

**操作系统课程设计**

|  |  |
| --- | --- |
| **实验四、程序控制** | **实验四，内存监控** |

学院：计算机学院

专业：计算机科学与技术

学生姓名：夏奇拉

学号： 1820171025

： 07111705

目录

[目的](#__RefHeading___Toc1766_1221548062) [3](#__RefHeading___Toc1766_1221548062)

[问题讨论](#__RefHeading___Toc1744_1221548062) [3](#__RefHeading___Toc1744_1221548062)

[执行 [Windows]](#__RefHeading___Toc1768_1221548062)  [5](#__RefHeading___Toc1768_1221548062)

[结果与分析 [Windows]](#__RefHeading___Toc1770_1221548062)  [9](#__RefHeading___Toc1770_1221548062)

[执行 [Linux]](#__RefHeading___Toc604_992951594)  [10](#__RefHeading___Toc604_992951594)

[结果与分析[Linux]](#__RefHeading___Toc1770_12215480621)  [15](#__RefHeading___Toc1770_12215480621)

[参考：](#__RefHeading___Toc606_992951594)  [16](#__RefHeading___Toc606_992951594)

# 目的

Windows 实验：

Windows 设计了一个内存监视器，它需要：

实时显示当前系统的内存使用情况，包括系统地址空间的布局和物理内存的使用情况；

实时显示实验2进程控制（ParentProcess.exe）的虚拟地址空间布局和工作集信息

相关系统调用：

GetSystemInfo、VirtualQueryEx、VirtualAlloc、GetPerformanceInfo、GlobalMemoryStatusEx...

Linux实验：

使用top命令查看系统，子命令P、T、M

使用ps -A查看所有进程，找到ProcessParent的pid

使用top -p pid查看ProcessParent程序的状态；

使用pmap -d pid查看ProcessParent的内存使用情况

# 问题讨论

实验要求程序必须显示

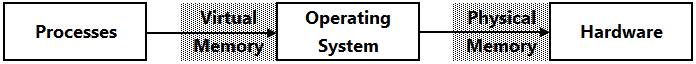
1. 系统的内存使用
   1. 系统地址空间布局
   2. 物理内存使用
2. 虚拟地址空间布局
3. parentprocess.exe 来自实验室 2 的工作信息

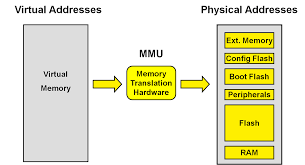
**什么是系统内存？**

计算机系统内存由称为随机存取存储器 (RAM) 的物理内存和虚拟内存组成。系统内存不是永久存储，就像计算机关闭时保存其内容的硬盘驱动器一样。

进程是包含在程序中的一组任务，它通过一系列称为线程的指令执行。进程在操作系统上运行，操作系统管理所有运行进程的硬件资源。

操作系统为在物理内存上运行的所有进程提供虚拟内存。操作系统的虚拟内存管理器使用一种称为分页的方法将虚拟地址空间映射到物理地址空间，这样所有进程都可以在物理内存上运行。





# 执行 [视窗]

**获取系统信息**

GetSystemInfo() 检索有关当前系统的信息。它接受指向接收信息的**SYSTEM\_INFO**结构的指针。

SYTEM\_INFO结构包含有关当前计算机系统的信息**。**

* wProcessorArchitecture：已安装操作系统的处理器架构。
* DwPageSize：页面大小和页面保护和提交的粒度。这是**VirtualAlloc**函数使用的页面大小。
* lpMinimumApplicationAddress：指向应用程序和动态链接库 (DLL) 可访问的最低内存地址的指针。
* lpMaximumApplicationAddress：指向应用程序和 DLL 可访问的最高内存地址的指针。
* dwAllocationGranularity：可以分配虚拟内存的起始地址的粒度。

**虚拟查询Ex**

**VirtualQueryEx**检索有关指定进程的虚拟地址空间内一系列连续页面的信息。返回值是信息缓冲区中返回的实际字节数。所有页面的可能状态包括 MEM\_COMMIT、MEM\_RESERVE、MEM\_FREE、MEM\_PRIVATE、MEM\_MAPPED 或 MEM\_IMAGE。如果函数失败，则返回值为零。它采用以下参数：

