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 сбора системной информации в ОС Windows

Работу выполнил студент гр. 53501/3 _____ Мартынов С. А.

Работу принял преподаватель _____ Душутина Е. В.

Санкт-Петербург
2015

Оглавление

Постановка задачи

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

В процессе изучения предполагается разработать консольную утилиту, отображающую на экран 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 исполняется код.

По результатам работы предоставить отчёт, исходные коды сделать доступными по адресу `https://github.com/SemenMartynov/SPbPU_ParallelProgramming`.

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).

Метод GetUTCTime()

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

Метод GetFUserName()

Возвращает полное имя пользователя, с учётом имени домена.

Метод GetHostname()

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

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.

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.

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).

Метод GetTotalMemory()

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

Метод GetFreeMemory()

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

Метод GetPagefileMemory()

Возвращает (в гибибайтах) общий объём системного файла подкачки.

Метод GetVideoInformation()

Возвращает подробную информацию по видиосистеме. 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.

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.

Метод GetWindowsBuild()

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

Метод GetWindowsRole()

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

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.

Метод GetUptimeInformation()

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

Метод GetUptimeInformation()

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

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::wstringstream 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::wstringstream 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-
          about-System-User-Processes
```

```
80    std::wstringstream ss;
81    TCHAR szVolume[MAX_PATH + 1];
82    TCHAR szFileSystem[MAX_PATH + 1];
83
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 formal to display serial number (VOL command)
108       ss << "\tNumbr:\t" << HIWORD(dwSerialNumber) << "-" << LOWORD(
              dwSerialNumber) << std::endl;
109
110       // 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 }
```

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.

Проведение эксперимента

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 1 представлены результаты работы. Они разбиты на 4 секции: общая информация, информация о видеосистеме, информация о накопителях и информация о сетевой системе.

Для подтверждения корректности работы программы, было сделано ещё несколько снимков. На рисунке 2 представлен диспетчер задач. Он показывает актуальное количестве ядер, объёмы оперативной памяти (общую физическую и свободную физическую) и общее времени работы.

