File Duplicator: An Example of Windows Services

ZBIGNIEW A. NOWACKI
Computer Engineering Department, Technical University of Lodz

The article presents a tool utility, designed by the author and termed File Duplicator, running under the Microsoft Windows operating system. We have demonstrated that in many cases, especially related to the automatic back-up services, this program can be found useful. One should pay attention to the fact that File Duplicator is an example of memory-resident applications called Windows Services. We have pointed out some differences between such residents and normal programs. A universal and reliable method for creating and installing Windows Services has been specified.

Key Words: **Windows service, Windows application, console application, disk storage, memory-resident, operating system**

1. Introduction

The principal motivation for the creation of File Duplicator was the widely understood safety of data being preserved in disk files. In the case of malfunction or malicious attack [1], it is good to have a copy of a lost file. The most popular method of achieving this goal is to archive the disk storage periodically [2]. The disadvantages of this approach are that the process may be time-consuming, and there is a possibility of loss of data that have been updated after the last backup.

File Duplicator removes these drawbacks, for the backup process takes place in a manner transparent to the user, and there is always a practically up-to-date copy of a selected disk file. However, this method can be used solely for the most important files; their maximum number depends on the size of the computer memory.

It is worth emphasizing that, in contrast to traditional archiving systems, File Duplicator does not create backup copies at regular intervals, but it makes them only if the original file has been changed. This novel approach can help to reduce the consumption of computing resources such as disk storage and CPU time.

Older computer users remember the text editors or word processors [3] that after the update of a document wrote its previous contents to the file with the extension ".BAK". This was convenient, but most of today's programs do not offer this facility. Therefore, it is worth noting that applying File Duplicator you may back up the previous contents of a chosen file as well. This option is of importance also because if, e.g., a virus modifies the file while retaining its name, then the current copy will be corrupted, but the prior copy will be correct.

Using the program is easy. Its executable can be downloaded from the author's web page [4], and run with command-line arguments discussed in Section 2. File Duplicator is able to be started with any method known under the Microsoft Windows operating system [5], but it is best to run it automatically at startup. This can be done by initiating the utility via the *Startup* folder (see Section 3.) or, better, by installing it as a Windows Service [6, 7]. Such a solution is a preferred technique to build the equivalent of a UNIX daemon [8].

Windows Services (cf. Section 4.) work without revealing its presence and usually remain unnoticed by the user. Thus they are able to be treated as a component of the operating system [9]. Their use and implementation is somewhat mysterious, mainly because of the ambiguities

and errors occurring in the descriptions that are available on the Internet. That is why Section 5. presents a universal and reliable method of installing Windows Services, and Section 6. contains the full source code of File Duplicator. The reader can easily adapt it to their goals and write their own Windows Service.

2. The command-line arguments of File Duplicator

Savefile.exe (the executable version of File Duplicator) requires at least one command-line argument specifying a disk file *file0*. The program resides in memory [8, 10] and ensures that anytime:

(a) There is a disk file *file1* containing the exact copy of *file0*.

(b) There is a disk file *file2* containing the prior copy of *file0*.

Precisely speaking, after starting *Savefile.exe* checks whether *file0* and *file1* exist. If the former exists, but the latter does not, *file0* is copied to *file1*. Next, the program checks every 2000 (the number may be changed in the fourth command-line argument) milliseconds whether *file0* has appeared or it has been modified. If so, then:

(i) *file2* (if it exists) is deleted.

(ii) *file1* (if it exists) is renamed to *file2*.

(iii) *file0* is copied to *file1*.

The name of *file1* may be specified in the second command-line parameter. Otherwise, it will have the same base name as *file0* with the extension ".$$$". Similarly, *file2* may be specified in the third argument or it equals the base name of *file1* plus the extension ".BAK". The first three parameters should be full paths unless you want to change solely the disk (typing a single letter followed by, maybe, a colon). For instance,

```
Savefile  d:\docs\myfile.doc  e:
```

specifies that *file1* is equal to *e:\docs\myfile.$$$*, and *file2* - to *e:\docs\myfile.bak*, while

```
Savefile  d:\docs\myfile.doc  d  d  1000
```

does not alter default names.