* HANDLE hProcess：查询内存信息的进程句柄。
* LPCVOID lpAddress：指向要查询的页面区域的基地址的指针。 **GetSystemInfo**函数用于确定主机上页面的大小。
* PMEMORY\_BASIC\_INFORMATION lpBuffer：指向 MEMORY\_BASIC\_INFORMATION 结构的指针**，**其中返回有关指定页面范围的信息。
* SIZE\_T dwLength：lpBuffer 参数指向的缓冲区的大小，以字节为单位。

**MEMORY\_BASIC\_INFORMATION**结构包含有关进程虚拟地址空间中页面范围的信息。其成员包括：

* PVOID BaseAddress：指向页面区域的基地址
* PVOID AllocationBase：指向由**VirtualAlloc**函数分配的页面范围的基地址的指针。 BaseAddress 成员指向的页面包含在此分配范围内。
* DWORD AllocationProtect：最初分配区域时的内存保护选项。
* WORD 分区ID
* SIZE\_T RegionSize：从所有页面具有相同属性的基地址开始的区域大小，以字节为单位
* DWORD State：该区域中页面的状态。状态包括 MEM\_COMMIT、MEM\_FREE 和 MEM\_RESERVE。
* DWORD Protect：区域内页面的访问保护
* DWORD 类型：区域中页面的类型，包括MEM\_IMAGE、MEM\_MAPPED和MEM\_PRIVATE。

**虚拟分配**

**VirtualAlloc**函数保留、提交或更改调用进程的虚拟地址空间中页面区域的状态。此函数分配的内存自动初始化为零。如果函数成功，返回值是页面分配区域的基地址。

每个页面都有一个关联的页面状态（空闲、保留或提交）。 **VirtualAlloc**函数可以执行以下操作：

* 提交保留页面区域
* 保留一个免费页面区域
* 同时保留和提交空闲页面区域

它需要 4 个参数。

* LPVOID LpAddress：要分配的区域的起始地址。
* SIZE\_T dwSize：区域大小，以字节为单位
* DWORD flAllocationType：内存分配的类型。此参数必须包含以下值之一：MEM\_COMMIT、MEM\_RESERVE、MEM\_RESET、MEM\_RESET\_UNDO
* DWARD flProtect：要分配的页面区域的内存保护。

如何使用此功能的示例： [Reserving and Committing Memory - Win32 apps |微软学习](https://learn.microsoft.com/en-us/windows/win32/Memory/reserving-and-committing-memory)

**获取性能信息**

GetPerfomanceInfo**函数**保留包含在**PERFORMANCE\_INFORMATION**结构**中的性能值**。如果函数成功，则返回值为 TRUE。如果函数失败，则返回值为 FALSE。它采用以下参数：

* PPERFORMANCE\_INFORMATION pPerformanceInformation：指向接收性能信息的**PERFORMANCE\_INFORMATION**结构的指针。
* DWORD cb： **PERFORMANCE\_INFORMATION**结构的大小，以字节为单位。

**PERFORMANCE\_INFORMATION**结构包含性能信息。其成员包括：

* DWORD cb：此结构的大小（以字节为单位）
* SIZE\_T CommitTotal：系统当前提交的页数。
* SIZE\_T CommitLimit：在不扩展页面文件的情况下，系统可以提交的当前最大页面数。
* SIZE\_T 提交峰值；自上次系统重启后同时处于提交状态的最大页数。
* SIZE\_T PhysicalTotal：实际物理内存量，以页为单位。
* SIZE\_T PhysicalAvailable：当前可用的物理内存量，以页为单位。
* SIZE\_T SystemCache：系统缓存内存量，以页为单位。这是备用列表加上系统工作集的大小。
* SIZE\_T KernelTotal：当前在分页和非分页内核池中的内存总和，以页为单位。
* SIZE\_T KernelPaged：当前在分页内核池中的内存，以页为单位。
* SIZE\_T KernelNopaged：当前在非分页内核池中的内存，以页为单位。
* SIZE\_T PageSize：页面的大小，以字节为单位。
* DWORD 句柄数：
* 当前打开的句柄数。
* DWORD ProcessCount：当前进程数。
* DWORD ThreadCount：当前线程数。

