

Санкт-Петербургский государственный политехнический университет  
Институт Информационных Технологий и Управления  
Кафедра компьютерных систем и программных технологий

---

Отчёт по практической работе  
по предмету «Системное программное обеспечение»

**УТИЛИТА СВОРА СИСТЕМНОЙ ИНФОРМАЦИИ В ОС WINDOWS**

Работу выполнил студент гр. 53501/3 \_\_\_\_\_ Мартынов С. А.

Работу принял преподаватель \_\_\_\_\_ Душутина Е. В.

Санкт-Петербург  
2015

# Оглавление

<b>Постановка задачи</b>	<b>3</b>
<b>Введение</b>	<b>4</b>
<b>Класс системной информации</b>	<b>8</b>
Метод GetUserTime() . . . . .	8
Метод GetUTCTime() . . . . .	8
Метод GetFUserName() . . . . .	8
Метод GetHostname() . . . . .	9
Метод GetCPUVendor() . . . . .	9
Метод GetCPUNumber() . . . . .	9
Метод GetVolumesInformation() . . . . .	9
Метод GetTotalMemory() . . . . .	10
Метод GetFreeMemory() . . . . .	10
Метод GetPagefileMemory() . . . . .	10
Метод GetVideoInformation() . . . . .	10
Метод GetWindowsVersion() . . . . .	10
Метод GetWindowsBuild() . . . . .	11
Метод GetWindowsRole() . . . . .	11
Метод GetConnectionInformation() . . . . .	11
Метод GetUptimeInformation() . . . . .	11
Метод GetConnectedHardwareList() . . . . .	12
Листинг . . . . .	12
<b>Демонстрация работы программы</b>	<b>25</b>
Эксперимент 1. Виртуальная Win7 . . . . .	25
Эксперимент 2. Реальная Win7 . . . . .	25
<b>Заключение</b>	<b>28</b>

# Постановка задачи

В рамках данной работы необходимо ознакомиться с основными механизмами сбора системной информации в ОС Windows.

Необходимо рассмотреть имеющихся штатных и нештатных механизмов извлечения системной информации Windows.

В процессе работы предполагается изучить источники получения системной информации в Windows и разработать консольную утилиту, отображающую на экран (или в лог-файл) всю собранную информацию. Данная информация оказывается полезной, когда продукт уже передан на эксплуатацию конечному пользователю, и у него возникают проблемы, а разработчик не может получить физического доступа к машине, на которой выполняется код.

В конце сравнить результаты полученные разработанной утилитой с результатами других средств. Провести эксперимент на нескольких устройствах.

Полные исходные коды сделать доступными по адресу [https://github.com/SemenMartynov/SPbPU\\_SystemProgramming](https://github.com/SemenMartynov/SPbPU_SystemProgramming).

# Введение

Основные характеристики работы системы можно получить в диспетчере задач (рис. 1). Там находится информация о количестве доступных системе ядер и их загрузка. Есть информация по доступной и занятой памяти, а также о файле подкачки. Показано общее время работы системы.

Другие полезные данные можно почерпнуть из настроек сетевого подключения (рис. 2) и в оснастке управления дисками (рис. 3), где помимо самих томов можно увидеть файловую систему, объём и букву, под которой том смонтирован в систему.

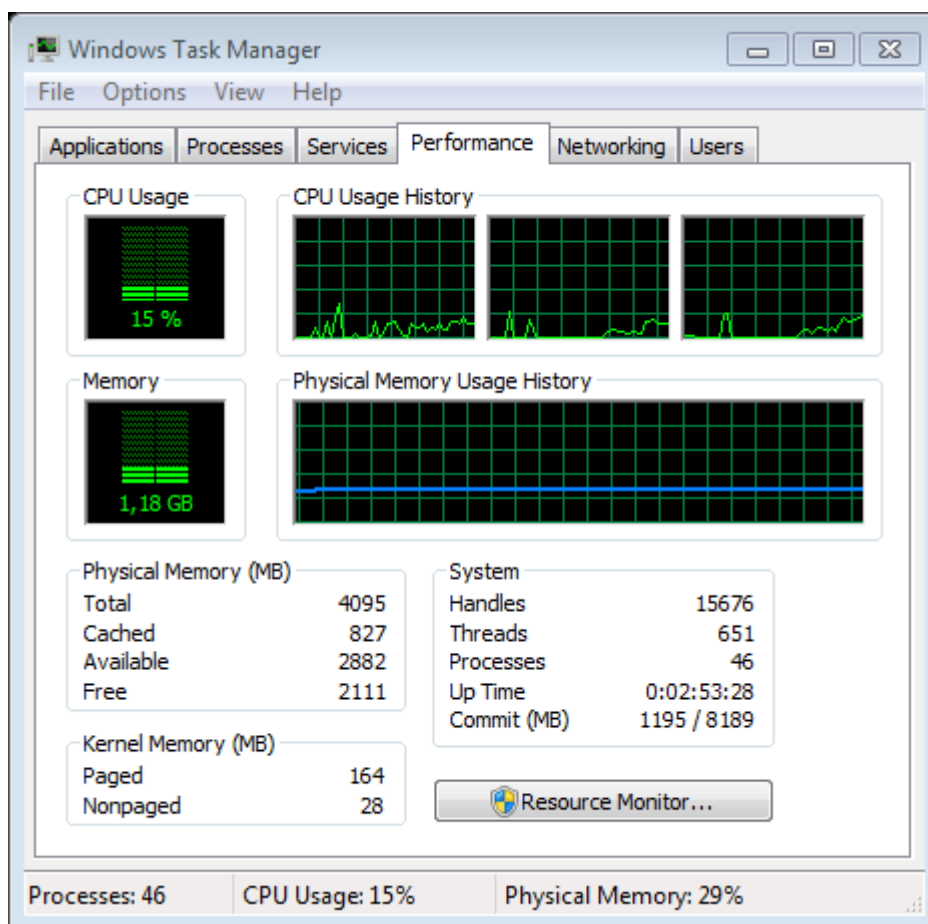


Рис. 1: Диспетчер задач

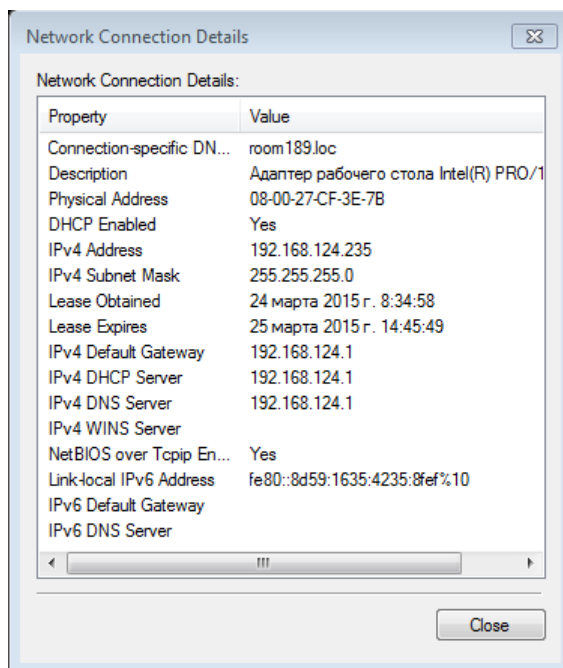


Рис. 2: Информация о сетевом подключении

Конфигуратор сетевых подключений даёт всю возможную статичную информацию о подключении.

