北京理工大學

操作系统课程设计

实验四、进程控制	Experiment 4, Memory Monitoring

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
班级：07111705

Table of Contents

Purpose

Experiment for Windows:

Windows designs a memory monitor that requires:

Real-time display of the memory usage in the current system, including the layout of the system address space and the usage of physical memory;

Real-time display of the virtual address space layout and working set information of the experiment 2 process control (ParentProcess.exe)

Related syscalls:

GetSystemInfo, VirtualQueryEx, VirtualAlloc, GetPerformanceInfo, GlobalMemoryStatusEx...

Linux experiment:

Use the top command to view the system, subcommands P, T, M

Use ps -A to view all processes and find the pid of ProcessParent

Use top -p pid to check the status of the ProcessParent program;

Use pmap -d pid to view the memory usage of ProcessParent

Problem Discussion

The experiment requires that the program must display

1. the system's memory usage
    1. the system address space layout
    2. the physical memory usage
2. virtual address space layout
3. parentprocess.exe from lab 2's working information

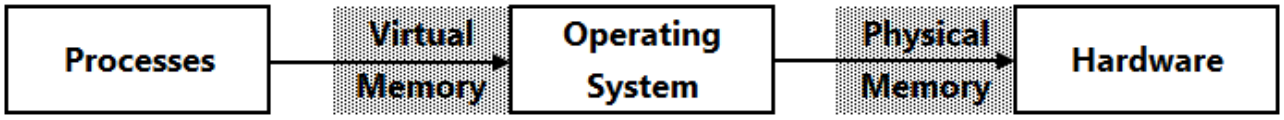
What is system memory?

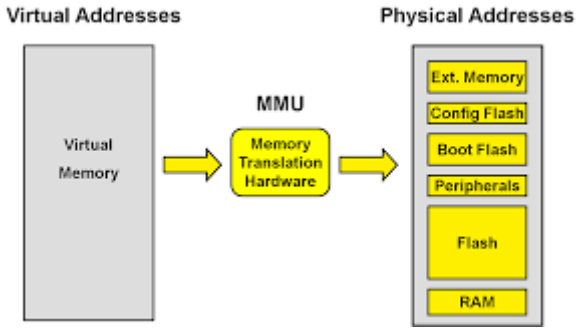
A computers system memory is made up physical memory, called Random Access Memory (RAM), and virtual memory. System memory is not permanent storage, like a hard disk drive that saves its contents when the computer is switched off.

A process is a set of tasks contained in a program, which executes through a sequence of instructions called threads. Processes run on the operating system and the operating system manages the hardware resources for all the running processes.

The operating system provides virtual memory to all processes which run on physical memory. The virtual memory manager of the operating system applies a method called

Paging to map the virtual address space to the physical address space, in such a way that all processes can get to run on the physical memory.





Execution [Windows]

GetSystemInfo

GetSystemInfo() retrieves information about the current system. It accepts a pointer to a **SYSTEM_INFO** structure that receives the information.

The **SYTEM_INFO** structure contains information about the current computer system.

- wProcessorArchitecture: The processor architecture of the installed operating system.
- DwPageSize: The page size and the granularity of page protection and commitment. This is the page size used by the **VirtualAlloc** function.
- lpMinimumApplicationAddress: A pointer to the lowest memory address accessible to applications and dynamic-link libraries (DLLs).
- lpMaximumApplicationAddress: A pointer to the highest memory address accessible to applications and DLLs.
- dwAllocationGranularity: The granularity for the starting address at which virtual memory can be allocated.

VirtualQueryEx

The **VirtualQueryEx** retrieves information about a range of consecutive pages within the virtual address space of specified processes. The return value is the actual number of bytes returned in the information buffer. The possible states of all the pages include MEM_COMMIT, MEM_RESERVE, MEM_FREE, MEM_PRIVATE, MEM_MAPPED or

MEM_IMAGE. If the function fails, the return value is zero. It takes the following parameters:

- HANDLE hProcess: A handle to the process whose memory information is queried.
- LPCVOID lpAddress: A pointer to the base address of the region of pages to be queried. The **GetSystemInfo** function is used to determine the size of a page on the host computer.
- PMEMORY_BASIC_INFORMATION lpBuffer: A pointer to a **MEMORY_BASIC_INFORMATION** structure in which information about the specified page range is returned.
- SIZE_T dwLength: The size of the buffer pointed to by the lpBuffer parameter in bytes.

The **MEMORY_BASIC_INFORMATION** structure contains information about a range of pages in the virtual address space of a process. Its members include:

- PVOID BaseAddress: A point to the base address of the region of pages
- PVOID AllocationBase: A pointer to the base address of a range of pages allocated by the **VirtualAlloc** function. The page pointed to by the BaseAddress member isb contained within this allocation range.
- DWORD AllocationProtect: The memory protection option when the region was initially allocated.
- WORD PartitionId
- SIZE_T RegionSize: The size of the region beginning at the base address in which all pages have identical attributes in bytes
- DWORD State: The state of the pages in the region. States include MEM_COMMIT, MEM_FREE and MEM_RESERVE.
- DWORD Protect: The access protection of the pages in the region
- DWORD Type: The type of pages in the region which include MEM_IMAGE, MEM_MAPPED and MEM_PRIVATE.