**全局内存状态Ex**

**GlobalMemoryStatusEx**函数检索有关系统当前物理和虚拟内存使用情况的信息。如果函数成功，则返回值为非零。它接受一个参数：

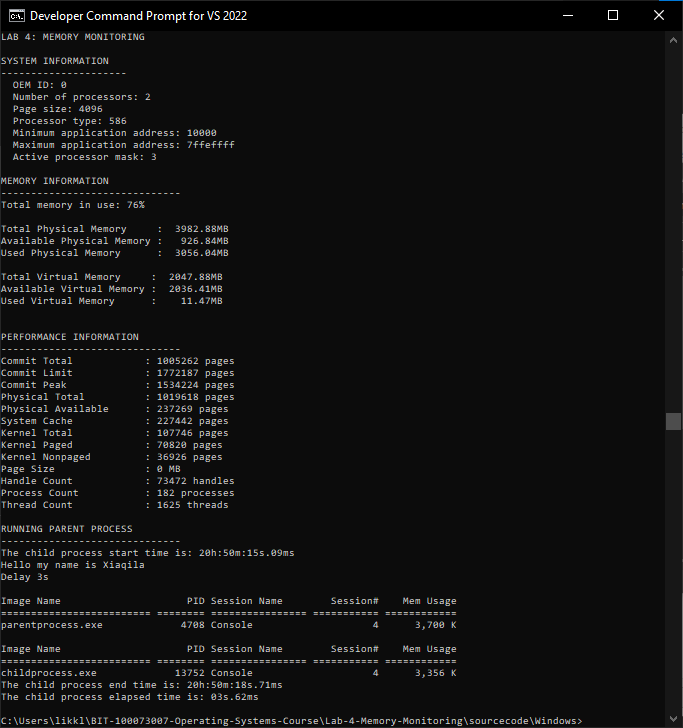
* LPMEMORYSTATUSEX lpBuffer：指向接收有关内存可用性信息的**MEMORYSTATUSEX结构的指针。**

MEMORYSTATUSEX结构包含有关物理和虚拟内存（包括扩展内存）的当前状态的信息**。**其成员包括：

* DWORD dwLength：结构的大小（以字节为单位）。该成员必须在调用**GlobalMemoryStatusEx之前设置**
* DWORD dwMemoryLoad：一个介于 0 和 100 之间的数字，指定正在使用的物理内存的近似百分比（0 表示没有内存使用，100 表示完全使用内存）。
* DWORDLONG ullTotalPhys：以字节为单位的实际物理内存量。
* DWORDLONG ullAvailPhys：当前可用的物理内存量，以字节为单位。它是备用列表、空闲列表和零列表的大小之和。
* DWORDLONG ullTotalPageFile：系统或当前进程的当前提交内存限制，以字节为单位，以较小者为准。要获取系统范围内提交的内存限制，请调用**GetPerformanceInfo**
* DWORDLONG ullAvailPageFile：当前进程可以提交的最大内存量，以字节为单位。该值等于或小于系统范围内的可用提交值。要计算系统范围内的可用提交值，请调用**GetPerformanceInfo**并从 CommitLimit 的值中减去**CommitTotal的**值。
* DWORDLONG ullTotalVirtual：调用进程虚拟地址空间的用户模式部分的大小，以字节为单位
* DWORDLONG ullAvailVirtual：当前在调用进程的虚拟地址空间的用户模式部分中未保留和未提交的内存量，以字节为单位。
* DWORDLONG ullAvailExtendedVirtual：保留。该值始终为 0。