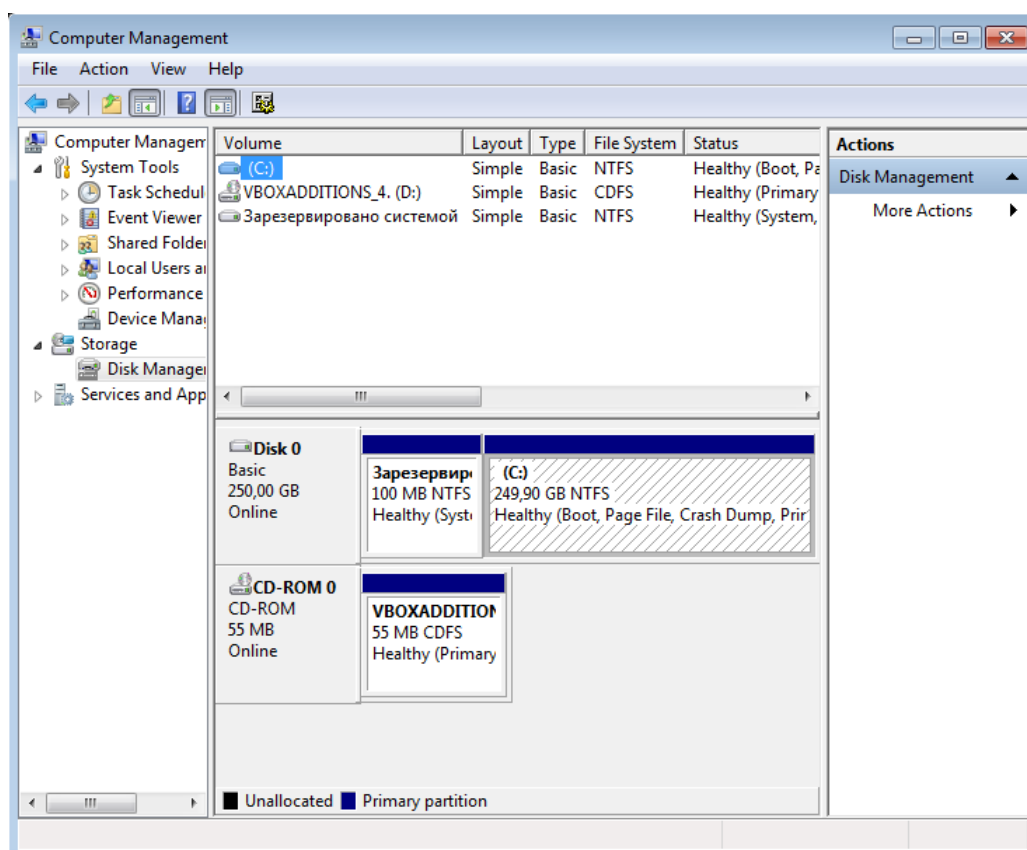


Рис. 3: Оснастка управления дисками

Более комплексным решением является системная утилита msinfo32 (рис 4). Она собирает и отображает данные о конфигурации системы как для локальных, так и для удаленных компьютеров. Сюда входит информация о конфигурации оборудования, компонентах компьютера, а также программном обеспечении, в том числе о подписанных и неподписанных драйверах. При устранении неполадок, связанных с конфигурацией системы, сотрудникам службы технической поддержки необходимы определенные данные о компьютере.

Для хранения данных о системе предназначены файлы с расширением .nfo. Кроме того, программа работает с файлами форматов .cab и .xml. Содержимое открытого файла .cab можно просматривать средствами системы.

