**Exercise 2 – ping checker – processes and Threads**

**Code document**

/\***File Name : main.c**

Authors: Roi Toledano 203700505

Yarden Avraham 302249925

Project Name: Exercise 2 - Ping - Processes and Threads

Using: Site\_Thread.c,Site\_Thread.h

Description: This is the main module of the project.

Manage the all process.

read addresses and open for each address a Thread for all the ping checks. \*/

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*Includes and Defines\*/

#include "Site\_Thread.h"

#define ERROR\_CODE ((int)(-1))

#define SUCCESS\_CODE ((int)(0))

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*void ThreadsValuesChecks

Parameters: ptr\_thread\_params - A pointer to a struct that contains the thread parameters.

size - Number of threads.

Returns: No returns - a void function.

Description: The function checks all the Error flags ans the ping status for each thread.

If there is an Error - the function prints a relevant note to the cmd

Moreover, the function prints to cmd if the address is reachable or not.\*/

void ThreadsValuesChecks(SITE\_THREAD\_params \*ptr\_thread\_params, int size) {

for (int jj = 0; jj < size; jj++) {

printf("%s", ptr\_thread\_params[jj].address);

if (ptr\_thread\_params[jj].error\_waiting) printf("Waiting for thread - ERROR\n");

if (ptr\_thread\_params[jj].error\_closing\_handle) printf("Closing handle - ERROR\n");

if (ptr\_thread\_params[jj].error\_process\_creation) printf("Process creation - ERROR\n");

if (ptr\_thread\_params[jj].error\_id\_null) printf("Thread ID is NULL - ERROR\n");

if (ptr\_thread\_params[jj].error\_thread\_creation) printf("Thread creation - ERROR\n");

if (ptr\_thread\_params[jj].error\_alloc) printf("Allocation - ERROR\n");

if (ptr\_thread\_params[jj].ping\_successfully\_passed) printf(" reachable\n");

else printf(" unreachable\n");

}

}

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*int main

Parameters: argc - number of strings ("commands") in cmd.

argv[] - A pointer to the strings array from cmd.

Returns: Success/Fail - An integer : 0 = success (-1) = Fail

Description: The Main of the project. Gets an array of adresses an opens for each address a thread that handles Pings.

waits for threads to finish and prints the results.\*/

int main(int argc, char \*argv[]) {

//parameters

int jj;

HANDLE \*ptr\_thread\_handle = NULL;

DWORD \*ptr\_thread\_id = NULL;

DWORD \*ptr\_exitcode = NULL;

SITE\_THREAD\_params \*ptr\_thread\_params = NULL;

DWORD \*ptr\_wait\_code = NULL;

DWORD wait\_code\_multiple;

// allocations for the array

ptr\_thread\_handle = (HANDLE\*)malloc((argc - 1) \* sizeof(HANDLE));

ptr\_thread\_id = (DWORD\*)malloc((argc - 1) \* sizeof(DWORD));

ptr\_exitcode = (DWORD\*)malloc((argc - 1) \* sizeof(DWORD));

ptr\_wait\_code = (DWORD\*)malloc((argc - 1) \* sizeof(DWORD));

ptr\_thread\_params = (SITE\_THREAD\_params\*)malloc((argc - 1) \* sizeof(SITE\_THREAD\_params));

// checks if one of the allocations failed.

if (ptr\_thread\_handle == NULL || ptr\_thread\_id == NULL || ptr\_exitcode == NULL || ptr\_thread\_params == NULL) {

printf("Allocating memmory failed\n");

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

for (int jj = 0; jj < argc - 1; jj++) {

ptr\_thread\_params[jj].error\_alloc = true;

}

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

return ERROR\_CODE;

}

//initiate threads parameters

for (int jj = 0; jj < argc - 1; jj++) {

ptr\_thread\_params[jj].address = argv[jj + 1];

ptr\_thread\_params[jj].error\_closing\_handle = false;

ptr\_thread\_params[jj].ping\_successfully\_passed = false;

ptr\_thread\_params[jj].error\_waiting = false;

ptr\_thread\_params[jj].error\_process\_creation = false;

ptr\_thread\_params[jj].error\_alloc = false;

ptr\_thread\_params[jj].error\_thread\_creation = false;

ptr\_thread\_params[jj].error\_id\_null = false;

}

// check if the input from command line is valid

if (argc < 2) {

printf("Not enough addresses entered - Exit program\n");

return ERROR\_CODE;