VirtualAlloc

The **VirtualAlloc** function reserves, commits, or changes the state of a region of pages in the virtual address space of the calling process. Memory allocated by this function is automatically initialized to zero. If the function succeeds, the return value is the base address of the allocated region of pages.

Each page has an associated page state(Free, Reserved or Committed). The **VirtualAlloc** function can perform the following operations:

- Commit a region of reserved pages
- Reserve a region of free pages
- Simultaneously reserve and commit a region of free pages

It takes 4 arguments.

- LPVOID LpAddress: the starting address of the region to allocate.
- SIZE_T dwSize: the size of the region in bytes
- DWORD flAllocationType: The type of memory allocation. This parameter must contain one of the following values: MEM_COMMIT, MEM_RESERVE, MEM_RESET, MEM_RESET_UNDO
- DWARD flProtect: the memory protection for the region of pages to be allocated.

Example of how to use this function: Reserving and Committing Memory - Win32 apps | Microsoft Learn

GetPerformanceInfo

The **GetPerfomanceInfo** function reserves the performance values contained in the **PERFORMANCE_INFORMATION** structure. If the function succeeds, the return value is TRUE. If the function fails the return value is FALSE. It takes the following argument:

- PPERFORMANCE_INFORMATION pPerformanceInformation: A pointer to a **PERFORMANCE_INFORMATION** structure that receives the performance information.
- DWORD cb: The size of the **PERFORMANCE_INFORMATION** structure, in bytes.

The **PERFORMANCE_INFORMATION** structure contains performance information. Its members include:

- DWORD cb: the size of this structure in bytes
- SIZE_T CommitTotal: the number of pages currently committed by the system.
- SIZE_T CommitLimit: The current maximum number of pages that can be committed by the system without extending the paging file(s).
- SIZE_T CommitPeak; The maximum number of pages that were simultaneously in the committed state since the last system reboot.
- SIZE_T PhysicalTotal: The amount of actual physical memory, in pages.
- SIZE_T PhysicalAvailable: The amount of physical memory currently available, in pages.
- SIZE_T SystemCache: The amount of system cache memory, in pages. This is the size of the standby list plus the system working set.
- SIZE_T KernelTotal: The sum of the memory currently in the paged and nonpaged kernel pools, in pages.
- SIZE_T KernelPaged: The memory currently in the paged kernel pool, in pages.
- SIZE_T KernelNonpaged: The memory currently in the nonpaged kernel pool, in pages.
- SIZE_T PageSize: The size of a page, in bytes.

- DWORD HandleCount:
- The current number of open handles.
- DWORD ProcessCount: The current number of processes.
- DWORD ThreadCount: The current number of threads.

GlobalMemoryStatusEx

The **GlobalMemoryStatusEx** function retrieves information about the systems current usage of both physical and virtual memory. If the function succeeds the return value is nonzero. It accepts one argument:

- LPMEMORYSTATUSEX lpBuffer: A pointer to a **MEMORYSTATUSEX** structure that receives information about memory availability.

The **MEMORYSTATUSEX** structure contains information about the current state of both physical and virtual memory, including extended memory. Its members include:

- DWORD dwLength: The size of the structure in bytes. This member must be set before calling **GlobalMemoryStatusEx**
- DWORD dwMemoryLoad: A number between 0 and 100 that specifies the approcimate percentage of physical memory that is in use (0 indicates no memory use and 100 indicates full memory use).
- DWORDLONG ullTotalPhys: The amount of actual physical memory in bytes.
- DWORDLONG ullAvailPhys: The amount of physical memory currently available, in bytes. It is the sum of the size of the standby, free, and zero lists.
- DWORDLONG ullTotalPageFile: The current committed memory limit for the system or the current process, whichever is smaller, in bytes. To get the system-wide committed memory limit, call **GetPerformanceInfo**
- DWORDLONG ullAvailPageFile: The maximum amount of memory the current process can commit, in bytes. This value is equal to or smaller than the system-wide available commit value. To calculate the system-wide available commit value, call **GetPerformanceInfo** and subtract the value of **CommitTotal** from the value of **CommitLimit**.
- DWORDLONG ullTotalVirtual: The size of the user-mode portion of the virtual address space of the calling process, in bytes
- DWORDLONG ullAvailVirtual: The amount of unreserved and uncommitted memory currently in the user-mode portion of the virtual address space of the calling process, in bytes.
- DWORDLONG ullAvailExtendedVirtual: Reserved. This value is always 0.