If you do not need the previous copy of *file0*, set the third parameter to a digit, e.g.,

```
Savefile  d:\docs\myfile.doc  d  0
```

Note that a command-line argument cannot contain spaces. Therefore, if you have a long name [11] with spaces, use either its short counterpart [12] or the following option. If the first character of the first parameter is not a letter, in the first three parameters it is replaced by a space. For example,

```
Savefile  &d:\program&files\my&file.doc  d:  e:\prior&file.doc
```

specifies that *file0* is equal to *d:\program files\my file.doc*, and *file2* - to *e:\prior file.doc*.

As follows from the above description, *Savefile.exe* backs up a single disk file. However, it is possible to run a number of its copies with different first parameters. The only limitation is the

computer memory capacity (a single copy of the program takes about 1 MB of memory). Note also that *file0* of a copy can be equal to *file2* of another. In this fashion you may obtain a chain of consecutive copies of a very important file.

3. Installing File Duplicator in the Startup folder

As *Savefile.exe* requires command-line arguments, it cannot be directly located in the *Startup* folder. However, you may compile an auxiliary C program **[**13, 14**]** containing [15]

```
#include <process.h>

#ifndef   P_NOWAIT
#define     P_NOWAIT     _P_NOWAIT
#endif

void main()
{
spawnl (P_NOWAIT, "full path\\savefile.exe",
          "full path\\savefile.exe",
          "first parameter for savefile.exe",
          "second parameter",...,NULL);
spawnl (P_NOWAIT, "full path\\savefile.exe",                    // optional
          "full path\\savefile.exe",
          "another parameter",NULL);
}
```

and its executable can be moved to the *Startup* folder.

4. Windows Services

Microsoft Windows Services [6, 7], earlier called NT Services, are memory-resident (i.e., working in the background) programs running in their own window stations that are different from the interactive station of the logged-on user. The reader familiar with Unix-type systems [16] may want to compare them with Unix daemons [8]. In fact, they also set up a mechanism for being called up either periodically or by an application at a later time, and otherwise remain idle in the background until explicitly stopped.

A window station [6] is a secure object containing a local clipboard and a group of local desktop objects. This implies that a Windows Service cannot show any user interface (with the exceptions described in [17]). Precisely speaking, a utility of this type is not able to input or output anything (unless it is located in the disk storage or a method of interprocess communication [18] is applied). Even error messages should not be raised in the user interface, since dialog boxes will not be visible and can cause the program to stop responding.

After installing, Windows Services are able to be automatically started when the system boots (earlier than the user logs into the computer). They run under the system account that has more privileges and permissions than a user (even administrator) account. At any time services can be stopped and restarted via Windows Task Manager. From the application level this may be done with the aid of, e.g., global semaphores [19-21].

5. Installing File Duplicator as a Windows Service

To install *Savefile.exe* as a Windows Service you will need utilities called *Instsrv.exe* and *Srvany.exe*. They were designed for Windows XP, but under Windows 7 and Windows Vista they work fine as well. The programs are contained in the Windows NT Resource Kit that can be downloaded from [22]. In this way you will obtain the *Rktools.exe* package. After running it, *Instsrv.exe*, *Srvany.exe* and other files will be written to the *c:\Program Files\Windows Resource Kits\Tools* folder, where *c:* might be replaced by another designation of the system disk.

Now perform the following steps:

(i) At a command prompt (running as administrator under Windows 7 and Vista), type the following command:

```
<path>\Instsrv.exe   SAVEFILE   <path>\Srvany.exe
```

where <*path*> is the drive and directory of the utilities (i.e., most frequently the above folder). This creates a service with the name *SAVEFILE* (it may be changed).