Рисунок 3 отображает свойства сетевого подключения. Все параметры соответствуют тому, что было получено нам с помощью программы SystemInformation.

На рисунке 4 запущена оснастка управления дисками. В ней можно видеть все локальные разделы, используемые в системе, их имена и используемые файловые системы.

Представленные снимки подтверждают корректность информации, которую мы получили. Стоит отметить, что в реальной системе эта информация разбросана по разным окнам и оснасткам, вто время как программа SystemInformation позволяет собрать всю информацию в едином месте и компактно её отобразить.

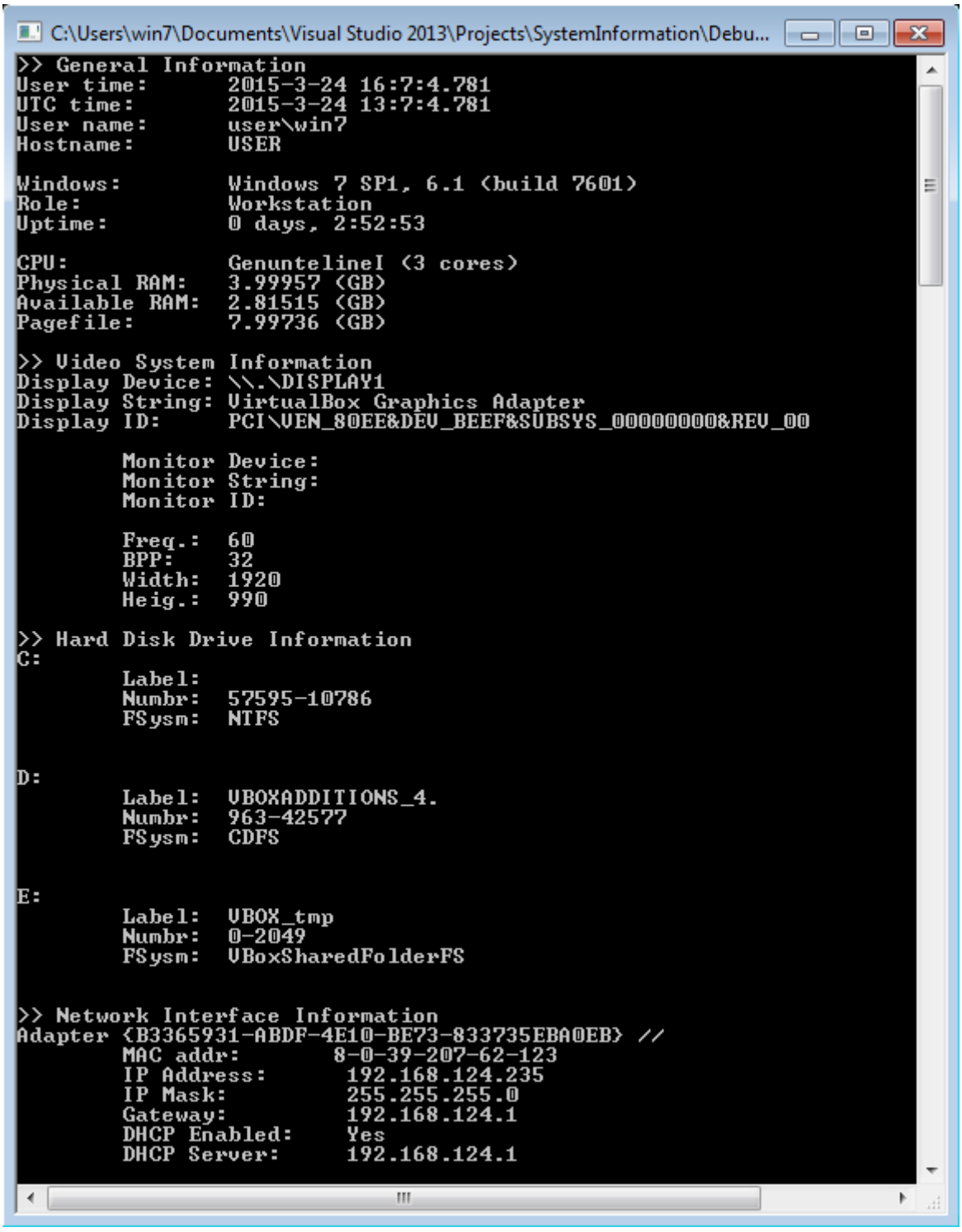

Рис. 1: Результаты работы программы SystemInformation

Разница между локальным временем и UTC соответствует Московскому часовому поясу.

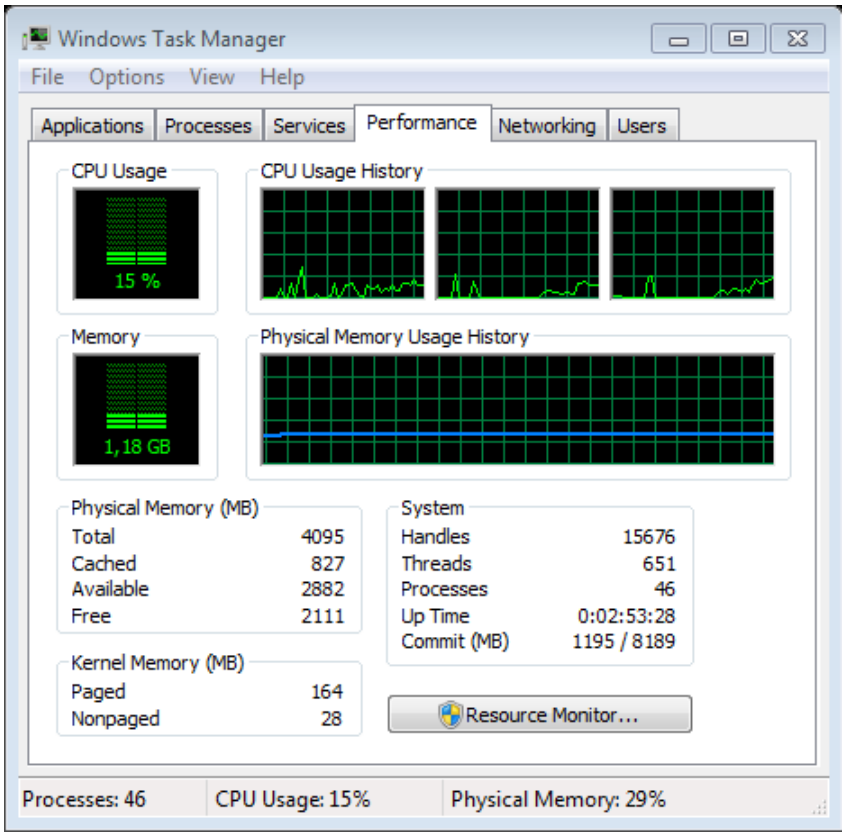

Рис. 2: Диспетчер задач

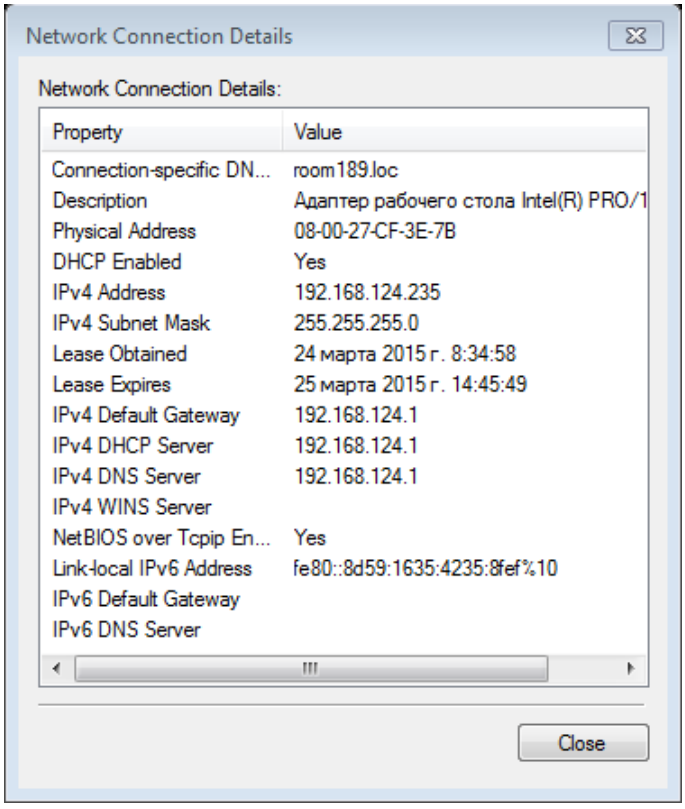

Рис. 3: Информация о сетевом подключении

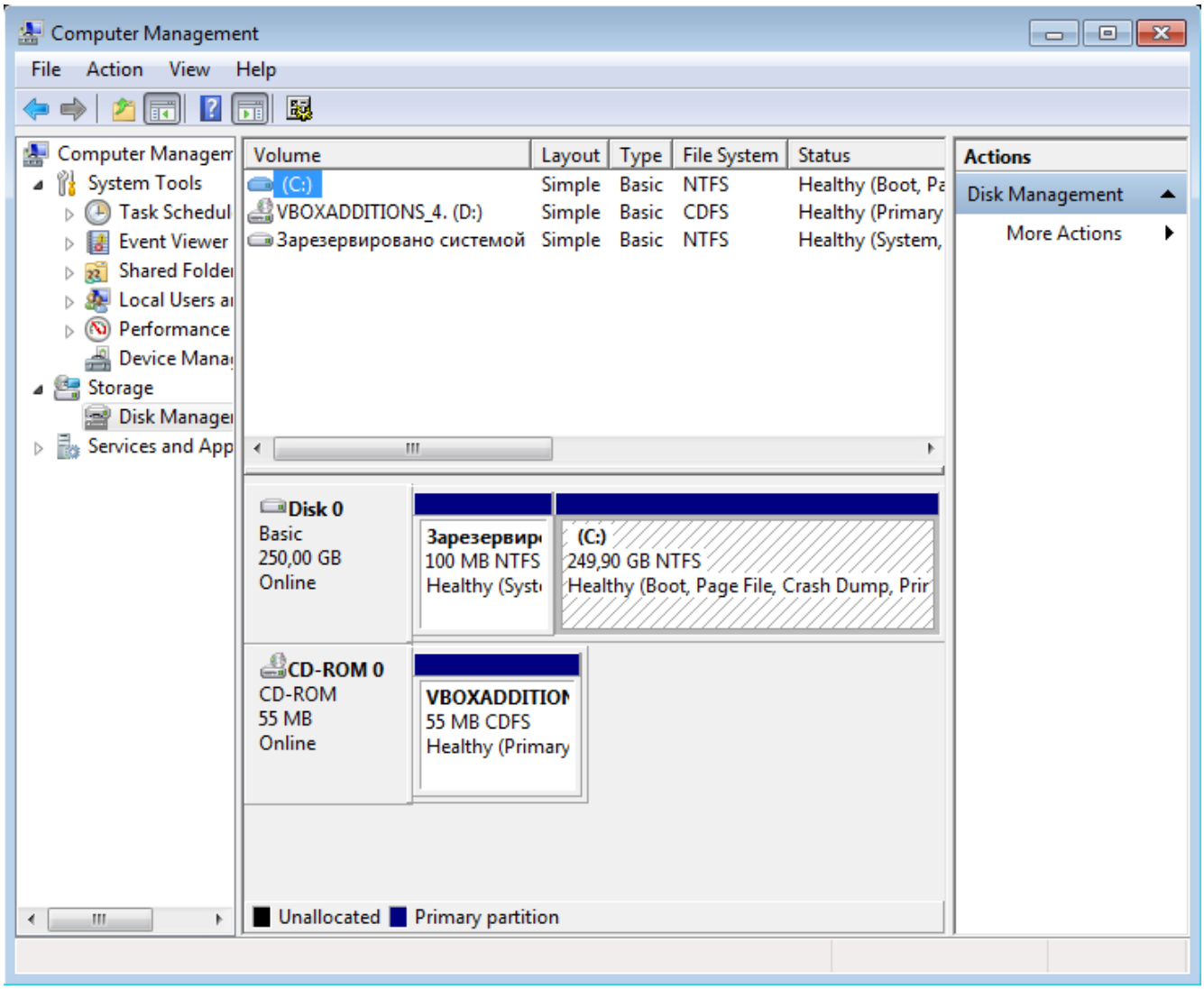

Рис. 4: Оснастка управления дисками

Листинг

Листинг 2 содержит исходный код программы SystemInformation. 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 "MySystem.h"
6
7 int _tmain(int argc, _TCHAR* argv[]) {
8   MySystem systm;
9   std::wcout << ">> General Information" << std::endl;
```

```
10    std::wcout << "User time: \t" << systm.GetUserTime() << std::endl;
11    std::wcout << "UTC time: \t" << systm.GetUTCTime() << std::endl;
12    std::wcout << "User name: \t" << systm.GetFUserName() << std::endl;