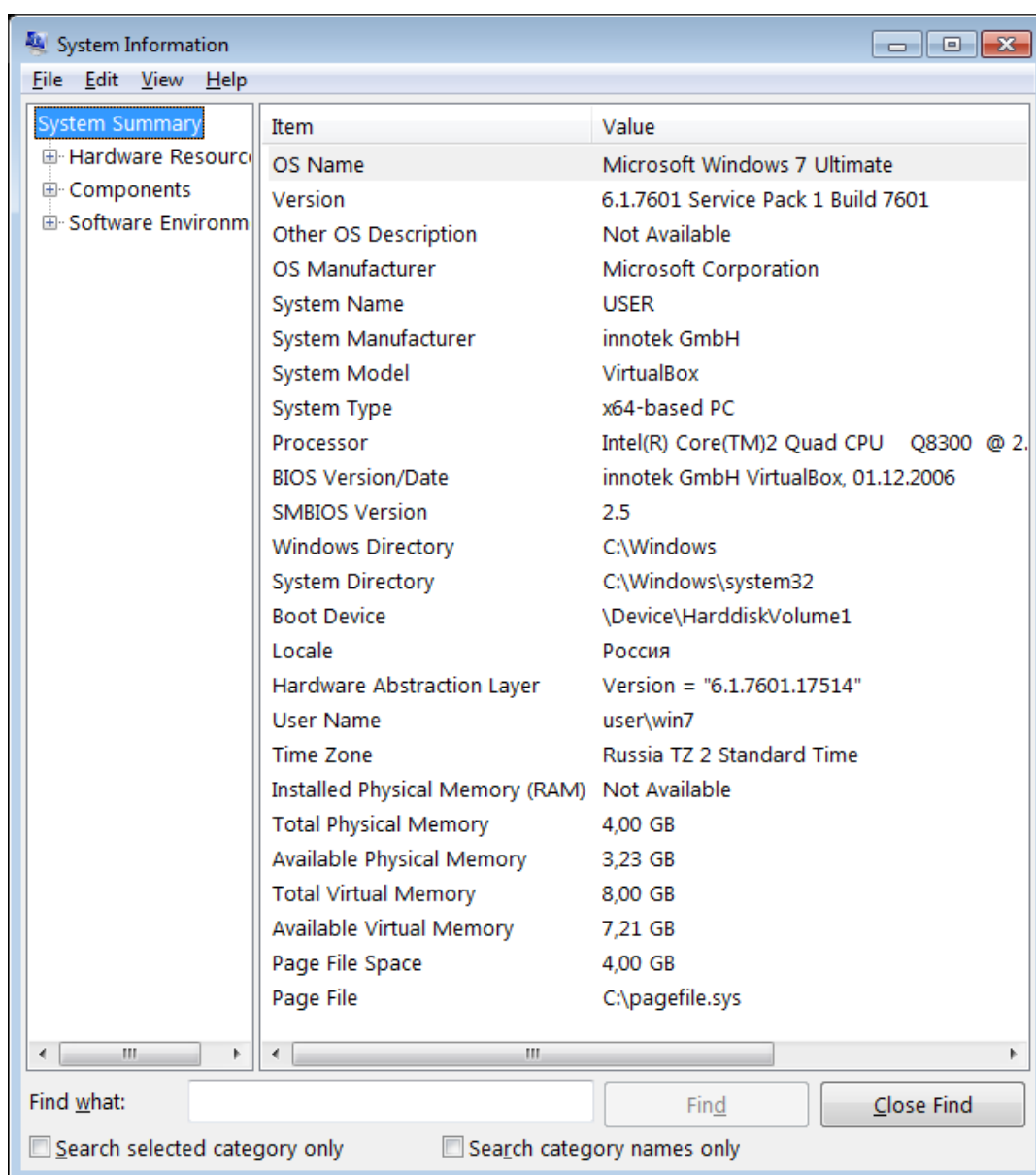


Рис. 4: штатная утилита msinfo32

Среди нештатных и бесплатных средств сборки системной информации наибольшей популярностью пользуются AIDA32, Everest Home и HWiNFO (рис. 5).

HWiNFO предоставляет детальную информацию об оборудовании в ОС Windows. Также существует портативная версия. При запуске HWiNFO открывает несколько окон: окно состояния процессоров, окно сводной информации о системе и самое большое окно с доступом к оборудованию. Сводная информация дает возможность быстро ознакомиться с конфигурацией основного оборудования компьютера (процессор, память, диски, видеокарта, оперативная память). Всё установленное оборудование поделено по типу принадлежности (память, процессоры и т.д.) и показано в виде дерева.

HWiNFO включает в себя монитор системы в режиме реального времени, что позволит увидеть реальные показания датчиков оборудования на текущий момент времени. Программа включает в себя специальные тесты производительности системы. Вся информация, которую представляет программа, может быть сохранена в различных форматах: текст, Html, Xml, Mhtml.

Стоит заметить, что программа сходу не правильно определила имеющийся в наличии процессор.

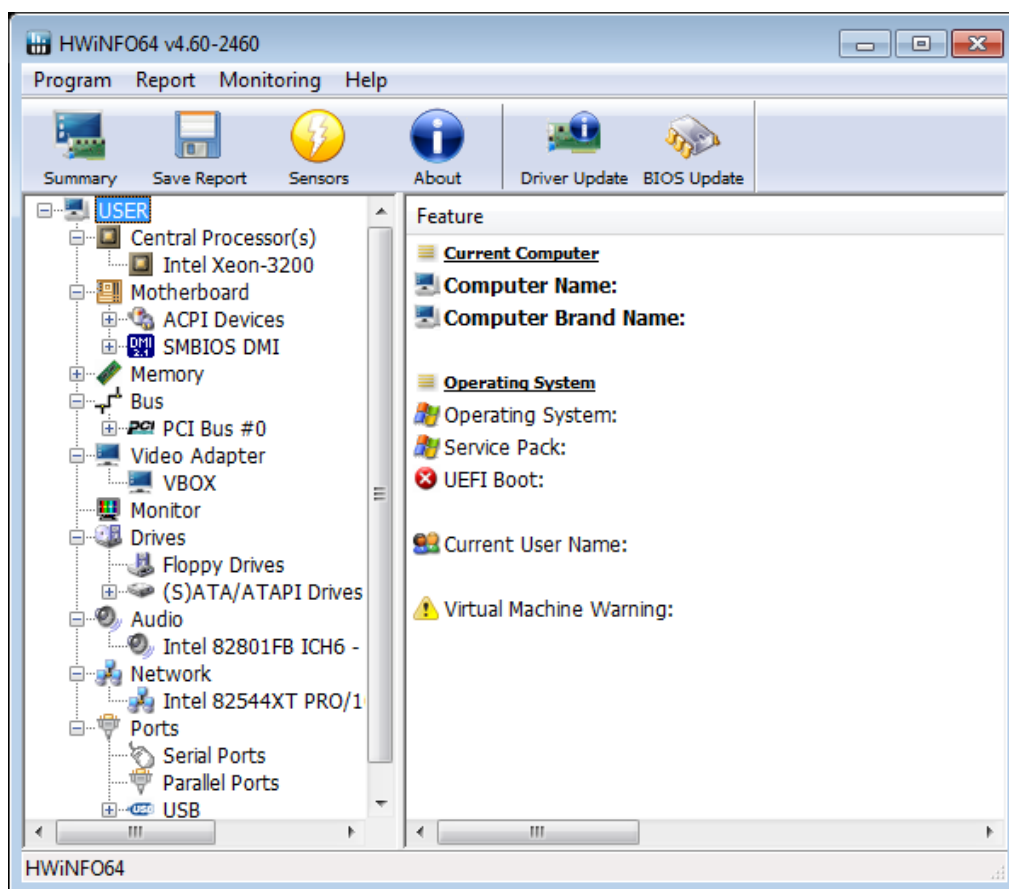


Рис. 5: Запуск утилиты HWiNFO

# Класс системной информации

Для сбора системной информации был разработан класс `MySystem`, методы которого отвечают за сбор различной системной информации. Ниже рассмотрены эти методы и предоставлен листинг их реализации.

## Метод `GetUserTime()`

Возвращает пользовательское время, т.е. время, локальное для пользователя (с учётом часового пояса).

Источник информации – системная структура `_SYSTEMTIME`.

## Метод `GetUTCTime()`

Возвращает мировое (UTC) время. Не зависит от локальных настроек пользователя.

Источник информации – системная структура `_SYSTEMTIME`.

## Метод `GetFUserName()`

Возвращает полное имя пользователя, с учётом имени домена.

Источник информации – системный вызов `GetComputerNameEx()` из `sysinfoapi.h`.



## Метод GetHostname()

Возвращает имя хоста. Это не полное доменное имя, но это имя может использоваться для доступа по сети в рамках одного широковебательного домена.

Источник информации – системный вызов GetComputerNameEx() из sysinfoapi.h.

## Метод GetCPUVendor()

Возвращает название производителя процессора (если это возможно; если нет вернёт пустую строку). Для работы используется ассемблерный код т.к. информация получается непосредственно из регистров процессора.

Источник информации – регистры центрального процессора.

## Метод GetCPUNumber()

Возвращает количество доступных ядер. Если на машине включена поддержка технологии Intel hyper-threading technology (или аналогичная технология виртуализации ядер), возвращаемое значение будет соответствовать количеству ядер, которое доступно ядре операционной системы.

Источник информации – системная структура \_SYSTEM\_INFO.