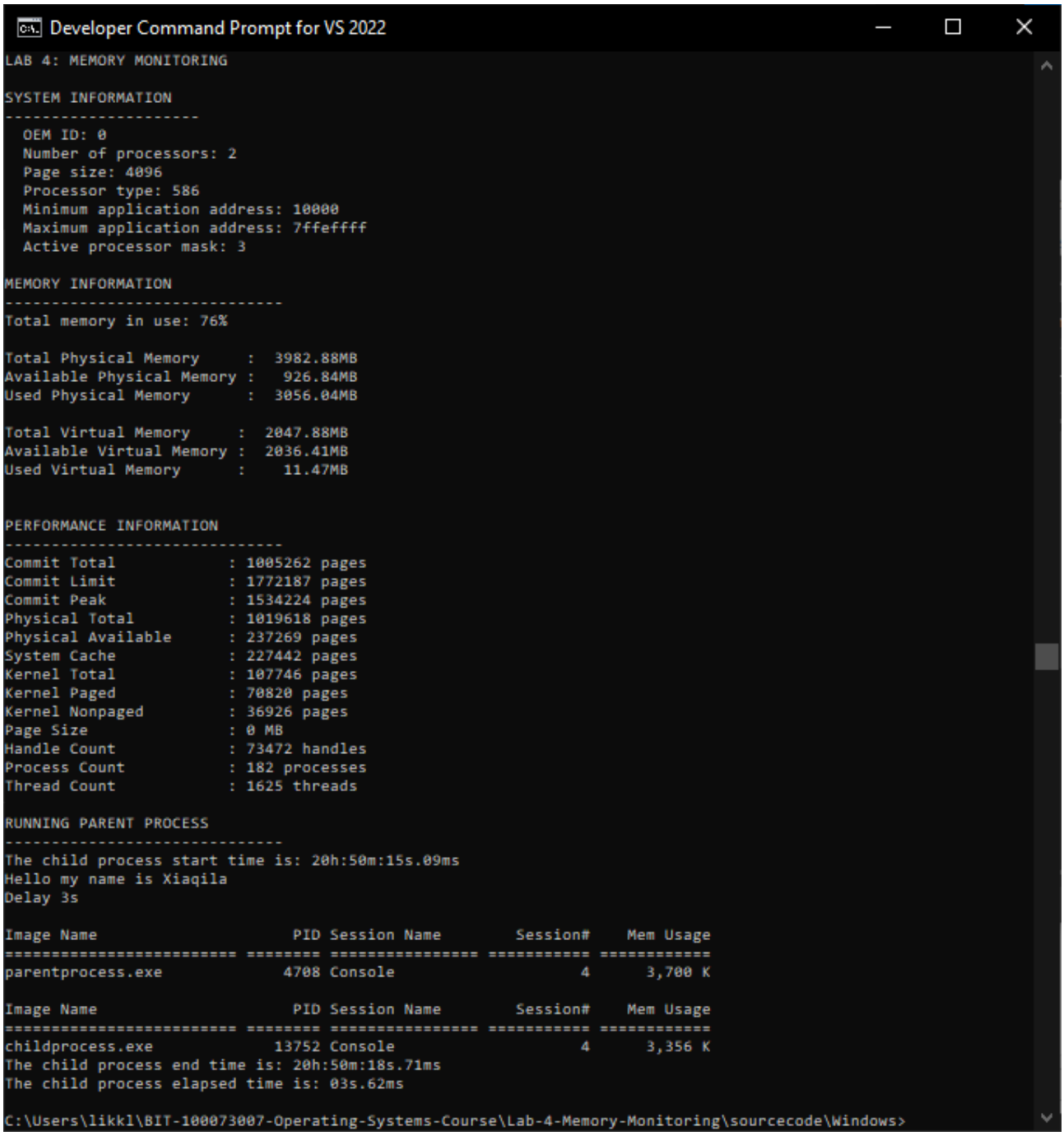
```
#include <windows.h>
#include <stdio.h>
#include <tchar.h>
#include <process.h>
#include <Psapi.h>
#pragma comment(lib, "user32.lib")

// Use to convert bytes to KB
#define DIV 1024

// Specify the width of the field in which to print the numbers.
// The asterisk in the format specifier "%*I64d" takes an integer
// argument and uses it to pad and right justify the number.
#define WIDTH 7

void PrintSystemInfo()
{
   printf("\nSYSTEM INFORMATION");
   printf("\n---------------------\n");

   SYSTEM_INFO siSysInfo;

   // Copy the hardware information to the SYSTEM_INFO structure.

   GetSystemInfo(&siSysInfo);

   // Display the contents of the SYSTEM_INFO structure.

   printf("  OEM ID: %u\n", siSysInfo.dwOemId);
   printf("  Number of processors: %u\n",
          siSysInfo.dwNumberOfProcessors);
   printf("  Page size: %u\n", siSysInfo.dwPageSize);
   printf("  Processor type: %u\n", siSysInfo.dwProcessorType);
   printf("  Minimum application address: %lx\n",
          siSysInfo.lpMinimumApplicationAddress);
   printf("  Maximum application address: %lx\n",
          siSysInfo.lpMaximumApplicationAddress);
   printf("  Active processor mask: %u\n",
          siSysInfo.dwActiveProcessorMask);
}

void PrintMemoryInfo()
{
   printf("\nMEMORY INFORMATION\n");
   printf("------------------------------\n");

   MEMORYSTATUSEX ms = {sizeof(MEMORYSTATUSEX)};

   ms.dwLength = sizeof(ms);

   // Retrieves information about the system's currenT usage of
physical memory
```

```
   GlobalMemoryStatusEx(&ms);

   printf("Total memory in use: %ld%%\n", ms.dwMemoryLoad);

   printf("\nTotal Physical Memory     : %8.2I64fMB \nAvailable
Physical Memory : %8.2I64fMB \nUsed Physical Memory      :
%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("Total Virtual Memory     : %8.2I64fMB \nAvailable
Virtual Memory : %8.2I64fMB \nUsed Virtual Memory      :
%8.2I64fMB \n\n", ms.ullTotalVirtual / (1024 * 1024.0),
ms.ullAvailVirtual / (1024 * 1024.0), ms.ullTotalVirtual / (1024 *
1024.0) - ms.ullAvailVirtual / (1024 * 1024.0));
}

void PrintPerformanceInfo()
{
   printf("\nPERFORMANCE INFORMATION\n");
   printf("---------------------------\n");

   PERFORMANCE_INFORMATION siPerfInfo;

   // Copy the hardware information to the SYSTEM_INFO structure.