13    std::wcout << "Hostname: \t" << systm.GetHostname() << std::endl;
14    std::wcout << std::endl;
15
16    std::wcout << "Windows: \t" << systm.GetWindowsVersion() << " (build " <<
          systm.GetWindowsBuild() << ")" << std::endl;
17    std::wcout << "Role: \t\t" << systm.GetWindowsRole() << std::endl;
18    std::wcout << "Uptime: \t" << systm.GetUptimeInformation() << std::endl;
19    std::wcout << std::endl;
20
21    std::wcout << "CPU: \t\t" << systm.GetCPUVendor() << " (" << systm.
          GetCPUNumber() << " cores)" << std::endl;
22    std::wcout << "Physical RAM: \t" << systm.GetTotalMemory() << " (GB)" <<
          std::endl;
23    std::wcout << "Available RAM: \t" << systm.GetFreeMemory() << " (GB)" <<
          std::endl;
24    std::wcout << "Pagefile: \t" << systm.GetPagefileMemory() << " (GB)" <<
          std::endl;
25    std::wcout << std::endl;
26
27    std::wcout << ">> Video System Information" << std::endl << systm.
          GetVideoInformation() << std::endl;
28    std::wcout << ">> Hard Disk Drive Information" << std::endl << systm.
          GetVolumesInformation();
29    std::wcout << ">> Network Interface Information" << std::endl << systm.
          GetConnectionInformation();
30
31    getwchar();
32    exit(0);
33 }
```

Заключение

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

В отличии 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. Данный код можно использовать в качестве динамической библиотеки в других, более масштабных проектах.