## Метод GetVolumesInformation()

Возвращает информацию о логических разделах, используемых в системе. По каждому разделу выводится его путь (как правило, заглавная буква латинского алфавита), метка (если она установлена), серийный номер и используемая файловая система (если она известна ядру операционной системы).

Источник информации – системный вызов GetVolumeInformationW() из FileApi.h.

## Метод GetTotalMemory()

Возвращает (в гигабайтах) общий объём физической оперативной памяти без файла подкачки.

Источник информации – системный класс \_MEMORYSTATUSEX.

## Метод GetFreeMemory()

Возвращает (в гигабайтах) общий объём свободной физической оперативной памяти без файла подкачки.

Источник информации – системный класс \_MEMORYSTATUSEX.

## Метод GetPagefileMemory()

Возвращает (в гигабайтах) общий объём системного файла подкачки.

Источник информации – системный класс \_MEMORYSTATUSEX.

## Метод GetVideoInformation()

Возвращает подробную информацию по видеосистеме. В начале формируется список всех видеоадаптеров (видеокарт), а потом список мониторов, подключённых к каждому из них.

По видеоадаптерам выводится имя производителя (если эта информация есть в системном реестре) и системный путь. По мониторам выводится имя производителя (если эта информация есть в системном реестре), системный путь, разрешение (количество пикселей по горизонтали и по вертикали) и частота обновления.

Источник информации – системный реестр.

## Метод GetWindowsVersion()

Возвращает предполагаемую версию операционной системы (с точностью до номера сервиспака) и её внутренний номер. Этот функционал системой поддерживаться довольно

странным образом и не гарантирует точности результата, однако в рамках наших экспериментов ошибок не наблюдалось.

Источник информации – множество функций из VersionHelpers.h.

## Метод GetWindowsBuild()

Возвращает номер сборки операционной системы. Бывает полезен для выявления различий в рамках одной версии операционной системы.

Источник информации – системная структура \_OSVERSIONINFOEXW.

## Метод GetWindowsRole()

Возвращает роль машины. Это может быть Workstation (рабочая станция), Server (сервер) и Domain Controller (контроллер домена).

Источник информации – системная структура \_OSVERSIONINFOEXW.

## Метод GetConnectionInformation()

Выводит подробную информацию по сетевым соединениям. Как и в случае с видеосистемой, вначале формируется список сетевых адаптеров, а потом по каждому из них список подключений.

По сетевому адаптеру выводится его системный путь, имя (если оно задано) и уникальный идентификатор (MAC-адрес). По сетевому подключению выводится IP-адрес, маска сети, шлюз (если указан) и источник получения адреса. Если адрес был получен по DHCP, этот факт будет указан, как и IP-адрес DHCP-сервера, выдавшего клиенту его IP-адрес.

Источник информации – системный вызов GetAdaptersInfo() из iphlpapi.h.

## Метод GetUptimeInformation()

Возвращает время работы системы с момента включения в часах, минутах и секундах.

Источник информации – функция GetTickCount64() из sysinfoapi.h.

## Метод GetConnectedHardwareList()

Возвращает список устройств, записи (драйвера) которых были обнаружены в реестре (для своей работы эта функция использует приватную функцию GetConnectedHardwareList(), но это сделано исключительно для упрощения работы с кодом). Изначально устройства группируются по классу, но могут быть перегруппированы и отфильтрованы в пользовательском коде.

Источник информации – системный реестр.

## Листинг

Листинг 1 содержит код реализации представленных выше функций. Заголовочный файл интереса не представляет и может быть найден по ссылке из постановки задачи.

Листинг 1: Файл реализации методов класса MySystem

```
1 #include "MySystem.h"
2
3 #include <sstream>
4 #include <iostream>
5 #include <Lmcons.h> // UNLEN
6
7 #include <VersionHelpers.h>
8
9 #include <iphlpapi.h>
10
11 MySystem::MySystem() : OneGB(1024 * 1024 * 1024) {
12     // Fix localtime
13     GetLocalTime(&stLocal);
14     // Fix systime
15     GetSystemTime(&stSystem);
16     // System information (CPU number)
17     GetSystemInfo(&sysInfo);
18     // Everything about memory
19     memoryStatus.dwLength = sizeof(MEMORYSTATUSEX);
20     GlobalMemoryStatusEx(&memoryStatus);
21     // Windows version
22     osvInfo.dwOSVersionInfoSize = sizeof(osvInfo);
23     GetVersionEx((OSVERSIONINFO*)&osvInfo);
24 }
25
26 MySystem::~MySystem() {
```

```

27 }
28
29 std::wstring MySystem::GetUserTime(){
30     // format: YYYY-MM-DD, HH:MM:SS.ms
31     std::stringstream ss;
32     ss << stLocal.wYear << "-" << stLocal.wMonth << "-" << stLocal.wDay << " "
        << stLocal.wHour << ":" << stLocal.wMinute << ":" << stLocal.wSecond
        << "." << stLocal.wMilliseconds;
33
34     return ss.str();
35 }
36
37 std::wstring MySystem::GetUTCTime(){
38     // format: YYYY-MM-DD, HH:MM:SS.ms
39     std::stringstream ss;
40     ss << stSystem.wYear << "-" << stSystem.wMonth << "-" << stSystem.wDay <<
        " " << stSystem.wHour << ":" << stSystem.wMinute << ":" << stSystem.
        wSecond << "." << stSystem.wMilliseconds;
41     return ss.str();
42 }
43
44 std::wstring MySystem::GetFUserName(){
45     // Full user's name
46     TCHAR userName[UNLEN + 1];
47     DWORD nULen = UNLEN;
48     GetUserNameEx(NameSamCompatible, userName, &nULen);
49
50     return std::wstring(userName);
51 }
52 std::wstring MySystem::GetHostname(){
53     //Computer name can be long
54     TCHAR scComputerName[MAX_COMPUTERNAME_LENGTH * 2 + 1];
55     DWORD lnNameLength = MAX_COMPUTERNAME_LENGTH * 2;
56     GetComputerNameEx(ComputerNameNetBIOS, scComputerName, &lnNameLength);
57
58     return std::wstring(scComputerName);
59 }
60
61 std::wstring MySystem::GetCPUVendor(){
62     int regs[4] = { 0 };
63     char vendor[13];
64     __cpuid(regs, 0);           // mov eax,0; cpuid
65     memcpy(vendor, &regs[1], 4); // copy EBX
66     memcpy(vendor + 4, &regs[2], 4); // copy ECX
67     memcpy(vendor + 8, &regs[3], 4); // copy EDX