   GetPerformanceInfo(&siPerfInfo, siPerfInfo.cb);

   printf("Commit Total\t\t: %d pages\n", siPerfInfo.CommitTotal);
   printf("Commit Limit\t\t: %d pages\n", siPerfInfo.CommitLimit);
   printf("Commit Peak\t\t: %d pages\n", siPerfInfo.CommitPeak);
   printf("Physical Total\t\t: %d pages\n",
siPerfInfo.PhysicalTotal);
   printf("Physical Available\t: %d pages\n",
siPerfInfo.PhysicalAvailable);
   printf("System Cache\t\t: %d pages\n", siPerfInfo.SystemCache);
   printf("Kernel Total\t\t: %d pages\n", siPerfInfo.KernelTotal);
   printf("Kernel Paged\t\t: %d pages\n", siPerfInfo.KernelPaged);
   printf("Kernel Nonpaged\t\t: %d pages\n",
siPerfInfo.KernelNonpaged);
   printf("Page Size\t\t: %d MB\n", siPerfInfo.PageSize / (1024 *
1024.0));
   printf("Handle Count\t\t: %d handles\n",
siPerfInfo.HandleCount);
   printf("Process Count\t\t: %d processes\n",
siPerfInfo.ProcessCount);
   printf("Thread Count\t\t: %d threads\n",
siPerfInfo.ThreadCount);
}

void PrintRunningProcesses( void )
{
   printf("\nCURRENTLY RUNNING PROCESSES\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("LAB 4: MEMORY MONITORING\n");

   PrintSystemInfo();
   PrintMemoryInfo();
   PrintPerformanceInfo();

   STARTUPINFO si;
   PROCESS_INFORMATION pi;
   ZeroMemory(&si, sizeof(si));
   si.cb = sizeof(si);
   ZeroMemory(&pi, sizeof(pi));
   if (argc != 3) /* argc should be 2 for correct execution */
   {
      printf("Usage in memorymonitoring: %s [cmdline]\n",
argv[0]);
      return;
   }
   // Start the child process.
   if (!CreateProcess(NULL,    // No module name (use command
line)
                      argv[1], // Command line
                      NULL,    // Process handle not inheritable
                      NULL,    // Thread handle not inheritable
                      FALSE,   // Set handle inheritance to FALSE
                      0,       // No creation flags
                      NULL,    // Use parent's environment block
                      NULL,    // Use parent's starting directory
                      &si,     // Pointer to STARTUPINFO structure
                      &pi)     // Pointer to PROCESS_INFORMATION
structure
   )
   {
      printf("CreateProcess failed (%d).\n", GetLastError());
      return;
   }

   PrintRunningProcesses();

   // Just ask tasklist command
```

```
    Sleep(5000);

    return 0;
}
```