(ii) Run Registry Editor (*Regedt32.exe* of the *System32* folder) and locate the following subkey:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SAVEFILE
```

(iii) From the *Edit* menu select *New Key* and type:

```
Parameters<Enter>
```

(iv) From the *Edit* menu choose *New String Value* and type:

```
Application<Enter>
```

(v) From the *Edit* menu (or clicking the right mouse button at *Application*) select *Modify...* and type:

```
<path>\savefile.exe  command-line parameters   <Enter>
```

where <*path*> is a full path to *Savefile.exe*, e.g.,

```
d:\programs\savefile.exe  c:\myfile.doc  d:\copies\copy1.doc  <Enter>
```

(vi) Close Registry Editor.

Savefile.exe will run automatically with these command-line parameters when the system is restarted. Many such services can be configured if necessary. They have to have different names, but they might use the same full paths of *Savefile.exe* and *Srvany.exe.*

Using *Instsrv.exe* it is also possible to delete services, e.g.,

```
Instsrv.exe  SAVEFILE  remove
```

Remember that earlier it may be necessary to stop the service with the aid of Task Manager.

When replacing *Savefile.exe* by another application, the foregoing method enables us to run it as a service. However, this makes sense only for some programs that precisely for this reason are able to be called Windows Services. Similar installation instructions can be found in [23], but they contain errors and say nothing about command-line parameters. Their use is vital here because of the absence of the normal user interface.

6. The full source code of File Duplicator

The program has been written in the C programming language [13, 14]. Its full source code is presented below:

```
// FILE DUPLICATOR
#include <windows.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>
#include <stdio.h>
#include <fcntl.h>
#include <share.h>
#include <ctype.h>
// CONSTANTS
#define    BUFSIZE              50000                  // the buffer size
#ifndef    MAX_PATH
  #define MAX_PATH          _MAX_PATH         // the maximum path size
#endif
// EXTERNAL VARIABLES
char buf[BUFSIZE];
char file[3][MAX_PATH+4];
#if defined(__BORLANDC__)
  struct stati64 state;
#else
  struct _stati64 state;
#endif
time_t prior,current;
__int64 psize,csize;
int espace;
// FUNCTION PROTOTYPES
void createlong(char *file, char *arg);                    // create long path
```

```
void createpath(int n, const char *ext, char *arg);            // create copy path
int copyfile(const char *from, const char *to);                 // copy files
time_t modtime(__int64 *size);              // time of last modification of file[0]
void CALLBACK ResidentProc(HWND, UINT, UINT, DWORD);
// MAIN FUNCTION
int main(int argc, char *argv[])
{
if(argc<2) return 1;
createlong(file[0],argv[1]+(isalpha(espace=*argv[1]) ? espace=0 : 1));
createpath(1, "$$$", argc>=3 ? argv[2] : NULL);          // create path of copy
createpath(2, "BAK", argc>=4 ? argv[3] : NULL);       // create path of prior copy
// TIMER DEFINITION
SetTimer(NULL,1,argc>=5 ? atoi(argv[4]) : 2000,ResidentProc);
prior=modtime(&psize);
// MESSAGE LOOP
MSG event;
while(GetMessage(&event, NULL, 0, 0))
  {
  DispatchMessage(&event);
  }
return 0;
}
// CREATE LONG PATH
void createlong(char *file, char *arg)
{
if(espace)
  {                                                  // replace espace by space
  char *p=arg=strdup(arg);
    for(;;)
      {
      p=strchr(p,espace);
      if(!p) break;
      *p++=' ';
      }
  }
if(!GetLongPathNameA(arg,file,MAX_PATH)) strcpy(file,arg);
if(!strchr(file,'.')) strcat(file,".");
strcat(file,"   ");
}
// CREATE COPY PATH
void createpath(int n, const char *ext, char *arg)
{
memcpy(file[n],file[n-1],MAX_PATH+3);
if(arg)
  {
  if(strlen(arg)<=2) file[n][0]=arg[0];              // only disk
  else
    {
    createlong(file[n],arg);                          // path
    return ;
    }
  }
strcpy(strrchr(file[n],'.')+1,ext);
}
// COPYING FILES
int copyfile(const char *from, const char *to)
{
int i;
int infile=sopen(from, O_RDONLY | O_BINARY, SH_DENYNO);
if (infile<0) return 0;
```