}

// Create a thread for each address

for (jj = 0; jj < argc - 1; jj++) {

ptr\_thread\_handle[jj] = CreateThreadSimple(SITEThread, &(ptr\_thread\_params[jj]), &(ptr\_thread\_id[jj]));

}

//For parallel functionality - wait that all the threads finish

wait\_code\_multiple = WaitForMultipleObjects(argc - 1, ptr\_thread\_handle, TRUE, INFINITE);

//Checks if one of the threads failed.

for (jj = 0; jj < argc - 1; jj++) {

if (GetExitCodeThread(ptr\_thread\_handle[jj], (&(ptr\_exitcode[jj]))) == 0) {

printf("Error when getting thread number %d with address %s exit code\n", jj, ptr\_thread\_params[jj].address);

return ERROR\_CODE;

}

}

//Checking and Printing results

ThreadsValuesChecks(ptr\_thread\_params, argc - 1);

//Closing all Thread handles

for (jj = 0; jj < argc - 1; jj++) {

if (CloseHandle(ptr\_thread\_handle[jj]) == 0) {

printf("ERROR while closing Thread number %d with address %s\n", jj, ptr\_thread\_params[jj].address);

return ERROR\_CODE;

}

}

//Freeing all allocated memory

free(ptr\_thread\_handle);

free(ptr\_thread\_id);

free(ptr\_wait\_code);

free(ptr\_thread\_params);

free(ptr\_exitcode);

return SUCCESS\_CODE;

}

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF FILE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\***File Name : Site\_Thread.h**

Authors: Roi Toledano 203700505

Yarden Avraham 302249925

Project Name: Exercise 2 - Ping - Processes and Threads

Using: C built-in libraries

Description: Site\_Thread.c header; \*/

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*Includes and Defines\*/

#include <windows.h>

#include <stdbool.h>

#include <stdio.h>

#include "CreatePingProcess.h"

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*Structs defenitions\*/

typedef enum{

SITE\_THREAD\_\_CODE\_SUCCESS,

SITE\_THREAD\_\_CODE\_FAILED,

SITE\_THREAD\_\_CODE\_ERROR = -1

} SITE\_THREAD\_\_return\_code;

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

typedef struct{

int site\_count;

char \*address;

DWORD res;

bool ping\_successfully\_passed;

bool error\_id\_null;

bool error\_process\_creation;

bool error\_closing\_handle;

bool error\_waiting;

bool error\_thread\_creation;

bool error\_alloc;

} SITE\_THREAD\_params;

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

// Functions declarations

DWORD WINAPI SITEThread(LPVOID lpParam);

HANDLE CreateThreadSimple(LPTHREAD\_START\_ROUTINE p\_start\_routine, SITE\_THREAD\_params \*p\_thread\_parameters, DWORD \*p\_thread\_id);

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF FILE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\***File Name : Site\_Thread.c**

Authors: Roi Toledano 203700505

Yarden Avraham 302249925

Project Name: Exercise 2 - Ping - Processes and Threads

Using: C built-in libraries

Description: Site Thread Module creates a new thread for each address we wish to check its ping \*/

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*Includes and Defines\*/

#include <windows.h>

#include "Site\_Thread.h"

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*HANDLE CreateThreadSimple

Parameters: p\_start\_routine - A pointer tha points to a function that notifies the host that a thread has started to execute.

p\_thread\_parameters - A pointer to a struct that contains all the parameters of the thread, including potential errors.

p\_thread\_id - A pointer to the thread's ID.

Returns: A HANDLE variable - The handle of the thread that was created.

Description: A function that calls CreateThread function, that creates a thread to execute within the virtual address space of the

calling process, and checks if the thread was created properly.\*/

HANDLE CreateThreadSimple(LPTHREAD\_START\_ROUTINE p\_start\_routine, SITE\_THREAD\_params \*p\_thread\_parameters, DWORD \*p\_thread\_id){

/\* Parameters \*/

HANDLE thread\_handle;

/\* Checks if the function received a null pointer \*/

if (NULL == p\_start\_routine){

printf("Error when creating a thread\n");

printf("Received null pointer\n");

exit(SITE\_THREAD\_\_CODE\_ERROR);

}

thread\_handle = CreateThread(

NULL, /\* default security attributes \*/

0, /\* use default stack size \*/

p\_start\_routine, /\* thread function \*/

p\_thread\_parameters, /\* argument to thread function \*/

0, /\* use default creation flags \*/

p\_thread\_id); /\* returns the thread identifier \*/

/\* Checks if the thread was created properly \*/

if (thread\_handle == NULL) {

p\_thread\_parameters->error\_thread\_creation = true;

printf("Error when creating a thread\n");

printf("Thread handler is NULL\n");

exit(SITE\_THREAD\_\_CODE\_ERROR);

}

return thread\_handle;

}

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*DHANDLE SITEThread

Parameters: lpParam - LPVOID type, pointer to the thread's parameters, can be any pointer we want.