Results and Analysis [Windows]

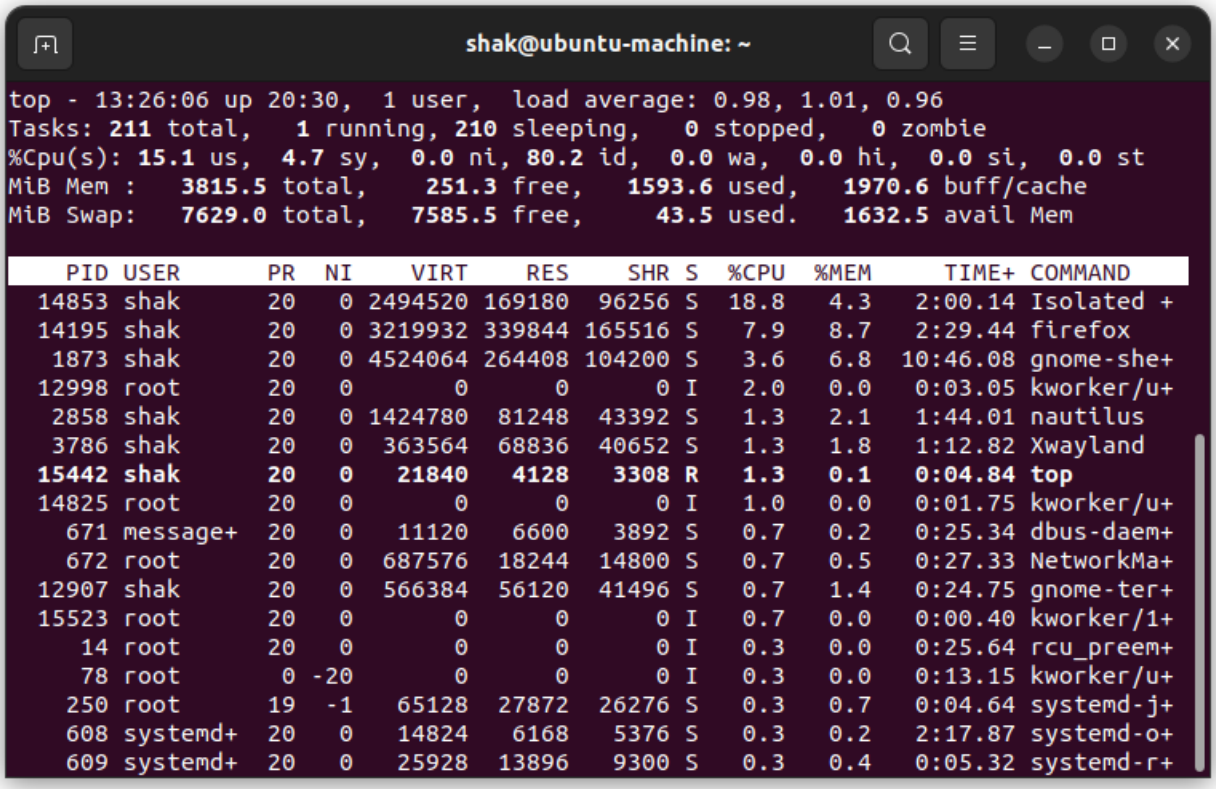


Execution [Linux]

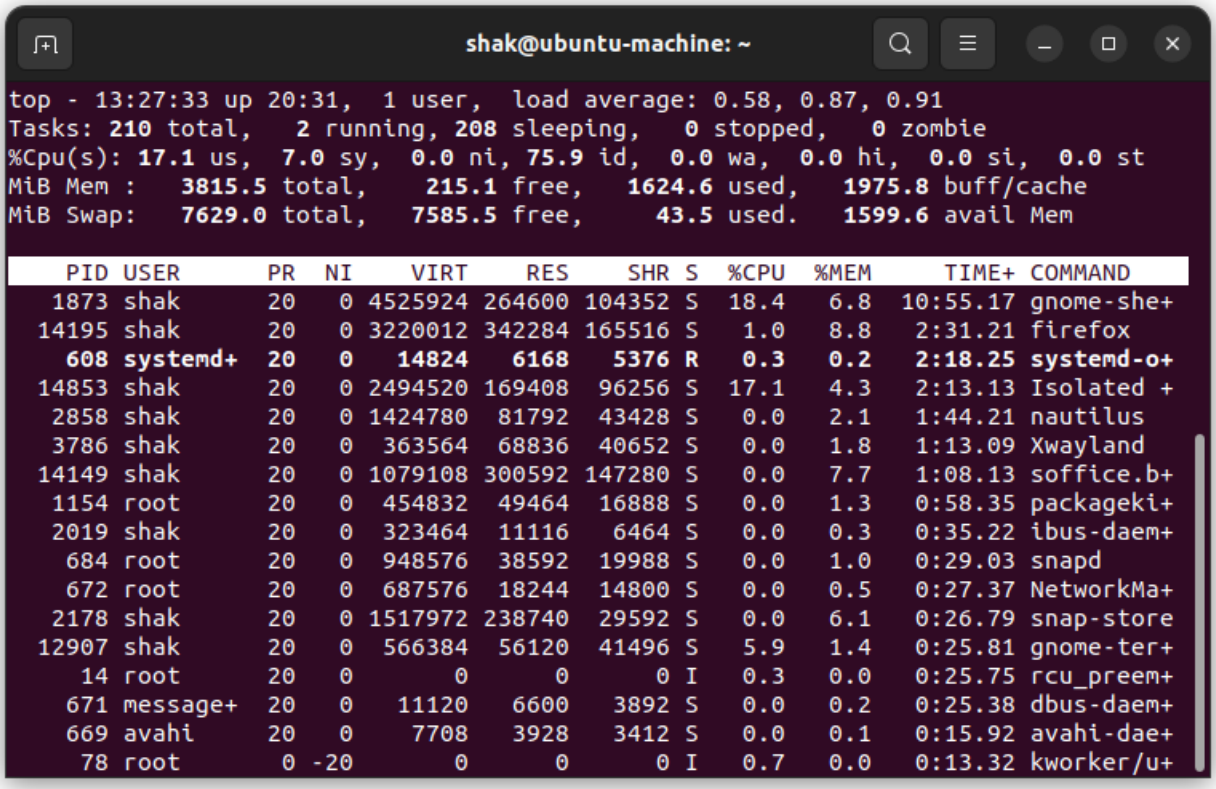
The top (table of processes) command shows a real-time view of running process in Linux. By default it sorts the process list by the %CPU column. The following commands can be used to sort using a different column:

- P. sort by the %CPU column
- T. sort by the TIME+ column
- M. sort by the %MEM column

