robocopyUI – User Guide and Implementation Details

1. Introduction

robocopyUI is a tool for copying large numbers of files and directories via *Windows Explorer* context menu with the aim of solving presumably the biggest misadvantage of Windows' *robocopy* command: copying multiple directories.

As the name suggests, *robocopyUI* uses Windows' *robocopy* command in the background. Detailed explanation of *robocopy* command can be found at docs.microsoft.com/en-us/windows-server/administration/windows-commands/robocopy. In short, robocopy's syntax is in form of:

```
robocopy <Source> <Destination> [<File>[ ...]] [<Options>]
```

where `<Source>` and `<Destination>` denote source and destination directories, `[<File>[...]]` denotes list of files to copy, and `[<Options>]` denotes robocopy options. Complete list of options can be found at the above mentioned link.

As mentioned, `<File>` argument can only be a file and not a directory. This means that for every directory we want to move from `<Source>` to `<Destination>` we have to write a separate robocopy command. There is a "\e" option which copies everything from `<Source>` to `<Destionation>`, but oftentimes we want to copy only several files and directores and not all of them. *robocopyUI* solves this problem by generating robocopy script for each selected directory and one additional script for selected files. Additionaly, there is a Explorer context menu shortcut "Execute Robocopy" which executes the generated script with current directory as `<Destination>`.

2. User Guide

Install robocopyUI by running the *robocopyUIInstaller.exe* which can be found at the github page of the project, under releases (https://github.com/Jovan-Zan/robocopyUI).

Uninstallation is even simpler, just remove *robocopyUI* in Control panel Apps and Features menu.

Using *robocopyUI* consists of five steps. These steps are portrayed by three figures shown bellow. The five steps are:

1. selecting files and directories we want to copy in Windows Explorer
2. right click → SendTo → Robocopy Script Generator
3. move to desired destination directory
4. right click → Execute Robocopy
5. monitor robocopy output in terminal (cmd)