```
int outfile=sopen(to, O_CREAT | O_RDWR | O_BINARY | O_TRUNC,
   SH_DENYWR, S_IREAD | S_IWRITE);
if (outfile<0) return 0;
for(;;)
   {
   i=read(infile,buf,BUFSIZE);
   if(!i) break;
   write(outfile,buf,i);
   }
close(outfile);
close(infile);
return 1;
}
// TIME OF LAST MODIFICATION OF FILE[0]
time_t modtime(__int64 *size)]
{
if(!_stati64(file[0], &state ))
   {
   if(size) *size=state.st_size;
   return state.st_mtime;
   }
if(size) *size=-1;
return 0;
}
// TIMER PROCEDURE
void CALLBACK ResidentProc(HWND, UINT, UINT, DWORD)
{
static int work;
if(work) return ;
work=1;
if(access(file[1],0)<0) copyfile(file[0],file[1]);
else
   {
   current=modtime(&csize);
   if(current && (current != prior || csize != psize))
      {
      if(isalpha(*file[2]))
         {
         unlink(file[2]);
         rename(file[1],file[2]);
         }
      copyfile(file[0],file[1]);
      prior=current;
      psize=csize;
      }
   }
work=0;
}
```

Let us discuss some aspects of this code. As Windows Services are designed to work without a graphical user interface, they can be console applications [24]. Therefore, although File Duplicator includes the *windows.h* file, it contains the *main* function instead of *WinMain* [25, 26].

Some tasks of services require a periodic testing of certain conditions and undertaking an action only when they are satisfied. For instance, File Duplicator has to examine if the file specified in the first parameter has appeared or has been modified. To this end, the *SetTimer* function [27] in *main* defines a so-called timer. As a result, the *ResidentProc* function will be

invoked every 2000 (by default) milliseconds. Note that the procedure of copying may be long-term, so the use of the *work* variable in the timer procedure is essential.

At the end of the *main* function one can find the loop testing events (manifested in coming messages) [28, 29]. In contrast to a normal loop, this sequence of instructions does not overburden the system even if it is being performed for a long time and by many programs simultaneously. Note that the message loop differs slightly from the analogous one in Windows applications [30-32]. For in the latter [26, 33] the call of *DispatchMessage* [34] is preceded by that of *TranslateMessage* [35] responsible for providing ASCII codes of characters entered from the keyboard. But services do not allocate the entry of this device, so *TranslateMessage* can be omitted here.

7. Conclusion

In the paper we present an easy to use and helpful tool utility, called File Duplicator, for creating up-to-date copies of important disk files. Its purpose, command-line arguments and installation methods have been discussed in detail. The full source code of File Duplicator has been also included.

It should be emphasized that File Duplicator is an example of applications called Windows Services. Hence the instructions incorporated in the work may be applied to run any program as a Windows Service. And the whole article can be regarded as a guide to writing such memory-resident utilities by those who know the C/C++ language [13, 14, 36] even if they have no experience in building so-called Windows applications [30-32].

References

1. Zimoch M. *Privacy and Security on the Internet*. Master's thesis supervised by Z. A. Nowacki, Lodz Technical University, 2009.

2. Back up - Guide to backing up the important files on your hard drive.
http://www.helpwithpcs.com/maintenance/back-up-backing-up-guide.htm

[2 January 2012].

3. Ignaczak M. *The Comparative Analysis of Computer Text Processing Systems.* Master's thesis supervised by Z. A. Nowacki, Lodz Technical University, 2008.

4. Modern TSR programs by Zbigniew Andrzej Nowacki.
