APPENDIX - I

ANALYSIS OF USER’S PROSPECTIVE ACTION

In this appendix users actions on Non-Self processes are discussed. The proposed approaches, Decision Tree Learning, Artificial Neural Learning and Genetic Algorithm; are used to identify the Non-Self operating system processes. After the identification of Non-Self processes by these approaches, the user of the computer system is informed about these Non-Self processes; that in your computer system following processes are identified as Non-Self. Then the user can take action on these Non-Self processes. In this appendix the analysis of the action taken against these Non-Self processes are explained.

A.1 Introduction

This appendix is the detail description of the module 4 “Action Taken Against the Non-Self Processes” explained in section of 3.2.5 of chapter 3 “Framework for Identification of Non-Self Operating System Process”. An action environment is provided to the user where he or she can select the action to be taken against the Non-Self processes. The Actions that a user can take includes:-

- Forced deletion of the Non-Self Process.
- Suspension of working of the Non-Self Process.

Algorithm of module 4

- Step 1: Take one process from the list of Non-Self processes.
- Step 2: Ask the user whether he wants to delete this process or not.
– Step 2.1: If the user wants to delete the Non-Self process, then delete the process and go to Step No.3

– Step 2.2: If the user does not want to delete the Non-Self process under consideration, then the process is left in the Non-Self processes list with its working being suspended.

• Step 3: Go for another process from the list of Non-Self processes.

• Step 4 Go to Step 2.

• Step 5: Repeat until the user is asked for his choice for all the Non-Self processes

Figure 3.5 of chapter 3 shows the user’s action on Non-Self processes in the form of flow chart. User can delete or suspend the Non-Self processes by using the various commands with the help of some tools.

Approaches of chapter 4, 5 and 6 provide a detailed list of Non-Self processes. The computer user takes one by one process from the list, if the user thinks that this process is not a Non-Self process, then the user can leave this process and a system call is generated to suspend its working. If the user thinks that the process is non-self, then he/she can kill that process using the kill command.

A.2 Possible Actions on OS Process

There are various methods to stop the execution of operating system processes. Processes may also be terminated by the system for a variety of reasons, including Security reasons also. To kill a process in a computer operating system, there are various functions and methods depending upon the operating system running on the computer.

In general a Kill or Terminate command are used to stop the execution of a process by many operating systems. These commands are used with various syntaxes and types by different operating system. In section A.3 to A.5 of this appendix these Kill and Termination methods are explained for various operating systems. In response to a KILL command, or other un-handled
process interrupts. A parent may kill its children if the task assigned to them is no longer needed. If the parent exits, the system may or may not allow the child to continue without a parent.

When a process terminates, all of its system resources are freed up, open files flushed and closed, etc. The process termination status and execution times are returned to the parent if the parent is waiting for the child to terminate, or eventually returned to init if the process becomes an orphan. Processes which are trying to terminate but which cannot because their parent is not waiting for them are termed zombies. These are eventually inherited by init as orphans and killed off.

A.3 End to The Process in Windows Operating System

To end a windows operating system process is a very easy task. As the user has a detailed list of Non-Self processes identified by the approaches of chapter 4, 5 and 6; the user presses Ctrl + Alt + Delete keys from the keyboard. After pressing these keys a window of Security with various options comes out.

![Windows Security](image.png)

**Figure A.1:** Windows after pressing Ctrl+Alt+Delete
There are six choices in this window as shown in Figure A.1 first user can lock the computer, second user can log off from the current login, third user can shut down the computer system, fourth user can change the password of user login, fifth user can choose the option task manager and last and sixth option is cancelled the security window and come to normal operation. To end a process execution the user has to select the fifth option task manager.

After selecting this option the user sees a window of windows task manager as shown in Figure A.3. User can find the details i.e. Process Name, User Name, CPU time, Memory Usage from the list generated by the proposed approach of chapter 4, 5 and 6; the user takes one by one Non-Self processes and end the execution of these processes with the help of window task manager as shown in Figure A.2.

Figure A.2: Window of Task Manager
Processes may be terminated by making the exit( ) system call, typically returning an int. This int is passed along to the parent if it is doing a wait( ), and is typically zero on successful completion and some non-zero code in the event of problems.

Child code:

    int exitCode;

    exit( exitCode ); // return exitCode; has the same effect when executed from main( )

Parent code:

    pid_t pid;

    int status

    pid = wait( &status );

    // pid indicates which child exited. exitCode in low-order bits of status

    // macros can test the high-order bits of status for why it stopped

**Process.Kill Method- Win32Exception**

Kill forces a termination of the process, while CloseMainWindow only requests a termination. When a process with a graphical interface is executing, its message loop is in a wait state. The message loop executes every time a Windows message is sent to the process by the operating system. Calling CloseMainWindow sends a request to close the main window, which, in a well-formed application, closes child windows and revokes all running message loops for the application.

The request to exit the process by calling CloseMainWindow does not force the application to quit. The application can ask for user’s verification before quitting, or it can refuse to quit. To force the application to quit, use the Kill method. The behavior of CloseMainWindow is identical to that of a user closing an application's main window using the system menu. Therefore, the
request to exit the process by closing the main window does not force the application to quit immediately.

A.4 The Ending of a Process in Unix Operating System

After getting the list of Non-Self process list, if the user using the Unix operating system, he or she can kill Unix Processes by using the KILL command. Ending a process can be done in several different ways. Often, from a console-based command, sending a CTRL + C keystroke (the default interrupt character) will exit the command.