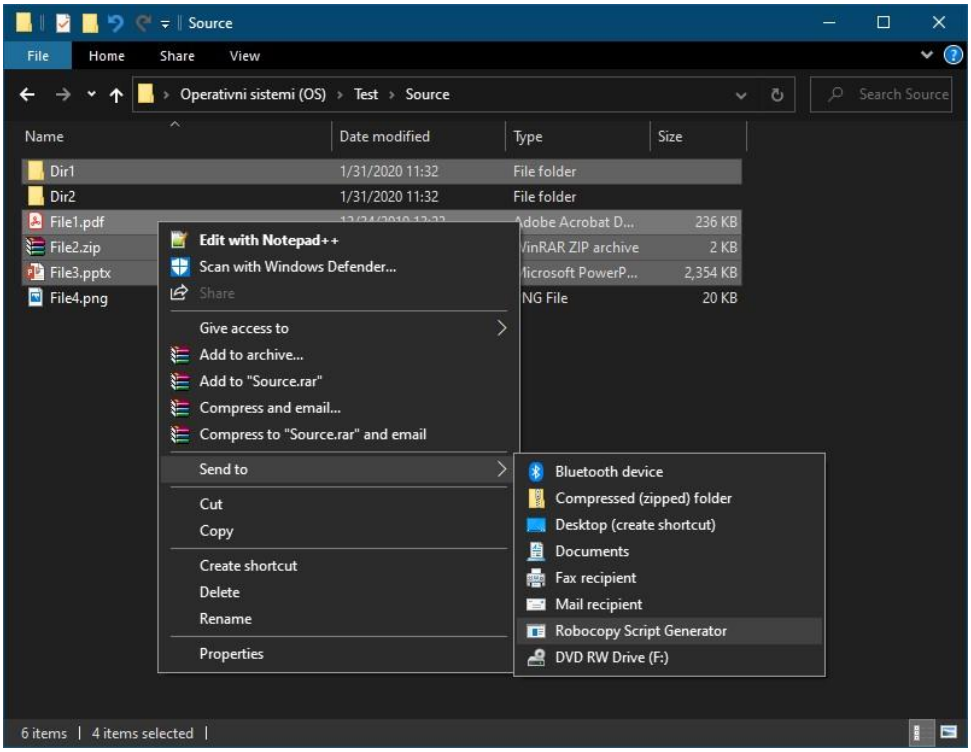

Figure 1 Generate Robocopy Script

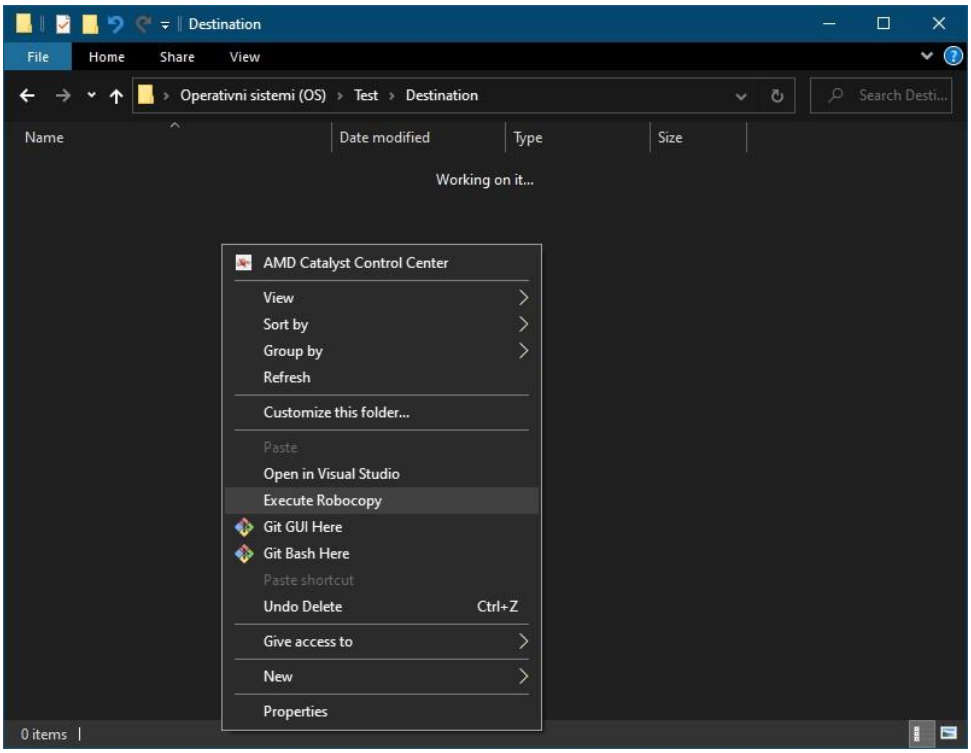

Figure 2 Execute Robocopy Script

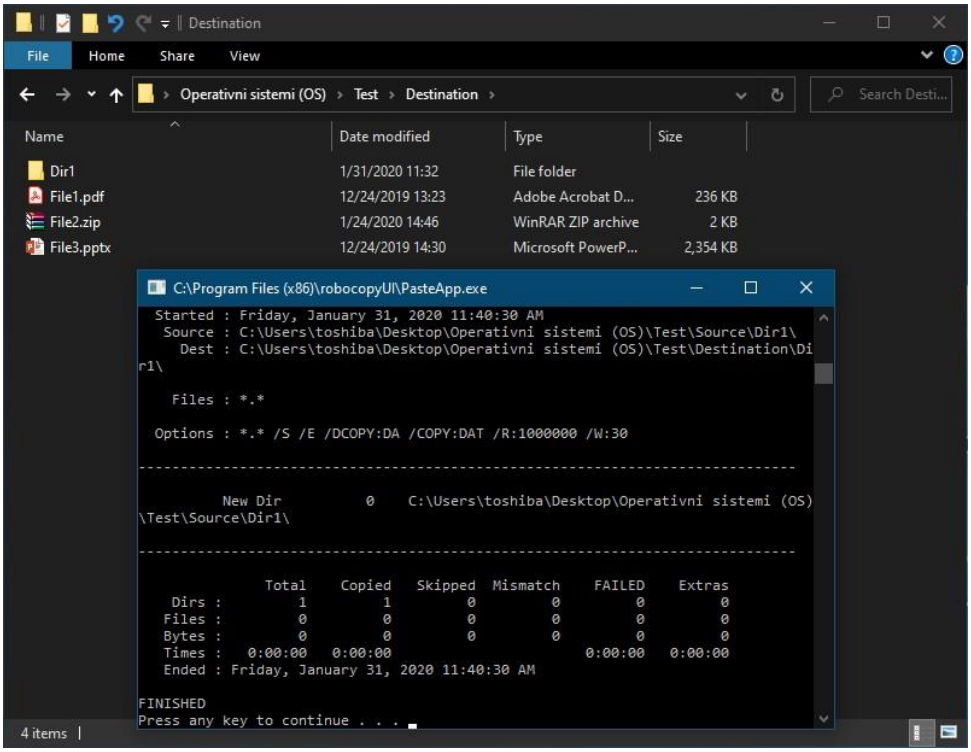

Figure 3 Monitor Robocopy output

In the example shown on figures above, three files and one directory are selected: files "File1.pdf", "File2.zip", "File3.pptx" and directory "Dir1". For these items the following robocopy script will be generated:

```
robocopy "C:\Users\toshiba\Desktop\Operativni sistemi (OS)\Test\Source"
"C:\Users\toshiba\Desktop\Operativni sistemi (OS)\Test\Destination" "File1.pdf"
"File2.zip" "File3.pptx"
robocopy "C:\Users\toshiba\Desktop\Operativni sistemi (OS)\Test\Source\Dir1"
"C:\Users\toshiba\Desktop\Operativni sistemi (OS)\Test\Destination\Dir1" /e
```

The first robocopy command copies selected files, and the second copies directory named "Dir1". If we were to select "Dir2" the script would have one additional robocopy command for this directory. For every additional directory we select there will be an additional robocopy command to copy that directory.

In general, if we copy files "File_1", …, "File_n" and directories "Dir_1", …, "Dir_m" from `<Source>` to `<Destination>` we would get:

```
robocopy "<Source>" "<Destination>" "File_1" "File_2" … "File_n"
robocopy "<Source>\Dir_1" "<Destination>\Dir_1" /e
robocopy "<Source>\Dir_2" "<Destination>\Dir_2" /e
…
robocopy "<Source>\Dir_m" "<Destination>\Dir_m" /e
```

3. Implementation

robocopyUI tool consists of three apps: *CopyApp*, *ClipboardApp* and *PasteApp*. The apps communicate using a named memory-mapped file (MMF). Name of the memory mapped file and names of other named synchronization objects are defined in header named `"constants.h"` which is included in all three apps. CopyApp writes to MMF the path of `<Source>` directory, number of selected items and filenames of selected items, each in a separate line. PasteApp reads the contents of MMF, generates and executes robocopy script. ClipboardApp is a single instance app, meaning at most one instance is running at all times. It's sole purpose is to hold a handle (pointer) to MMF at all times and prevent system from removing it from RAM. Significant performance advantage when writing large numbers of characters is the main reason we use MMF over named pipes, message queues and traditional files. The flow of execution is illustrated on the figure below. Numbers one to four (in blue boxes) denote the order of execution.