Returns: A DWORD variable - SITE\_THREAD\_\_CODE\_SUCCESS = 0 , SITE\_THREAD\_\_CODE\_FAILED = 1.

Description: A function that creates a new thread for a process.\*/

DWORD WINAPI SITEThread(LPVOID lpParam)

{

/\* Parameters \*/

SITE\_THREAD\_params \*thread\_params = NULL;

AddressInfo ping\_params;

int ping\_flag = 0;

/\* Check if lpParam is NULL \*/

if (NULL == lpParam)

{

return SITE\_THREAD\_\_CODE\_FAILED;

}

/\* Casting lpParam to be type SITE\_THREAD\_params, a struct we defined and its value is in thread\_params \*/

thread\_params = (SITE\_THREAD\_params \*)lpParam;

/\*Initialazing paramater of AddressInfo\*/

ping\_params.address = thread\_params->address;

ping\_params.exitcode = 1;

ping\_params.allocation\_status = false;

ping\_params.closing\_handle\_status = false;

ping\_params.process\_creation\_status = false;

ping\_params.waiting\_status = false;

/\* First try of ping \*/

thread\_params->res = CreateProcessSimpleMain(&ping\_params);

/\* Checks the ping again if failed the first time \*/

if ((thread\_params->res) != 0) {

if (!ping\_flag) {

ping\_flag = 1;

thread\_params->res = CreateProcessSimpleMain(&ping\_params);

}

}

/\*Updates the Thread's parameters\*/

if (ping\_params.exitcode == 0) {

thread\_params->ping\_successfully\_passed = true;

}

else {

thread\_params->ping\_successfully\_passed = false;

}

thread\_params->error\_closing\_handle = ping\_params.closing\_handle\_status;

thread\_params->error\_waiting = ping\_params.waiting\_status;

thread\_params->error\_process\_creation = ping\_params.process\_creation\_status;

return SITE\_THREAD\_\_CODE\_SUCCESS;

}

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF FILE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\***File Name : CreatePingProcess.h**

Authors: Roi Toledano 203700505

Yarden Avraham 302249925

Project Name: Exercise 2 - Ping - Processes and Threads

Using: C built-in libraries

Description: CreatePingProcess.c header; \*/

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*Includes and Defines\*/

#include <windows.h>

#include <stdio.h>

#include <stdbool.h>

#include <tchar.h> /\* for TCHAR, \_T() \*/

#define TIMEOUT\_IN\_MILLISECONDS 20000

#define BRUTAL\_TERMINATION\_CODE 0x55

#define MAXSIZE 200

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*Structs defenitions\*/

typedef enum{

PING\_PROCESS\_\_CODE\_SUCCESS,

PING\_PROCESS\_\_CODE\_FAILED,

PING\_PROCESS\_\_CODE\_ERROR = -1

} PING\_PROCESS\_return\_code;

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

typedef struct address\_info{

char \*address;

DWORD exitcode;

bool closing\_handle\_status;

bool waiting\_status;

bool allocation\_status;

bool process\_creation\_status;

}AddressInfo;

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*Functions declarations\*/

BOOL CreateProcessSimple(LPTSTR CommandLine, PROCESS\_INFORMATION \*ProcessInfoPtr);

DWORD CreateProcessSimpleMain(AddressInfo \*ping\_params);

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF FILE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\***File Name : CreatePingProcess.c**

Authors: Roi Toledano 203700505

Yarden Avraham 302249925

Project Name: Exercise 2 - Ping - Processes and Threads

Using: CreatePingProcess.h

Description: Ping Process Module creates a new process with a ping command (one of the built in commands in windows) \*/

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*Includes and Defines\*/

#include "CreatePingProcess.h"

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*DWORD CreateProcessSimpleMain

Parameters: ping\_params - A pointer to a struct that contains all the parameters that the process needs for make a ping.

Returns: A DWORD variable - PING\_PROCESS\_\_CODE\_SUCCESS = 0 , PING\_PROCESS\_\_CODE\_FAILED = 1, PING\_PROCESS\_\_CODE\_ERROR = -1.