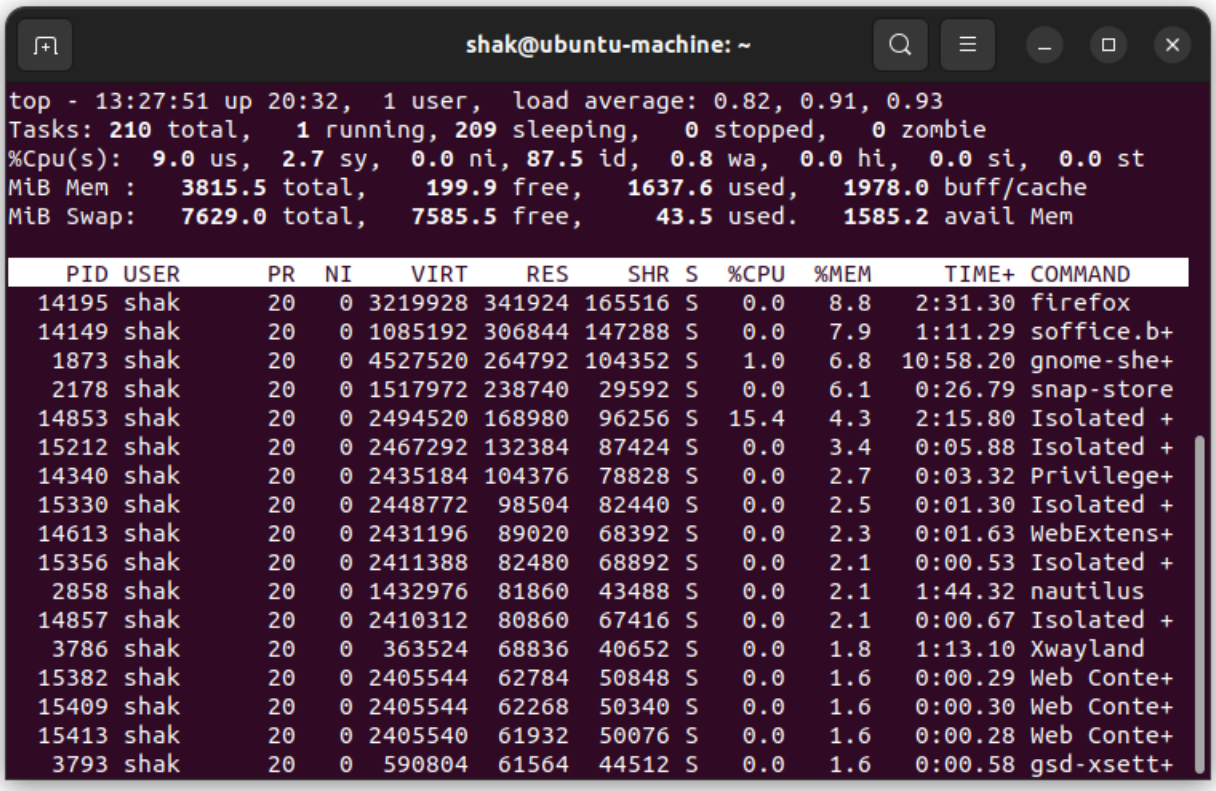
top SORTED BY %CPU USING P SUBCOMMAND



top SORTED BY TIME+ USING T SUBCOMMAND

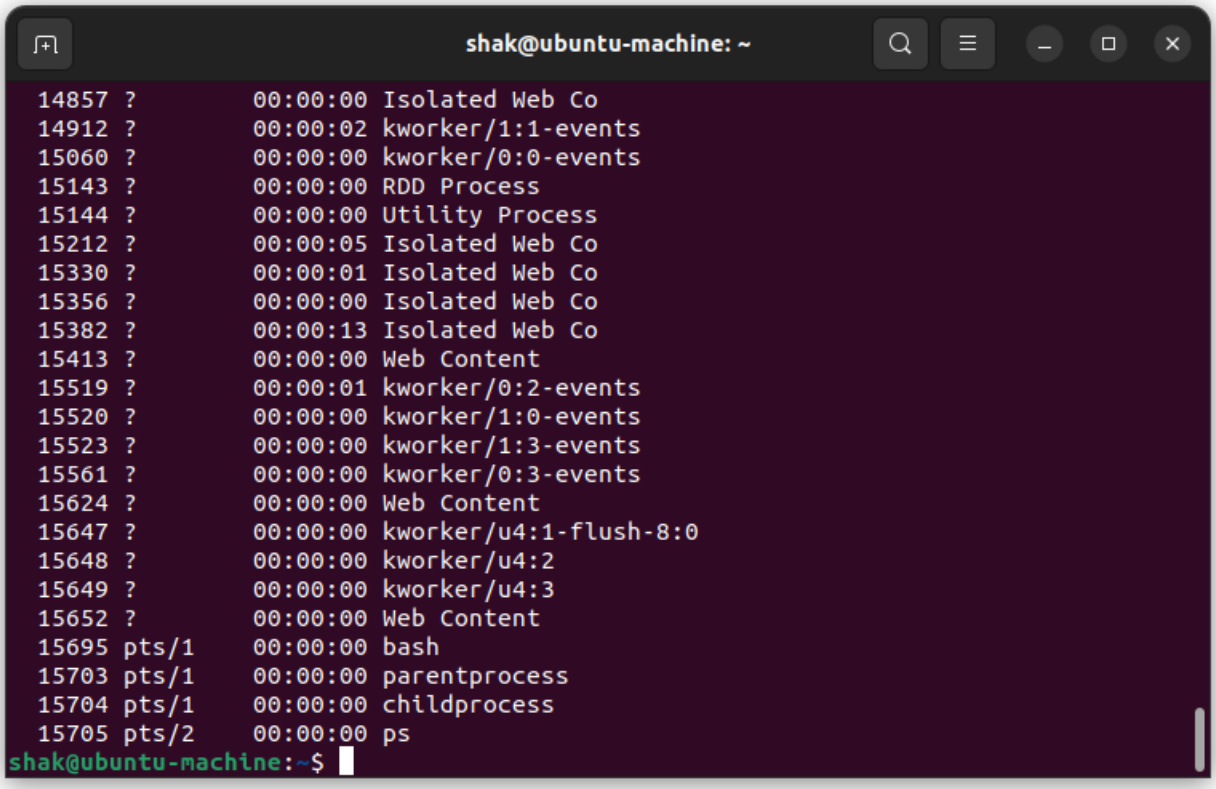


top SORTED BY %MEM USING M SUBCOMMAND