```

```

68     vendor[12] = '\\0';
69
70     std::string tmp(vendor);
71     return std::wstring(tmp.begin(), tmp.end());
72 }
73
74 int MySystem::GetCPUNumber(){
75     return sysInfo.dwNumberOfProcessors;
76 }
77
78 std::wstring MySystem::GetVolumesInformation(){
79     // see http://www.codeproject.com/Articles/115061/Determine-Information-
80         about-System-User-Processes
81     std::wstringstream ss;
82     TCHAR szVolume[MAX_PATH + 1];
83     TCHAR szFileSystem[MAX_PATH + 1];
84     DWORD dwSerialNumber;
85     DWORD dwMaxLen;
86     DWORD dwSystemFlags;
87
88     TCHAR szDrives[MAX_PATH + 1];
89     DWORD dwLen = GetLogicalDriveStrings(MAX_PATH, szDrives);
90     TCHAR* pLetter = szDrives;
91
92     BOOL bSuccess;
93
94     while (*pLetter) {
95         bSuccess = GetVolumeInformation(pLetter, // The source
96             szVolume, MAX_PATH, // Volume Label (LABEL)
97             &dwSerialNumber, &dwMaxLen, // Serial Number (VOL)
98             &dwSystemFlags,
99             szFileSystem, MAX_PATH); // File System (NTFS, FAT...)
100
101         if (bSuccess) {
102             ss << *pLetter << ":" << std::endl;
103
104             // LABEL command
105             ss << "\\tLabel:\\t" << szVolume << std::endl;
106
107             // Standard format to display serial number (VOL command)
108             ss << "\\tNumbr:\\t" << HIWORD(dwSerialNumber) << "-" << LOWORD(
109                 dwSerialNumber) << std::endl;
110
111             // File-System

```

```

111         ss << "\\tFSysm:\\t" << szFileSystem << std::endl << std::endl << std::
            endl;
112     }
113     else {
114         ss << "No data for " << pLetter << std::endl << std::endl << std::endl
            ;
115     }
116
117     while (*++pLetter); // Notice Semi-colon!
118     pLetter++;
119 }
120 return ss.str();
121 }
122
123 double MySystem::GetTotalMemory(){
124     return memoryStatus.ullTotalPhys / OneGB;
125 }
126
127 double MySystem::GetFreeMemory(){
128     return memoryStatus.ullAvailPhys / OneGB;
129 }
130
131 double MySystem::GetPagefileMemory(){
132     return memoryStatus.ullTotalPageFile / OneGB;
133 }
134
135 std::wstring MySystem::GetVideoInformation(){
136     std::wstringstream ss;
137     int deviceIndex = 0;
138     int result;
139
140     do {
141         PDISPLAY_DEVICE displayDevice = new DISPLAY_DEVICE();
142         displayDevice->cb = sizeof(DISPLAY_DEVICE);
143
144         result = EnumDisplayDevices(NULL, deviceIndex++, displayDevice, 0);
145
146         if (displayDevice->StateFlags & DISPLAY_DEVICE_ACTIVE) {
147             PDISPLAY_DEVICE monitor = new DISPLAY_DEVICE();
148             monitor->cb = sizeof(DISPLAY_DEVICE);
149
150             EnumDisplayDevices(displayDevice->DeviceName, 0, monitor, 0);
151
152             ss << "Display Device:\\t" << displayDevice->DeviceName << std::endl;
153             ss << "Display String:\\t" << displayDevice->DeviceString << std::endl;

```

```

154     ss << "Display ID:\t" << displayDevice->DeviceID << std::endl << std::
        endl;;
155
156     ss << "\tMonitor Device:\t" << monitor->DeviceName << std::endl;
157     ss << "\tMonitor String:\t" << monitor->DeviceString << std::endl;
158     ss << "\tMonitor ID:\t" << monitor->DeviceID << std::endl;
159
160     PDEVMODE dm = new DEVMODE();
161     if (EnumDisplaySettings(displayDevice->DeviceName,
        ENUM_CURRENT_SETTINGS, dm)) {
162         ss << std::endl;
163         ss << "\tFreq.: \t" << dm->dmDisplayFrequency << std::endl;
164         ss << "\tBPP: \t" << dm->dmBitsPerPel << std::endl;
165         ss << "\tWidth: \t" << dm->dmPelsWidth << std::endl;
166         ss << "\tHeig.: \t" << dm->dmPelsHeight << std::endl;
167     }
168 }
169
170 } while (result);
171
172 //ss << nWidth << "x" << nHeight;
173 return ss.str();
174 }
175
176 std::wstring MySystem::GetWindowsVersion(){
177     // See https://msdn.microsoft.com/en-us/library/ms724429%28VS.85%29.aspx
178     DWORD dwWinVer = GetVersion();
179     std::wstringstream ss;
180
181     if (IsWindows8Point1OrGreater()) {
182         ss << "Windows 8.1";
183     }
184     else if (IsWindows8OrGreater()) {
185         ss << "Windows 8";
186     }
187     else if (IsWindows7SP1OrGreater()) {
188         ss << "Windows 7 SP1";
189     }
190     else if (IsWindows7OrGreater()) {
191         ss << "Windows 7";
192     }
193     else if (IsWindowsVistaSP2OrGreater()) {
194         ss << "Vista SP2";
195     }
196     else if (IsWindowsVistaSP1OrGreater()) {

```



```

197     ss << "Vista SP1";
198 }
199 else if (IsWindowsVistaOrGreater()) {
200     ss << "Vista";
201 }
202 else if (IsWindowsXPSP3OrGreater()) {
203     ss << "XP SP3";
204 }
205 else if (IsWindowsXPSP2OrGreater()) {
206     ss << "XP SP2";
207 }
208 else if (IsWindowsXPSP1OrGreater()) {
209     ss << "XP SP1";
210 }
211 else if (IsWindowsXPOrGreater()) {
212     ss << "XP";
213 }
214 ss << ", " << LOBYTE(LOWORD(dwWinVer)) << "." << HIBYTE(LOWORD(dwWinVer));
215
216 return ss.str();
217 }
218
219 double MySystem::GetWindowsBuild(){
220     return osvInfo.dwBuildNumber;
221 }
222
223 std::wstring MySystem::GetWindowsRole(){
224     std::wstringstream ss;
225
226     switch (osvInfo.wProductType) {
227     case VER_NT_WORKSTATION:
228         ss << "Workstation";
229         break;
230     case VER_NT_SERVER:
231         ss << "Server";
232         break;
233     case VER_NT_DOMAIN_CONTROLLER:
234         ss << "Domain Controller";
235         break;
236     default:
237         ss << "Unknown";
238     }
239
240     return ss.str();
241 }