This works when the process is running in foreground mode. If a process is running in background mode, then first you would need to get its Job ID using the ps command and after that you can use KILL command to kill the process as follows:

```
$ps -f
UID PID PPID C STIME TTY TIME CMD
amrood 6738 3662 0 10:23:03 pts/6 0:00 first_one
amrood 6739 3662 0 10:22:54 pts/6 0:00 second_one
amrood 3662 3657 0 08:10:53 pts/6 0:00 -ksh
amrood 6892 3662 4 10:51:50 pts/6 0:00 ps -f
$kill 6738
Terminated
```

The Killing or Termination of a Process in Linux / UNIX Systems

In Unix operating system, user can kill Unix Processes by using the KILL command. To kill a Unix process following statements can be used.

```
kill [signal] PID
kill -15 PID
kill -9 PID
kill -SIGTERM PID
```
```
kill [options] -SIGTERM PID

# kill 3486     //3486 is pid  Or      $ sudo kill 3486
```

To kill two or more processes PIDs as required can be used in a single command, the syntax is as follows

```
kil pid1 pid2 pid3
kill -15 pid1 pid2 pid3
kill -9 pid1 pid2 pid3
kill -9 3546 5557 4242
```

### A.5 Actions on Non-Self Process of Other Operating Systems

**Android** – for an android application System.exit(0) and finish() can be used to exit the current activity. When the finish method is used, it has not closed the complete process. To stop the execution of process use the code” android.os.Process.killProcess(android.os.Process.myPid());” in Main Activity.

**Mac OS** - Mac OS is the public release of the Apple operating system, which includes various unique features not found in any other operating system. To kill a Mac OS process anyone can use top command or open Applications > Utilities > Activity Monitor and kill it from there. On Mac OS, there's Cmd/Alt/Escape, which reliably shows running processes.

You can use Activity Monitor to kill pesky apps or processes that just won’t close. Launch Activity Monitor from ~/Applications/Utilities and make sure the drop-down menu at the top of the window says “All Processes.” Next, use either the search box or browse the list alphabetically to find the process(es) associated with application. Select each one and press “Quit Process.”
Hit Command+Option+Escape to bring up the simple “Force Quit Applications” window, then click to select the application name, followed by clicking the “Force Quit” button to end the application immediately. Think of this as a simplified version of Activity Monitor, and it’s also a great keystroke to remember to use since it allows for quickly ceasing multiple apps.

Using the command line is a surefire way to force an application or process to quit by issuing the low-level kill command. Launch the Terminal and type one of the following commands: killall [processname], kill -9 [pid]. For example, “killall Safari” would kill all instances of the Safari process. If you know the process id then you can use the second command.

**IBM OS/2**- Jointly developed by Microsoft and IBM to operate with Intel microprocessors, OS/2 was originally a 16-bit operating system that was designed to work with 286 processors and first introduced in 1987. OS/2 later became a graphical interface similar to Windows, but also supported a command line. In fact, many OS/2 and DOS commands are the same.

To kill a process in IBM OS/2 DosKillProcess command is used by its process ID. What happens is that a KILLPROCESS exception is sent to the given process or group of processes. This by default will write all file buffers, and handles opened by the process will be closed. It is possible to intercept the exception with the use of DosSetExceptionHandler, and then the process will make its own nice exit and call DosExit. If another process has issued a DosWaitChild for a process terminated it will get termination code "Unintercepted DosKillProcess"

**AmigaOS** - AmigaOS is the proprietary native operating system of the Amiga personal computer. It was developed first by Commodore International and introduced with the launch of the first Amiga, the Amiga 1000, in 1985. Early versions of AmigaOS required the Motorola 68000 series of 16-bit and 32-bit microprocessors.

Later versions were developed by Haage & Partner (AmigaOS 3.5 and 3.9) and then Hyperion Entertainment (AmigaOS 4.0-4.1). A PowerPC microprocessor is required for the most recent release, AmigaOS 4. Command Line Interface for user with which he can interact with AmigaOS processes. To remove an AmigaOS process the ENDSHELL command is used. Ctrl-C, sends a BREAK command to the current process (halts the process).
**Symbian**- Symbian was a closed-source mobile operating system and computing platform designed for smartphones and currently maintained by Accenture. Symbian was originally developed by Symbian Ltd., as a descendant of Psion’s EPOC and runs exclusively on ARM processors, although an unreleased x86 port existed. The current form of Symbian is an open-source platform developed by Symbian Foundation in 2009, as the successor of the original Symbian OS. Symbian was used by many major mobile phone brands, like Samsung, Motorola, Sony Ericsson, and above all by Nokia. It was the most popular smartphone OS on a worldwide average until the end of 2010.

To kill a process in Symbian OS by calling RProcess::Kill() or Terminate(). The method takes one integer argument – a code to indicate the reason the process was terminated through a return value. Your program will need the PowerMgmt extended capability and be Symbian Signed in order to perform a Kill() on a process. Otherwise the function will return KErrPermissionDenied(-46). RProcess::Panic(const TDesc& aCategory, TintReson) also terminates a process. Panic() indicate that some unrecoverable error was detected.

![System Performance vs Number of Non-Self Processes Killed](image)

**Figure A.3:** System Performance vs Number of Non-Self Processes Killed.
A.6 Summary

There is no effect on computer performance if user kills or end the execution of a process. If a process killed, then it’s free all of system resources. These resources can be used by the other Self processes and the system performance will increase. Figure A.4 shows how the system performance increases when a user ends the Non-Self processes executions. If a user thinks that the process identified as Non-Self is not harmful, then the user may leave this process for its normal execution.