|  |
| --- |
| #include <windows.h>  #include <stdio.h>  #include <tchar.h>  #include <进程.h>  #include <Psapi.h>  #pragma comment(lib, "user32.lib")  // 用于将字节转换为KB  #define DIV 1024  // 指定打印数字的字段宽度。  // 格式说明符“%\*I64d”中的星号取一个整数  // 参数并使用它来填充和右对齐数字。  #定义宽度 7  无效打印系统信息（）  {  printf("\n系统信息");  printf("\n--------------------\n");  SYSTEM\_INFO siSysInfo;  // 将硬件信息复制到 SYSTEM\_INFO 结构中。  GetSystemInfo(&siSysInfo);  // 显示 SYSTEM\_INFO 结构的内容。  printf("OEM ID: %u\n", siSysInfo.dwOemId);  printf("处理器数量：%u\n",  siSysInfo.dwNumberOfProcessors);  printf("页面大小：%u\n", siSysInfo.dwPageSize);  printf("处理器类型: %u\n", siSysInfo.dwProcessorType);  printf("最小应用地址：%lx\n",  siSysInfo.lpMinimumApplicationAddress);  printf("最大应用地址：%lx\n",  siSysInfo.lpMaximumApplicationAddress);  printf(" 活动处理器掩码: %u\n",  siSysInfo.dwActiveProcessorMask);  }  无效 PrintMemoryInfo()  {  printf("\n内存信息\n");  printf("----------------------------\n");  MEMORYSTATUSEX ms = {sizeof(MEMORYSTATUSEX)};  ms.dwLength = sizeof(ms);  // 检索有关系统当前物理内存使用情况的信息  GlobalMemoryStatusEx(&ms);  printf("使用的总内存: %ld%%\n", ms.dwMemoryLoad);  printf("\n物理内存总量：%8.2I64fMB\n可用物理内存：%8.2I64fMB\n已用物理内存：%8.2I64fMB\n\n", ms.ullTotalPhys / (1024 \* 1024.0), ms.ullAvailPhys / (1024 \* 1024.0), ms.ullTotalPhys / (1024 \* 1024.0) - ms.ullAvailPhys / (1024 \* 1024.0));  printf("总虚拟内存：%8.2I64fMB\n可用虚拟内存：%8.2I64fMB\n已用虚拟内存：%8.2I64fMB\n\n", ms.ullTotalVirtual / (1024 \* 1024.0), ms.ullAvailVirtual / (1024 \* 1024.0 ), ms.ullTotalVirtual / (1024 \* 1024.0) - ms.ullAvailVirtual / (1024 \* 1024.0));  }  无效 PrintPerformanceInfo()  {  printf("\n性能信息\n");  printf("----------------------------\n");  PERFORMANCE\_INFORMATION siPerfInfo；  // 将硬件信息复制到 SYSTEM\_INFO 结构中。  GetPerformanceInfo(&siPerfInfo, siPerfInfo.cb);  printf("提交总数\t\t: %d 页\n", siPerfInfo.CommitTotal);  printf("提交限制\t\t: %d 页\n", siPerfInfo.CommitLimit);  printf("提交峰值\t\t: %d 页\n", siPerfInfo.CommitPeak);  printf("纸本总计\t\t: %d 页\n", siPerfInfo.PhysicalTotal);  printf("物理可用\t: %d 页\n", siPerfInfo.PhysicalAvailable);  printf("系统缓存\t\t: %d 页\n", siPerfInfo.SystemCache);  printf("内核总数\t\t: %d 页\n", siPerfInfo.KernelTotal);  printf("内核分页\t\t: %d 页\n", siPerfInfo.KernelPaged);  printf("Kernel Nonpaged\t\t: %d pages\n", siPerfInfo.KernelNonpaged);  printf("页面大小\t\t: %d MB\n", siPerfInfo.PageSize / (1024 \* 1024.0));  printf("句柄计数\t\t: %d 个句柄\n", siPerfInfo.HandleCount);  printf("进程数\t\t: %d 进程\n", siPerfInfo.ProcessCount);  printf("线程计数\t\t: %d 线程\n", siPerfInfo.ThreadCount);  }  void PrintRunningProcesses（无效）  {  printf("\n当前正在运行的进程\n");  printf("----------------------------\n");  system("tasklist /FI \"IMAGENAME eq parentprocess.exe\"");  system("tasklist /FI \"IMAGENAME eq childprocess.exe\"");  system("tasklist /FI \"IMAGENAME eq memorymonitoring.exe\"");    }  void \_tmain(int argc, TCHAR \*argv[])  {  printf("实验 4：内存监控\n");  打印系统信息();  打印内存信息（）；  打印性能信息（）；  启动信息 si;  PROCESS\_INFORMATION pi；  ZeroMemory(&si, sizeof(si));  si.cb = sizeof(si);  ZeroMemory(&pi, sizeof(pi));  if (argc != 3) /\* argc 应该是 2 才能正确执行 \*/  {  printf("内存监控中的使用情况: %s [cmdline]\n", argv[0]);  返回;  }  // 启动子进程。  if (!CreateProcess(NULL, // 没有模块名称（使用命令行）  argv[1], // 命令行  NULL, // 进程句柄不可继承  NULL, // 线程句柄不可继承  FALSE, // 将句柄继承设置为 FALSE  0, // 没有创建标志  NULL, // 使用父级的环境块  NULL, // 使用父目录的起始目录  &si, // 指向 STARTUPINFO 结构的指针  &pi) // 指向 PROCESS\_INFORMATION 结构的指针  )  {  printf("CreateProcess 失败 (%d).\n", GetLastError());  返回;  }  PrintRunningProcesses();  // 只询问任务列表命令  睡眠（5000）；    返回 0；  } |