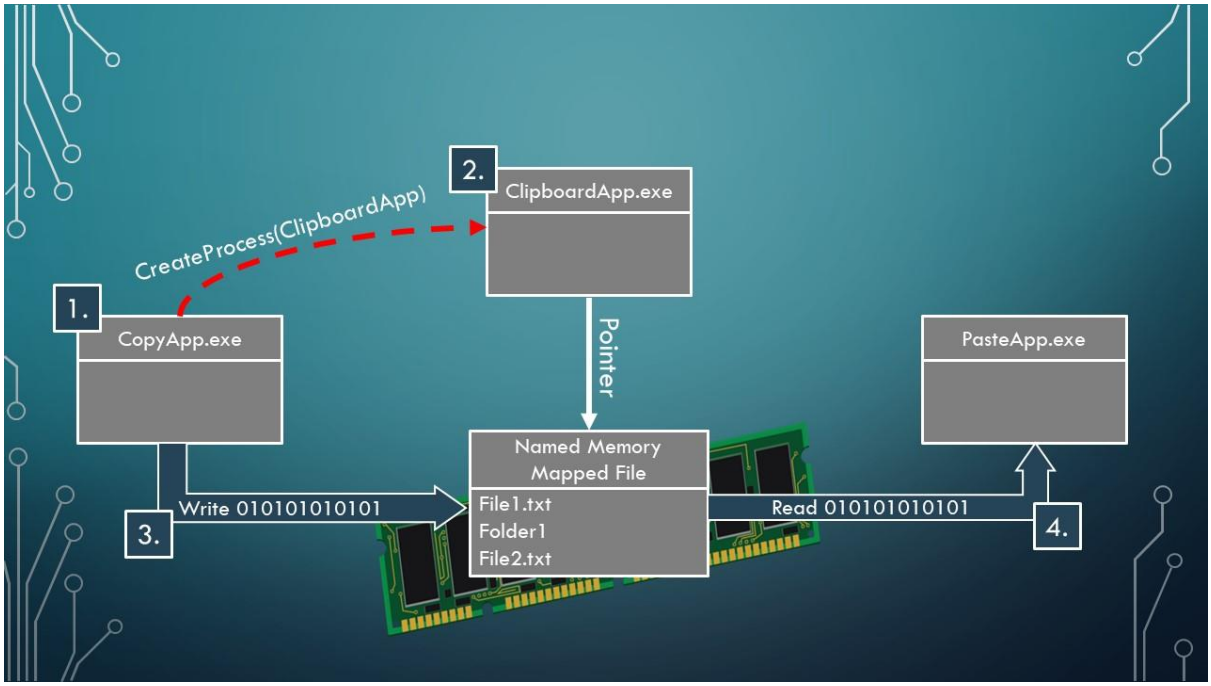

Figure 4 robocopyUI's flow of execution

- Step 1 – When we click "SendTo → Robocopy Script Generator", an instance of CopyApp is started.
- Step 2 – CopyApp creates an instance of ClipboardApp. If an instance of ClipboardApp is already running, the instance we just created simply exits. If not, newly created instance aquires a handle (pointer) to MMF.
- Step 3 – CopyApp writes to MMF the path of `<Source>` directory, number of selected items and filenames of those items, each in a separate line.
- Step 4 – When we click "Execute robocopy" PasteApp is started. It generates and executes the robocopy script. Path of `<Destination>` folder is passed to PasteApp as a command line argument. Explanation of how this is possible will be discussed

together with explanation how to add options to context menu ("SendTo → Robocopy Script Generator" and "Execute robocopy" options).

There are three synchronization issuse we must take care of. Firstly, ClipboardApp must aquire a handle to MMF before CopyApp exits. Secondly, only one instance of ClipboardApp may run at any moment. And as third, we must have reader-writer synchronization between CopyApp and PasteApp when reading and writing to MMF.

First issue we solve quite neatly using named event object which is shared between CopyApp and ClipboardApp. We use `CreateEvent()` in CopyApp to create the named event (if it's not already there) in unsignaled state and store the return handle in variable named `hAquiredMMFHandleEvent`. CopyApp then waits for event to be set to signalled state. If an instance of ClipboardApp is already running, the event will already be in a signalled state. If not, a newly created instance of ClipboardApp then creates the MMF and sets the named event to signalled state.

Named mutex is used to solve the secont issue of making ClipboardApp a one-instance app. The mutex is named `hCBAMutex` in the ClipboardApp. The code is self-explanatory.

Instead of a usual reader-writer synchronization which grants shared access to readers and exclusive access to writers, we use exclusive access for both readers and writers. Main reason for such a decision is the fact that reading and writing to MMF usually takes a fraction of a second. Even when we select hundreads or thousands of items, reading/writing to MMF takes less than a second, which is insignificant compared to the time needed to execute the robocopy script. As with ClipboardApp, a named mutex (variable named `hMMFMutex`) is used by CopyApp and PasteApp to aquire exclusive access to MMF.

Here we describe what actions must be taken in order to add "Robocopy Script Generator" to "SendTo" menu and "Execute Robocopy" to "right-click" menu. They are added by adding keys and commands to Windows registry (regedit.exe). These will be done automatically during installation (see *InstallataionScript.nsi*).

- RoboCopy – add shortcut to RoboCopy.exe to SendTo folder, which is at: "C:\Users\<username>\AppData\Roaming\Microsoft\Windows\SendTo" and name the shortcut "Robocopy Script Generator"
- RoboPaste – add 2 keys (*RoboPaste* and *command*) to Windows registry:

```
1) Computer\HKEY_CLASSES_ROOT
          Directory
                  Background
                          Shell
                          + RoboPaste

2)  Computer\HKEY_CLASSES_ROOT
          Directory
                  Background
                          Shell
                                  RoboPaste
                                  + command
                                  @default = "<PasteApp.exe path>" "%v"
```

default is an attribute of key *command* and is of type REG_SZ. Set its value to "<PasteApp.exe path>" "%v". "PasteApp.exe" path should point to the installation folder of *robocopyUI*. %v is replaced with the path of directory where the user made a right-click.

4. Support and Feedback

For any issues you encounter or friendly suggestions you may have, please contact me at "*mr16006@alas.matf.bg.ac.rs*"