http://www.nova.pc.pl/software.htm [2 January 2012].

5. Kominiak L. *The Genesis, Evolution and Structure of Microsoft Windows Operating Systems*. Master's thesis supervised by Z. A. Nowacki, Lodz Technical University, 2007.

6. Introduction to Windows Service Applications.
http://msdn.microsoft.com/en-us/library/d56de412(VS.80).aspx [2 January 2012].

7. Services.
http://msdn.microsoft.com/en-us/library/ms685141.aspx [2 January 2012].

8. Writing resident programs under Linux.
http://rudy.mif.pg.gda.pl/~bogdro/linux/tsr_tut_linux_en.html [2 January 2012].

9. Stallings W. *Operating Systems: Internals and Design Principles.* Prentice Hall, New Jersey, 2009.

10. Tomczyk M. *The Methods of Creating Resident Programs in the DOS and Windows Systems*. Master's thesis supervised by Z. A. Nowacki, Lodz Technical University, 2001.

11. Naming Files, Paths, and Namespaces.
http://msdn.microsoft.com/en-us/library/aa365247(v=vs.85).aspx [2 January 2012].

12. GetShortPathName Function.
http://msdn.microsoft.com/en-us/library/aa364989(v=vs.85).aspx [2 January 2012].

13. Kernighan B. W., Ritchie D. M. *The C Programming Language. Second Edition.* Prentice Hall, New Jersey, 1988.

14. C Language Tutorial.
http://einstein.drexel.edu/courses/Comp_Phys/General/C_basics/ [2 January 2012].

15. spawn functions.
http://www.users.pjwstk.edu.pl/~jms/qnx/help/watcom/clibref/src/spawn.html

[2 January 2012].

16. Bukowiecki M. *The Genesis, Development and Practical Applications of the Linux System.* Master's thesis supervised by Z. A. Nowacki, Lodz Technical University, 2007.

17. Interactive Services.
http://msdn.microsoft.com/en-us/library/ms683502(v=VS.85).aspx [2 January 2012].

18. Interprocess Communications.
http://msdn.microsoft.com/en-us/library/windows/desktop/aa365574(v=vs.85).aspx
[2 January 2012].

19. Szurgot R. *Parallel Programming in Windows*. Master's thesis supervised by Z. A. Nowacki, Lodz Technical University, 2009.

20. Multiple Threads.
http://msdn.microsoft.com/en-us/library/ms684254(v=vs.85).aspx [2 January 2012].

21. Concurrent Processes: Basic Issues.
http://cnx.org/content/m12312/latest/ [2 January 2012].

22. Windows Server 2003 Resource Kit Tools.
http://www.microsoft.com/download/en/confirmation.aspx?id=17657 [2 January 2012].

23. How To Create a User-Defined Service.
http://support.microsoft.com/default.aspx?scid=kb;en-us;137890 [2 January 2012].

24. Running Console Applications from Windows.
http://www.hermetic.ch/rundos.htm [2 January 2012].

25. WinMain Entry Point.
http://msdn.microsoft.com/en-us/library/ms633559(v=vs.85).aspx [2 January 2012].

26. WinMain - application entry point.
http://www.toymaker.info/Games/html/winmain.html [2 January 2012].

27. SetTimer Function.
http://msdn.microsoft.com/en-us/library/ms644906(v=vs.85).aspx [2 January 2012].

28. MSG Structure.
http://msdn.microsoft.com/en-us/library/ms644958(v=vs.85).aspx [2 January 2012].

29. GetMessage Function.
http://msdn.microsoft.com/en-us/library/ms644936(v=vs.85).aspx [2 January 2012].

30. Petzold C. *Programming Windows*. Microsoft Press, 1998.

31. Richter J. *Programming Applications for Microsoft Window.* Microsoft Press, 1999.

32. Prosise J. *Programming Windows with MFC.* Microsoft Press, 1999.

33. Creating a Message Loop.
http://msdn.microsoft.com/en-us/library/ms644928(v=vs.85).aspx#creating_loop
[2 January 2012].

34. DispatchMessage Function.
http://msdn.microsoft.com/en-us/library/ms644934(v=vs.85).aspx [2 January 2012].

35. TranslateMessage Function.
http://msdn.microsoft.com/en-us/library/ms644955(v=vs.85).aspx [2 January 2012].

36. Stroustrup B. *The C++ Programming Language. Third Edition.* Addison Wesley, New Jersey, 1997.