## 结果与分析 [Windows]

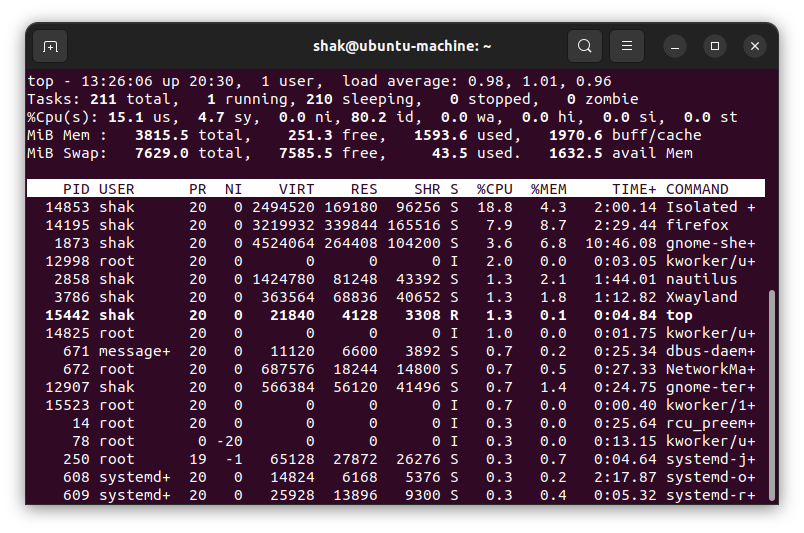


# 执行 [Linux]

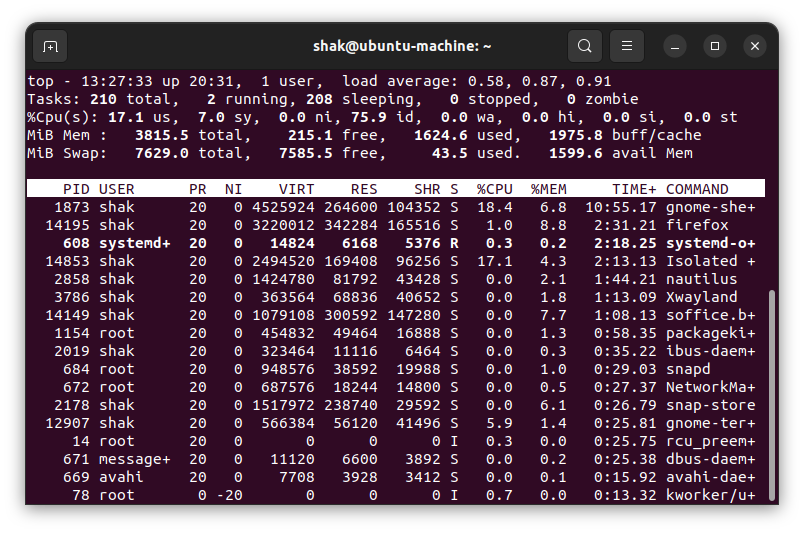
top（进程表）命令显示了 Linux 中正在运行的进程的实时视图。默认情况下，它按 %CPU 列对进程列表进行排序。以下命令可用于使用不同的列进行排序：