```

```

242
243 std::wstring MySystem::GetConnectionInformation(){
244     // See https://msdn.microsoft.com/en-us/library/ms724429%28VS.85%29.aspx
245     std::wstringstream ss;
246     PIP_ADAPTER_INFO pAdapterInfo;
247     ULONG ulOutBufLen = sizeof(IP_ADAPTER_INFO);
248
249     pAdapterInfo = (IP_ADAPTER_INFO *)MALLOC(sizeof(IP_ADAPTER_INFO));
250     if (GetAdaptersInfo(pAdapterInfo, &ulOutBufLen) == NO_ERROR) {
251         PIP_ADAPTER_INFO pAdapter = pAdapterInfo;
252         while (pAdapter) {
253             ss << "Adapter " << pAdapter->AdapterName << " /" << pAdapter->
                Description << "/" << std::endl;
254             ss << "\tMAC addr:\t";
255             for (UINT i = 0; i < pAdapter->AddressLength; i++) {
256                 if (i == (pAdapter->AddressLength - 1))
257                     ss << (int)pAdapter->Address[i] << std::endl;
258                 else
259                     ss << (int)pAdapter->Address[i] << "-";
260             }
261             ss << "\tIP Address:\t " << pAdapter->IpAddressList.IpAddress.String
                << std::endl;
262             ss << "\tIP Mask:\t " << pAdapter->IpAddressList.IpMask.String << std
                ::endl;
263             ss << "\tGateway:\t " << pAdapter->GatewayList.IpAddress.String << std
                ::endl;
264             if (pAdapter->DhcpEnabled) {
265                 ss << "\tDHCP Enabled:\t Yes" << std::endl;
266                 ss << "\tDHCP Server:\t " << pAdapter->DhcpServer.IpAddress.String
                    << std::endl;
267             }
268             else
269                 ss << "\tDHCP Enabled: No" << std::endl;
270
271             pAdapter = pAdapter->Next;
272         }
273     }
274     FREE(pAdapterInfo);
275     return ss.str();
276 }
277 std::wstring MySystem::GetUptimeInformation(){
278     std::wstringstream ss;
279
280     unsigned long uptime = (unsigned long)GetTickCount64();
281     unsigned int days = uptime / (24 * 60 * 60 * 1000);

```

```

282     uptime %= (24 * 60 * 60 * 1000);
283     unsigned int hours = uptime / (60 * 60 * 1000);
284     uptime %= (60 * 60 * 1000);
285     unsigned int minutes = uptime / (60 * 1000);
286     uptime %= (60 * 1000);
287     unsigned int seconds = uptime / (1000);
288
289     ss << days << " days, " << hours << ":" << minutes << ":" << seconds;
290
291     return ss.str();
292 }
293
294 void MySystem::FillHardwareInfo(HDEVINFO& di, SP_DEVINFO_DATA& did,
    HardwareInfo& hd) {
295     std::locale loc;
296     BYTE* pbuf = NULL;
297     DWORD reqSize = 0;
298     if (!SetupDiGetDeviceRegistryProperty(di, &did, SPDRP_DEVICEDESC, NULL,
        NULL, 0, &reqSize))
299     {
300         //error, but loop might continue?
301     }
302
303     pbuf = new BYTE[reqSize > 1 ? reqSize : 1];
304     if (!SetupDiGetDeviceRegistryProperty(di, &did, SPDRP_DEVICEDESC, NULL,
        pbuf, reqSize, NULL))
305     {
306         // device does not have this property set
307         memset(pbuf, 0, reqSize > 1 ? reqSize : 1);
308     }
309     hd.devDescription = (wchar_t*)pbuf;
310     delete[] pbuf;
311
312     TCHAR devInstanceId[MAX_DEVICE_ID_LEN];
313     memset(devInstanceId, 0, MAX_DEVICE_ID_LEN);
314     //pbuf = new BYTE[reqSize > 1 ? reqSize : 1];
315     if (SetupDiGetDeviceInstanceId(di, &did, devInstanceId, MAX_DEVICE_ID_LEN,
        NULL) == FALSE) {
316         //error, but loop might continue?
317     }
318     hd.devInstanceID.assign(devInstanceId);
319     //delete[] pbuf;
320
321     reqSize = 0;
322     if (!SetupDiGetDeviceRegistryProperty(di, &did, SPDRP_MFG, NULL, NULL, 0,

```

```

        &reqSize))
323 {
324     //error, but loop might continue?
325 }
326
327 pbuf = new BYTE[reqSize > 1 ? reqSize : 1];
328 if (!SetupDiGetDeviceRegistryProperty(di, &did, SPDRP_MFG, NULL, pbuf,
        reqSize, NULL))
329 {
330     // device does not have this property set
331     memset(pbuf, 0, reqSize > 1 ? reqSize : 1);
332 }
333 hd.hardwareMFG = (wchar_t*)pbuf;
334
335 // Small hack for VM
336 if (hd.hardwareMFG.length() > 1 && !std::isalpha(hd.hardwareMFG[1], loc))
337     hd.hardwareMFG.assign(L"(none)");
338
339 delete[] pbuf;
340 }
341
342
343 void MySystem::GetConnectedHardwareList(std::multimap<std::wstring,
        HardwareInfo>& result) {
344     result.clear();
345
346     std::vector<GUID> allClassGuids;
347     // SetupDiGetClassDevs() does not work if the class GUID is NULL and the
348     // 4th parameter is different from DIGCF_ALLCLASSES. Therefore we collect
349     all
350     // the class GUIDs in the first step, and then collect all the connected
351     devices
352     // by feeding the collected GUIDs to SetupDiGetClassDevs() in the 2nd step
353     // Step 1
354     {
355         HDEVINFO di = SetupDiGetClassDevs(NULL,
356             NULL,
357             NULL,
358             DIGCF_ALLCLASSES);
359
360         if (di == INVALID_HANDLE_VALUE) {
361             DWORD ret = ::GetLastError();
362             throw - 1;
363         }

```

```

363
364     int iIdx = 0;
365     while (true) {
366         SP_DEVINFO_DATA did;
367         did.cbSize = sizeof(SP_DEVINFO_DATA);
368         if (SetupDiEnumDeviceInfo(di, iIdx, &did) == FALSE) {
369             if (::GetLastError() == ERROR_NO_MORE_ITEMS)
370             {
371                 break;
372             }
373             else {
374                 //error, but loop might continue?
375             }
376         }
377         if (std::find(allClassGuids.begin(), allClassGuids.end(), did.
378             ClassGuid) == allClassGuids.end()) {
379             allClassGuids.push_back(did.ClassGuid);
380         }
381         iIdx++;
382     }
383     if (SetupDiDestroyDeviceInfoList(di) == FALSE) {
384         //error, but should be ignored?
385     }
386     // Step 2
387     for (unsigned int i = 0; i < allClassGuids.size(); i++) {
388         HDEVINFO di = SetupDiGetClassDevs(&allClassGuids[i],
389             NULL,
390             NULL,
391             DIGCF_PRESENT);
392         if (di == INVALID_HANDLE_VALUE) {
393             throw ::GetLastError();
394         }
395
396
397         int iIdx = 0;
398         HardwareInfo hd;
399         while (true)
400         {
401             SP_DEVINFO_DATA did;
402             did.cbSize = sizeof(SP_DEVINFO_DATA);
403             if (SetupDiEnumDeviceInfo(di, iIdx, &did) == FALSE) {
404                 if (::GetLastError() == ERROR_NO_MORE_ITEMS) {
405                     break;
406                 }