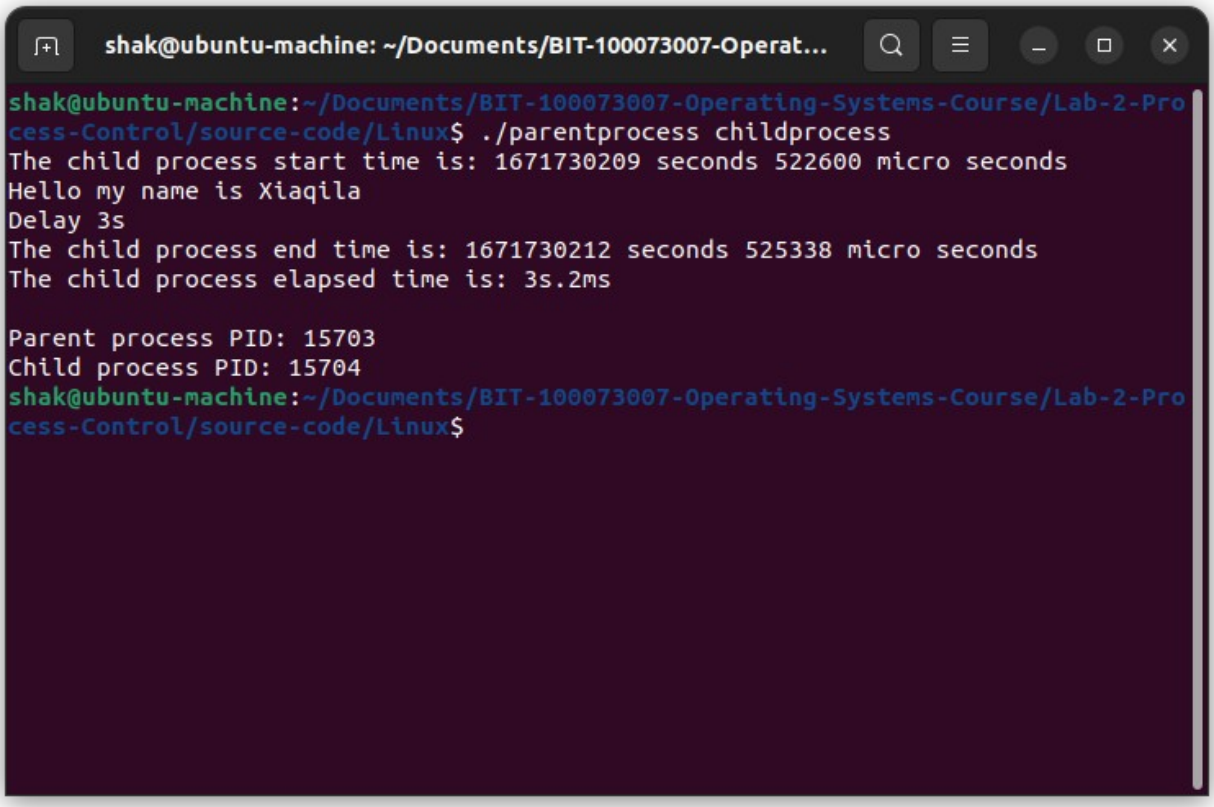


USING ps -A TO FIND THE PID OF PARENTPROCESS

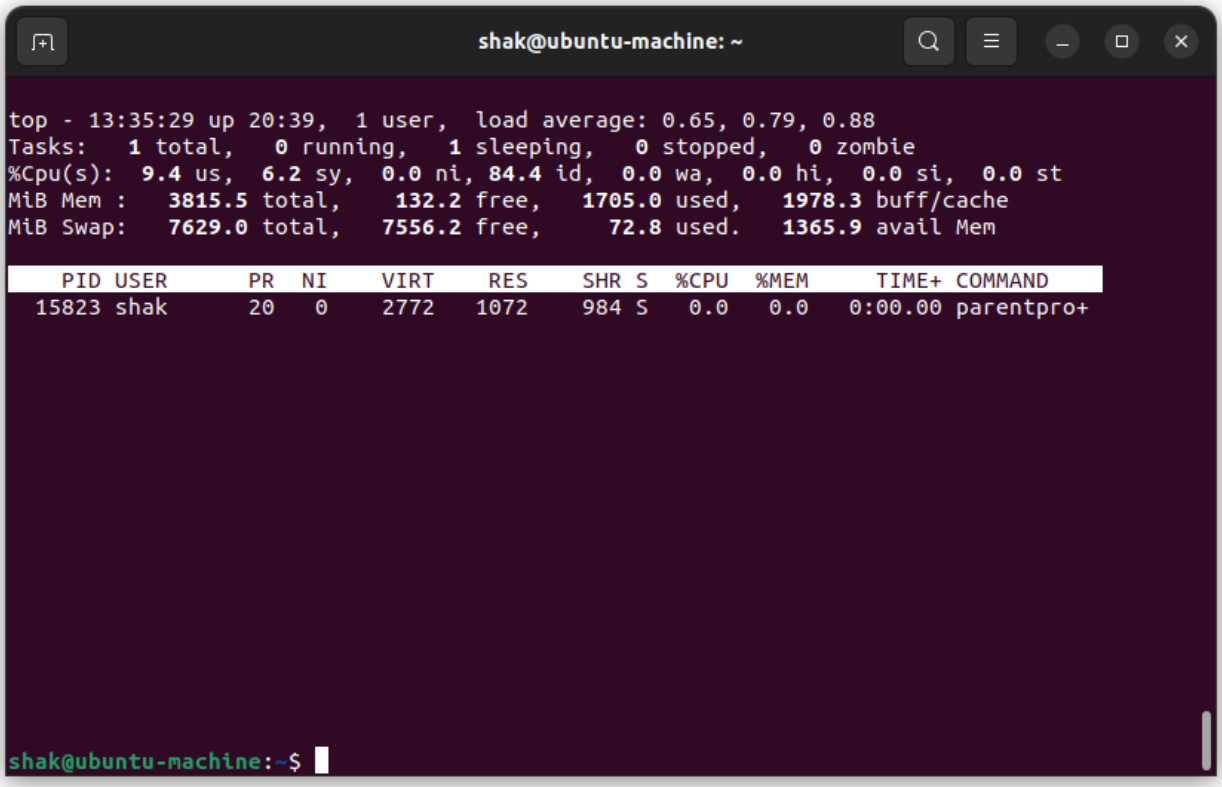
The *ps command* allows you to list the status of processes running on your system easily.