Description: The main of this mudule: Gets parameters, creates process, checks exit codes, close handles and update values.\*/

DWORD CreateProcessSimpleMain(AddressInfo \*ping\_params)

{

//Parameters

int closed\_thread = -1;

int closed\_process = -1;

PROCESS\_INFORMATION procinfo;

DWORD waitcode;

DWORD exitcode;

BOOL retVal;

TCHAR command[MAXSIZE] = \_T("ping -n 1 "); /\* <ISP> TCHAR is a win32 \*/

/\* generic char which may be either a simple (ANSI) char or a unicode char, \*/

/\* depending on behind-the-scenes operating system definitions. Type LPTSTR \*/

/\* is a string of TCHARs. Type LPCTSTR is a const string of TCHARs. \*/

/\* Start the child process. \*/

// attach the address to the ping command

strcat\_s(command, 50, ping\_params->address);

// Create a Ping Process

retVal = CreateProcessSimple(command, &procinfo);

//Checks if failed

if (retVal == 0)

{

ping\_params->process\_creation\_status = true;

printf("Process Creation Failed!\n");

return PING\_PROCESS\_\_CODE\_ERROR;

}

// Wait the process will finish

waitcode = WaitForSingleObject(procinfo.hProcess, TIMEOUT\_IN\_MILLISECONDS); /\* Waiting 20 secs for the process to end \*/

// Checks the waitcode

switch (waitcode)

{

case WAIT\_TIMEOUT: // Error

ping\_params->waiting\_status = true;

return PING\_PROCESS\_\_CODE\_ERROR;

case WAIT\_OBJECT\_0: // The process finishes correctly

break;

default:

break;

}

/\* If the Process is still alive - kill the project \*/

if (waitcode == WAIT\_TIMEOUT)

{

printf("Process was not terminated before timeout!\nTerminating brutally!\n"); //NEED TO DELETE

TerminateProcess(procinfo.hProcess, BRUTAL\_TERMINATION\_CODE); /\* Terminating process with an exit code of 55h \*/

Sleep(10); /\* Waiting a few milliseconds for the process to terminate \*/

}

GetExitCodeProcess(procinfo.hProcess, &exitcode);

// close handlers

closed\_process = CloseHandle(procinfo.hProcess);/\* Closing the handle to the process \*/

closed\_thread = CloseHandle(procinfo.hThread); /\* Closing the handle to the main thread of the process \*/

//checks if the handler closing failed

if (closed\_process == 0 || closed\_thread == 0) {

ping\_params->closing\_handle\_status = true;

return PING\_PROCESS\_\_CODE\_ERROR;

}

// checks for problems in exitcode

if (exitcode == 0) {

ping\_params->exitcode = 0;

return PING\_PROCESS\_\_CODE\_SUCCESS;

}

else {

return PING\_PROCESS\_\_CODE\_FAILED;

}

}

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*BOOL CreateProcessSimple

Parameters: CommandLine - the command that the process will do.

ProcessInfoPtr - A pointer to a struct that contains all the process parameters ; for update values.

Returns: A BOOL variable - Success = 0 , Fail = 1.

Description: A simple function that calls CreateProcess function.\*/

BOOL CreateProcessSimple(LPTSTR CommandLine, PROCESS\_INFORMATION \*ProcessInfoPtr)

{

STARTUPINFO startinfo = { sizeof(STARTUPINFO), NULL, 0 }; /\* <ISP> here we \*/

/\* initialize a "Neutral" STARTUPINFO variable. Supplying this to \*/

/\* CreateProcess() means we have no special interest in this parameter. \*/

/\* This is equivalent to what we are doing by supplying NULL to most other \*/

/\* parameters of CreateProcess(). \*/

return CreateProcess(

NULL, /\* No module name (use command line). \*/

CommandLine, /\* Command line. \*/

NULL, /\* Process handle not inheritable. \*/

NULL, /\* Thread handle not inheritable. \*/

FALSE, /\* Set handle inheritance to FALSE. \*/

CREATE\_NO\_WINDOW, /\* creation/priority flags. \*/

NULL, /\* Use parent's environment block. \*/

NULL, /\* Use parent's starting directory. \*/

&startinfo, /\* Pointer to STARTUPINFO structure. \*/

ProcessInfoPtr /\* Pointer to PROCESS\_INFORMATION structure. \*/

);

}

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF FILE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*------------------------------------------------------------------------------------------------------------------------------------------\*/