```

```

407         else {
408             //error, but loop might continue?
409         }
410     }
411
412     BYTE* pbuf = NULL;
413     DWORD reqSize = 0;
414     if (!SetupDiGetDeviceRegistryProperty(di, &did, SPDRP_CLASS, NULL,
415         NULL, 0, &reqSize))
416     {
417         //error, but loop might continue?
418     }
419
420     pbuf = new BYTE[reqSize > 1 ? reqSize : 1];
421     if (!SetupDiGetDeviceRegistryProperty(di, &did, SPDRP_CLASS, NULL,
422         pbuf, reqSize, NULL))
423     {
424         // device does not have this property set
425         memset(pbuf, 0, reqSize > 1 ? reqSize : 1);
426     }
427
428     FillHardwareInfo(di, did, hd);
429
430     result.insert(std::multimap<std::wstring, HardwareInfo>::value_type((
431         wchar_t*)pbuf, hd));
432     delete[] pbuf;
433     iIdx++;
434 }
435 }
436 }

```

Листинг 2 содержит исходный код программы SystemInformation, которая была написана для деонсрации работы в классом MySystem. Как можно видеть, программа занимается только выводом информации, полученной от класса MySystem.

Листинг 2: Исходный код программы SystemInformation

```

1 #include <tchar.h>
2 #include <cstdlib>
3 #include <string>
4 #include <iostream>
5 #include <fstream>

```

```

6 #include "MySystem.h"
7
8 int _tmain(int argc, _TCHAR* argv[]) {
9     std::locale::global(std::locale(""));
10    MySystem system;
11    std::wcout << ">> General Information" << std::endl;
12    std::wcout << "User time: \t" << system.GetUserTime() << std::endl;
13    std::wcout << "UTC time: \t" << system.GetUTCtime() << std::endl;
14    std::wcout << "User name: \t" << system.GetFUserName() << std::endl;
15    std::wcout << "Hostname: \t" << system.GetHostname() << std::endl;
16    std::wcout << std::endl;
17
18    std::wcout << "Windows: \t" << system.GetWindowsVersion() << " (build " <<
        system.GetWindowsBuild() << ")" << std::endl;
19    std::wcout << "Role: \t\t" << system.GetWindowsRole() << std::endl;
20    std::wcout << "Uptime: \t" << system.GetUptimeInformation() << std::endl;
21    std::wcout << std::endl;
22
23    std::wcout << "CPU: \t\t" << system.GetCPUVendor() << " (" << system.
        GetCPUNumber() << " cores)" << std::endl;
24    std::wcout << "Physical RAM: \t" << system.GetTotalMemory() << " (GB)" <<
        std::endl;
25    std::wcout << "Available RAM: \t" << system.GetFreeMemory() << " (GB)" <<
        std::endl;
26    std::wcout << "Pagefile: \t" << system.GetPagefileMemory() << " (GB)" <<
        std::endl;
27    std::wcout << std::endl;
28
29    std::wcout << ">> Video System Information" << std::endl << system.
        GetVideoInformation() << std::endl;
30    std::wcout << ">> Hard Disk Drive Information" << std::endl << system.
        GetVolumesInformation();
31    std::wcout << ">> Network Interface Information" << std::endl << system.
        GetConnectionInformation();
32    std::wcout << std::endl;
33
34    std::multimap<std::wstring, HardwareInfo> results;
35    system.GetConnectedHardwareList(results);
36    std::wcout << ">> Devices" << std::endl;
37
38    std::wstring devclass = _T("none");
39    for (std::multimap<std::wstring, HardwareInfo>::const_iterator it =
        results.begin(); it != results.end(); ++it) {
40        if (devclass.compare(it->first)) {
41            std::wcout << it->first << L": " << std::endl;

```

```
42     devclass.assign(it->first);
43 }
44 std::wcout << L"\tDescription: " << it->second.devDescription << std::
    endl;
45 std::wcout << L"\tInstance ID: " << it->second.devInstanceID << std::
    endl;
46 std::wcout << L"\tManufacturer: " << it->second.hardwareMFG << std::endl
    << std::endl;
47 }
48
49 //std::wcout << L"Done!" << std::endl;
50 getch();
51 exit(0);
52 }
```



# Демонстрация работы программы

Для демонстрации практической части была разработана маленькая программа SystemInformation. Пользуясь методами класса MySystem, она собирает системную информацию и выводит её на экран. Вывод можно перенаправить в файл.

## Эксперимент 1. Виртуальная Win7

Программа была запущена на виртуальной машине. Из особенностей следует отметить три ядра, доступные системе (это сделано специально, для комфортной работы гипервизора), имя видеоадаптера (виртуальная машина использует собственный драйвер) и имя производителя центрального процессора (виртуальная машина эмулирует X64 процессор).

На рисунке 6 представлены результаты работы. Они разбиты на 5 секции: общая информация, информация о видеосистеме, информация о накопителях и информация о сетевой системе. Последней секцией идёт информация обо всех устройствах, имеющихся в системе. Эта информация получена из реестра, без реального опроса оборудования. Полный текстовый лог лежит в папке со всеми прочими логами.

Разница между локальным временем и UTC соответствует Московскому часовому поясу.

В корректности полученной информации можно убедиться по рисункам 1, 2 и 3.

## Эксперимент 2. Реальная Win7

Эксперимент проводился на реальном ноутбуке Lenovo Thinkpad T410s. Программа показала значительно более высокое быстродействие, по сравнению с предыдущими запусками на виртуальных машинах. Это создало сложности при попытке сделать скриншот работы (рисунок 7). Подробный лог находится в папке с логами. Как показывает рисунок 8, данные корректы.

```
C:\Windows\system32\cmd.exe - SystemInformation.exe

UTC time:      2 015-4-15 7:52:34.520
User name:     user\win7
Hostname:      USER

Windows:       Windows 7 SP1, 6.1 (build 7601)
Role:          Workstation
Uptime:        0 days, 0:54:17

CPU:           Genuntelinel (3 cores)
Physical RAM:  3.99957 (GB)
Available RAM: 3.17392 (GB)
Pagefile:      7.99736 (GB)

>> Video System Information
Display Device: \\.\DISPLAY1
Display String: VirtualBox Graphics Adapter
Display ID:     PCI\VEN_80EE&DEV_BEEF&SUBSYS_00000000&REV_00

        Monitor Device:
        Monitor String:
        Monitor ID:

        Freq.: 60
        BPP: 32
        Width: 1 920
        Heig.: 990

>> Hard Disk Drive Information
C:
    Label:
    Numbr: 57 595-10 786
    FSysm: NTFS

D:
    Label: UBOXADDITIONS_4.
    Numbr: 963-42 577
    FSysm: CDPS

E:
    Label: UBOX_tmp
    Numbr: 0-2 049
    FSysm: VBoxSharedFolderFS

>> Network Interface Information
Adapter {B3365931-ABDF-4E10-BE73-833735EBA0EB} //
    MAC addr: 8-0-39-207-62-123
    IP Address: 192.168.124.235
    IP Mask: 255.255.255.0
    Gateway: 192.168.124.1
    DHCP Enabled: Yes
    DHCP Server: 192.168.124.1
```