* P. 按 %CPU 列排序
* T. 按 TIME+ 列排序
* M. 按 %MEM 列排序

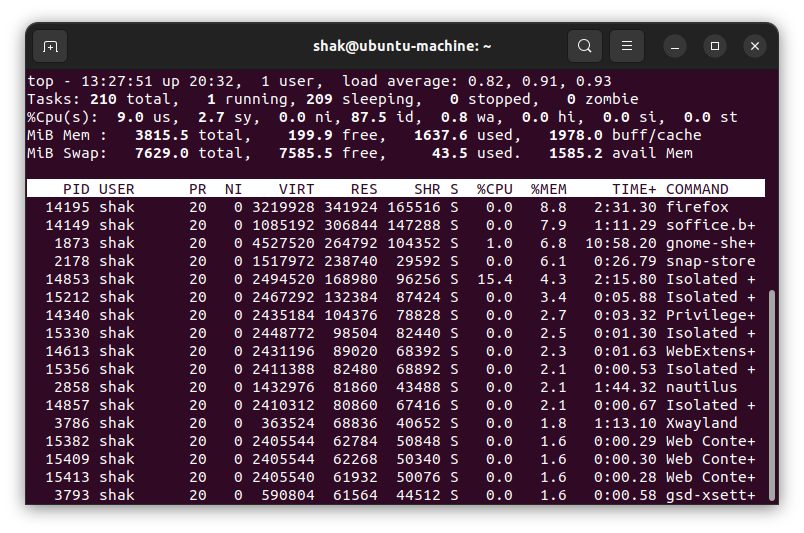
**使用 P 子命令按 %CPU 排序**

****

**按时间排序 + 使用 T 子命令**

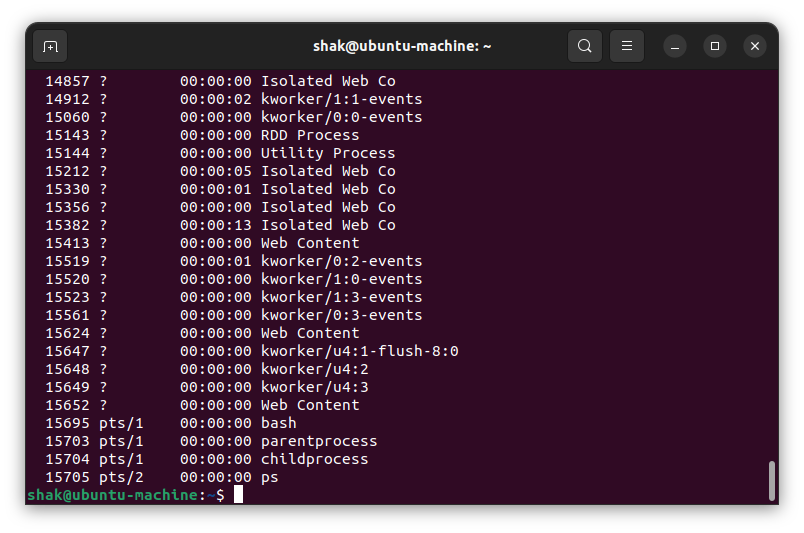
****

**使用 M 子命令按 %MEM 排序**

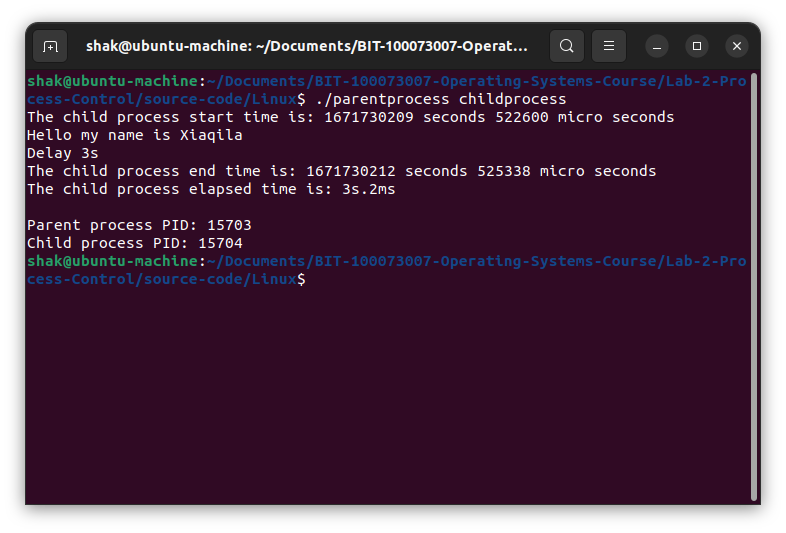
****

**使用 ps -A 查找父进程的 PID**

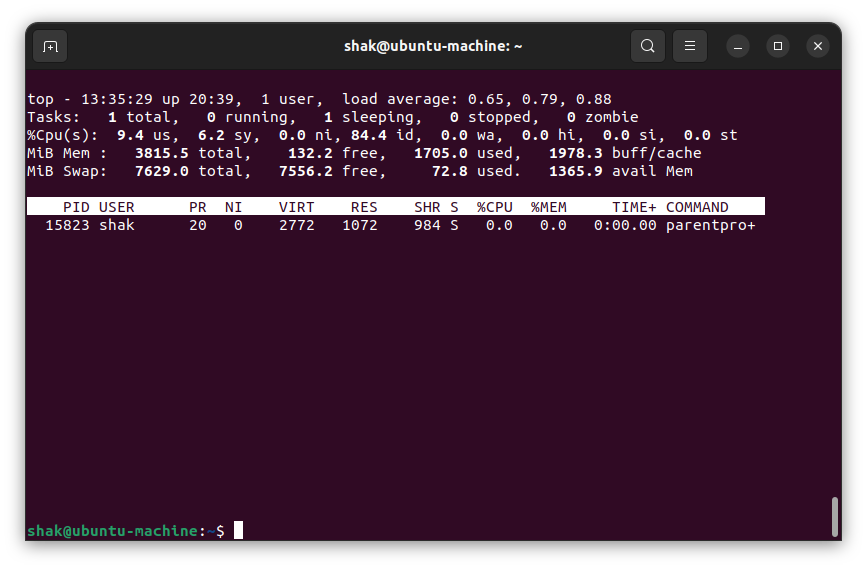
ps 命令允许您轻松列出系统上运行的进程的状态。



parentprocess 的 pid – 15703。这在 parentprocess 命令的输出中得到确认

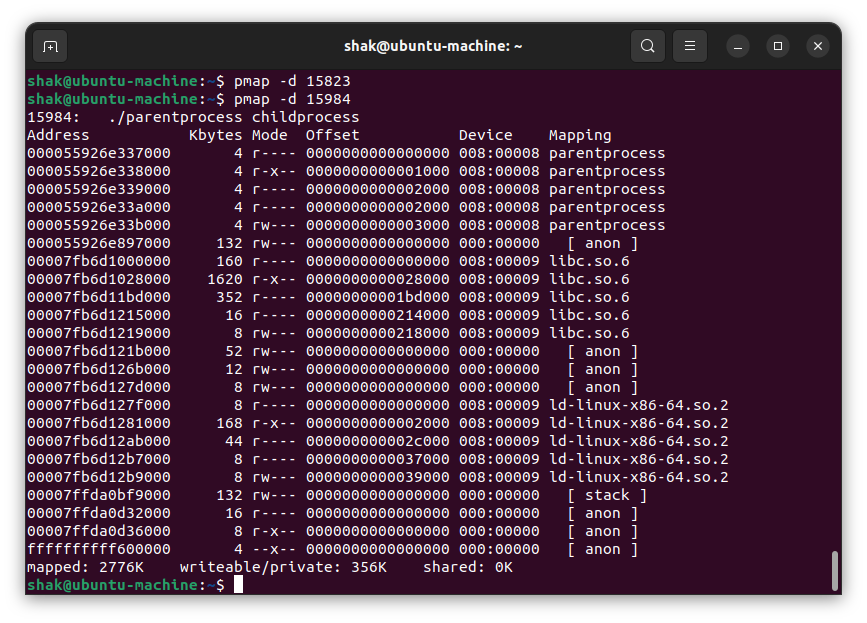


**使用 top -p pid 检查父进程的状态**

****

**使用pmap -d pid查看ProcessParent的内存使用情况**

的pmap命令用于显示进程的内存映射。



# 参考：

* <https://learn.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsysteminfo>
* <https://learn.microsoft.com/en-us/windows/win32/api/sysinfoapi/ns-sysinfoapi-system_info>
* <https://learn.microsoft.com/en-gb/windows/win32/sysinfo/getting-hardware-information?redirectedfrom=MSDN>
* <https://www.installsetupconfig.com/win32programming/windowsvolumeapis1_6.html>
* <https://learn.microsoft.com/en-us/windows/win32/winprog64/virtual-address-space>
* <https://learn.microsoft.com/en-us/windows/win32/memory/memory-management>
* <https://www.tutorialspoint.com/operating_system/os_memory_management.htm>#
* [随机存取存储器 (RAM) 如何影响性能 |戴尔美国](https://www.dell.com/support/kbdoc/en-us/000129805/how-random-access-memory-ram-affects-performance)
* [Windows 10 中的物理和虚拟内存 - Microsoft Community](https://answers.microsoft.com/en-us/windows/forum/all/physical-and-virtual-memory-in-windows-10/e36fb5bc-9ac8-49af-951c-e7d39b979938)
* [页面状态 - Win32 应用 |微软学习](https://learn.microsoft.com/en-us/windows/win32/Memory/page-state)
* [MEMORY\_BASIC\_INFORMATION (winnt.h) - Win32 应用程序 |微软学习](https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-memory_basic_information)