
913 root 20 0 285M 8176 7196 S 0.0 0.4 0:00.00 /usr/sbin/lightdm
879 root 20 0 285M 8176 7196 S 0.0 0.4 0:00.02 /usr/sbin/lightdm
1286 root 20 0 429M 85412 41916 S 0.0 4.2 0:00.00 /usr/lib/xorg/Xorg -core :0 -seat
1287 root 20 0 429M 85412 41916 S 0.0 4.2 0:00.00 /usr/lib/xorg/Xorg -core :0 -seat
1288 root 20 0 429M 85412 41916 S 0.0 4.2 0:00.00 /usr/lib/xorg/Xorg -core :0 -seat
F1Help F2Setup F3SearchF4FilterF5Tree F6SortByF7Nice -F8Nice +F9Kill F10Quit
Right Ctrl

（图 7）htop 命令

六、 讨论、心得

此次实验，模拟了我们在 Windows 下经常使用的任务管理器，了解了操作系统对进程信息的控制，对进程的操作。对操作系统的内存管理、地址空间布局等概念有了进一步认识。

值得注意的是，此次实验在 Linux 下并没有编码，但是通过 Terminal 以及相应的系统命令我们同样完成了与 Windows 下类似的操作。而且通过简要对比 top 和 htop 命令，可以发现 htop 比 top 要更加友好、易用。