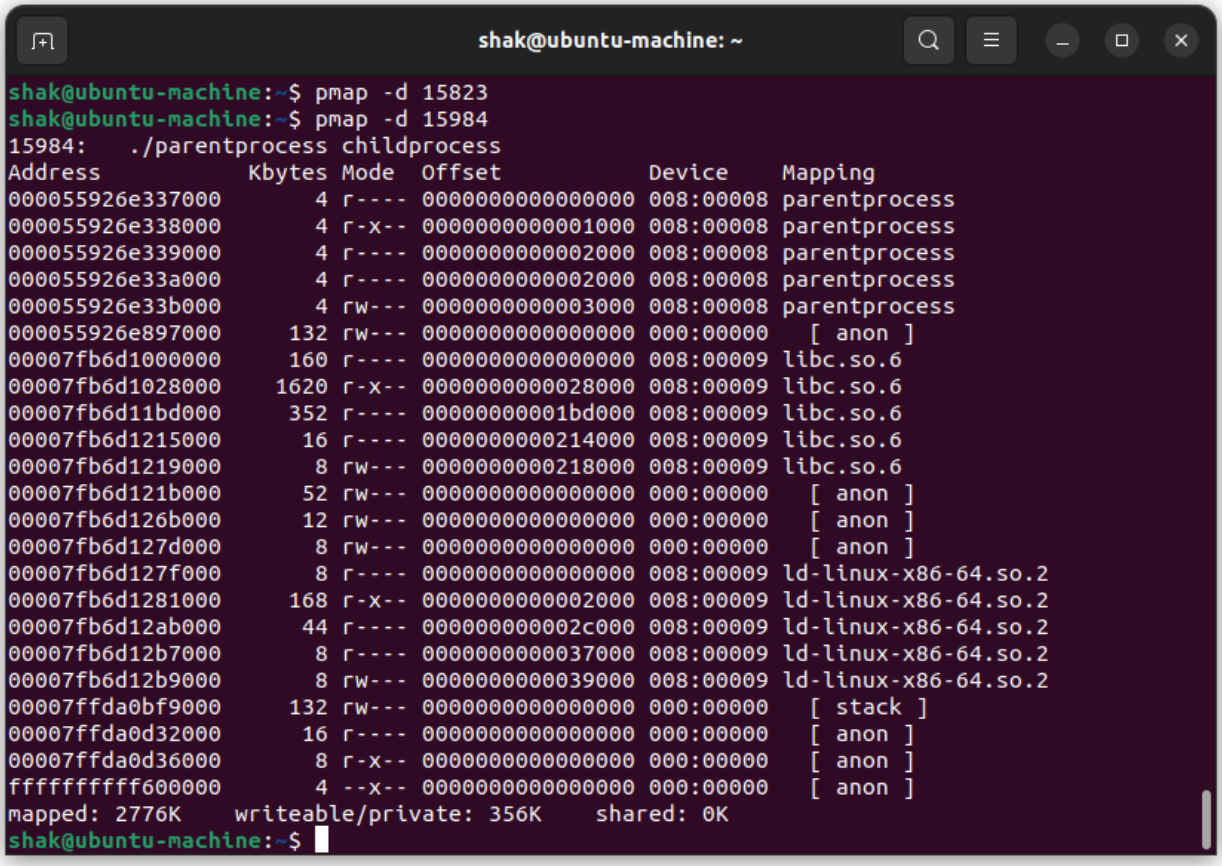
pid of parentprocess – 15703. which is confirmed in the output of the parentprocess command



USING top -p pid TO CHECK THE STATUS OF PARENTPROCESS



Use pmap -d pid to view the memory usage of ProcessParent
The *pmap command in Linux* is used to display the memory map of a process.



Reference:

- 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#
- How Random Access Memory (RAM) affects performance | Dell US
- Physical and Virtual Memory in Windows 10 - Microsoft Community
- Page State - Win32 apps | Microsoft Learn
- MEMORY_BASIC_INFORMATION (winnt.h) - Win32 apps | Microsoft Learn