Рис. 6: Результаты работы программы SystemInformation на виртуальной Win7

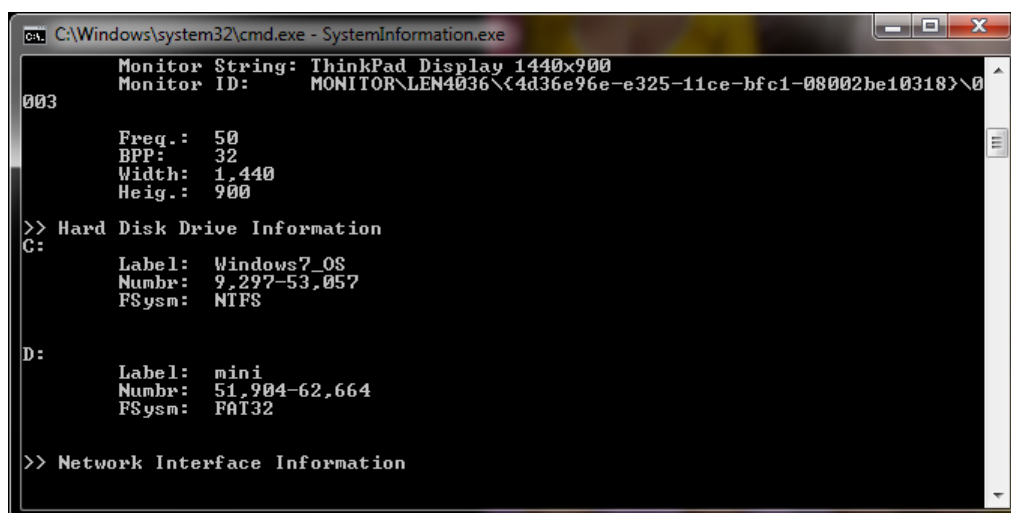


Рис. 7: Результаты работы программы SystemInformation на реальной Win7.

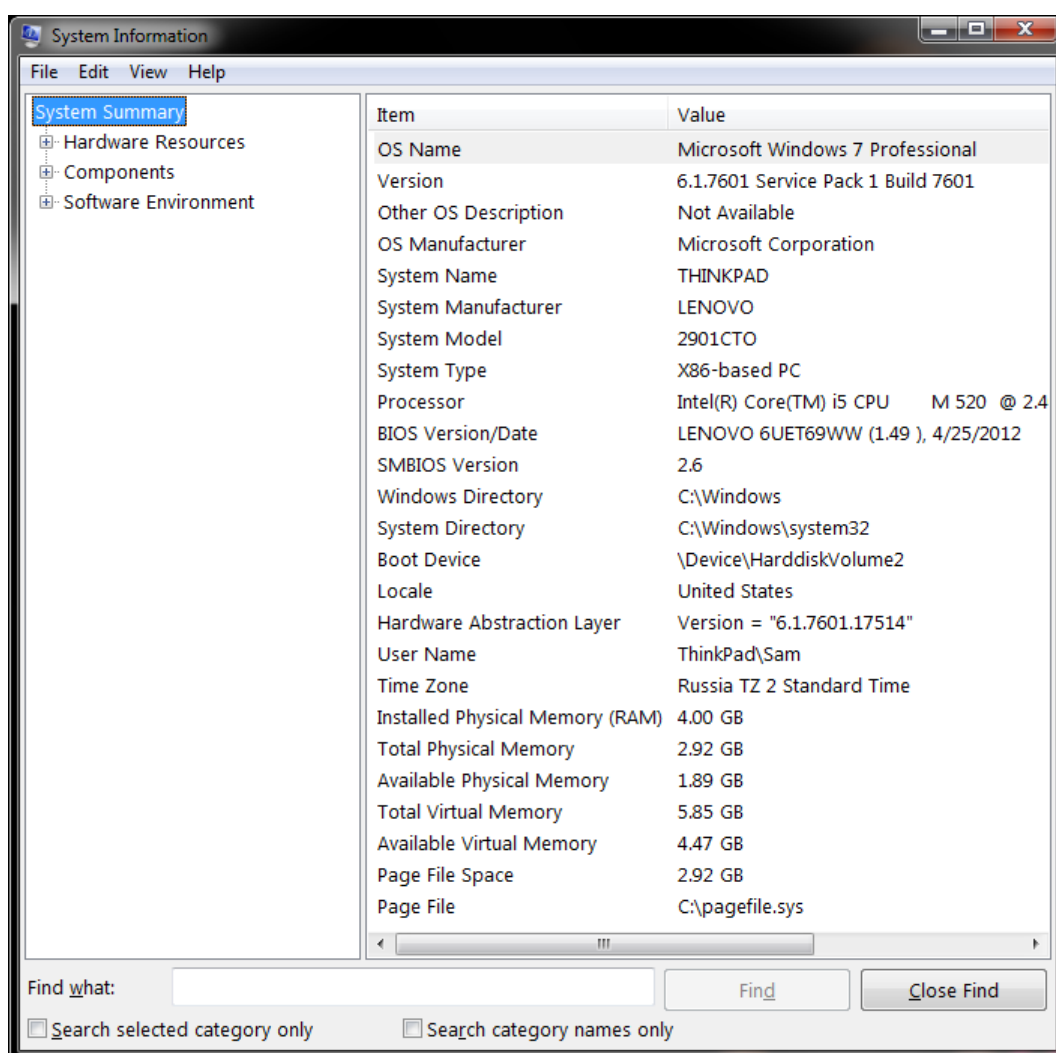


Рис. 8: Результаты работы программы SystemInformation

# Заключение

В данной работе были рассмотрены основные механизмы сбора системной информации в ОС Windows.

В отличие от мира linux, сбор системной информации осложнён разнообразностью форм её представления и разбросанностью по всей ОС.

Разработанная программа позволяет получить следующую информацию:

1. время пользователя и UTC время;
2. имя пользователя и имя хоста;
3. версию операционной системы, с точностью до номера сервиспака и сборки;
4. производителя центрального процессора и доступных ядрах;
5. различные параметры оперативной памяти (как физической так и файла подкачки);
6. параметры работы видеосистемы;
7. локальные файловые накопители и используемые файловые системы;
8. параметры работы сетевой системы.

Корректность работы программы была проверена путём сверки данных с другими системными источниками (лог-файлы находятся в общем архиве, в папке с логами). Данный код можно использовать в качестве динамической библиотеки в других, более масштабных проектах.