Interprocess Communication

Producer-Consumer Data Sharing

Author:

Alex Reigle

Date:

7/22/2020

Course:

CEG4350 Operation Systems Internals and Design

Professor Soon Chung

Wright State University

Table of Contents:

(1) General Design and Discussion:

The programs that follow implement interprocess communication methods in various ways. The methods implemented are pipe communication, indirect message passing using a mailbox, user datagram protocol sockets, and a binary semaphore. All methods were implemented in on a Windows operating system, using `<Windows.h>` functions. However, each method implements its process using different techniques to show understanding and control of various Server-Client and Parent-Child communication methods. These differences will be discussed in the design and discussion section of each method implemented. In the following sections, the programs written and discussed all generate 100 random integer value ranging from 0 to 100. The data produced and consumed are printed, saved, and identical to show that the processes have successfully communicated without distorting the information being shared or passed.

(2) Implementation One: Pipe Method

(2.1) Pipe Method Design:

This program is written in C++ using the IDE Visual Studio 2019 Community on a Windows 10 Operating System. This program consists of three source files, one parent and two children. The pipe method implemented makes use of a named pipe, `MYNAMEDPIPE`, that passes data from the producer process to the consumer process. Both the server and client can read from and write to the pipe.

In this program, a parent process, `PIPEParent.cpp`, creates two processes that pass the data from the server (producer process) to the client (consumer process). The parent process calls the child server process, `PipeServer.cpp`, using the Windows system call `CreateProcess()` and subsequently creates the child client process, `PipeClient.cpp`, using the same method. This allows the transfer of data across the pipe.

In this program, the data is passed from the producer (server) to the consumer (client) processes. The data itself takes the form of a vector of one hundred integers with random values. The data is written into the pipe buffer and then read by the consumer process. Then the consumer process writes a message into the buffer alerting the server that the data has been consumed. Once the message is written into the buffer the client child process completes and terminates. The server child process reads and prints the message from the buffer, disconnects and closes the pipe, and the terminates. Finally, the parent program completes. For ease of verification, both the producer and consumer processes write the data to a text file specified in the program.

(2.2) Pipe Method Source Code:

PIPEParent.cpp

```
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <stdlib.h>
#include <time.h>
#include <stdexcept>
#include <Windows.h>

using namespace std;

int CallProducerChild();
int CallConsumerChild();

int main()
{

	CallProducerChild();
	CallConsumerChild();

	return 0;
}

int CallProducerChild()
{
	//Initialize Process
	HANDLE hProcess = NULL;
	HANDLE hThread = NULL;
	STARTUPINFO si;
	PROCESS_INFORMATION pi;
	DWORD dwProcessId = 0;
	DWORD dwThreadId = 0;
	ZeroMemory(&si, sizeof(si));
	ZeroMemory(&pi, sizeof(pi));
	BOOL bCreateProcess = NULL;

	bCreateProcess = CreateProcess(

	L"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolution
s\\PIPEs\\PipeServer\\Debug\\PipeServer.exe", //App Name
		NULL, // Command Line
		NULL, // Process Attribute
		NULL, // Thread Attribute
		FALSE,// Inherit Handle
		0,    // Creation Flag
		NULL, // Environment Variable
		NULL, // Current Directory
		&si,  // Startup Info
		&pi   // Process Info
	);
```

```
	if (bCreateProcess == FALSE)
	{
		std::cout << "Producer Process Failed & Error No - " << GetLastError() << 
std::endl;
	}



	CloseHandle(pi.hProcess);
	CloseHandle(pi.hThread);

	return 0;
}
int CallConsumerChild()
{
	//Initialize Process
	HANDLE hProcess = NULL;
	HANDLE hThread = NULL;
	STARTUPINFO si;
	PROCESS_INFORMATION pi;
	DWORD dwProcessId = 0;
	DWORD dwThreadId = 0;
	ZeroMemory(&si, sizeof(si));
	ZeroMemory(&pi, sizeof(pi));
	BOOL bCreateProcess = NULL;

	bCreateProcess = CreateProcess(

	L"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolution
s\\PIPES\\PipeClient\\Debug\\PipeClient.exe", //App Name
		NULL, // Command Line
		NULL, // Process Attribute
		NULL, // Thread Attribute
		FALSE,// Inherit Handle
		0,    // Creation Flag
		NULL, // Environment Variable
		NULL, // Current Directory
		&si,  // Startup Info
		&pi   // Process Info
	);

	if (bCreateProcess == FALSE)
	{
		std::cout << "Consumer Process Failed & Error No - " << GetLastError() << 
std::endl;
	}



	CloseHandle(pi.hProcess);
	CloseHandle(pi.hThread);

	return 0;
}
```

PipeServer.cpp

```
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <stdlib.h>
#include <time.h>
#include <stdexcept>
#include <Windows.h>

using namespace std;

vector<int> Producer();

void newfile(string);
vector<int> generate(int, int);
void write(string, vector<int>);

int main(int argc, char* argv[])
{
	cout << "\t\t....PRODUCER SERVER...." << endl;

	//Produce the Data
	vector<int> data = Producer();
	char dataChar[1023] = "";
	for (int i = 0; i < data.size(); i++)
	{
		dataChar[i] = ('0' + data[i]);		//Convert data into character
values to be printed and transfered across pipe
	}

	//Named Pipe Local Variable
	HANDLE hCreateNamedPipe;
	char   szInputBuffer[1023];
	char   szOutputBuffer[1023];
	DWORD  dwInputBufferSize = sizeof(szInputBuffer);
	DWORD  dwOutputBufferSize = sizeof(szOutputBuffer);

	//ConnectNamedpipe Local Variable
	BOOL   bConnectNamedPipe;

	//WriteFile Local Variable
	BOOL   bWritefile;
	DWORD  dwWiteBufferSize = sizeof(dataChar);	//Print the data being transfered
over the pipe
	DWORD  dwNoBytesWrite;

	//FlushBuffer Local Variables
	BOOL bFlushFileBuffer;

	//ReadFile Local Variable
	BOOL   bReadfile;
	char   szReadFileBuffer[1023];
	DWORD  dwReadBufferSize = sizeof(szReadFileBuffer);
	DWORD  dwNoBytesRead;
```

```
	//CreateNamedPipe
	hCreateNamedPipe = CreateNamedPipe(
		L"\\\\.\\pipe\\MYNAMEDPIPE",
	//Pipe Name
		PIPE_ACCESS_DUPLEX,
	//Defines that we can both read and write at the same time.
		PIPE_TYPE_MESSAGE | PIPE_READMODE_MESSAGE | PIPE_WAIT, //Pipe Mode
		PIPE_UNLIMITED_INSTANCES,
	//Maximum Instances of Pipe (up to 256 pipe instances)
		dwOutputBufferSize,
	//Output Buffer Size
		dwInputBufferSize,
	//Input Buffer Size
		0,
			//Timeout (Doesn't Time Out)
		NULL
		//Security Parameter
	);
	if (hCreateNamedPipe == INVALID_HANDLE_VALUE) { cout << "Server: CreateNamedPipe
Failed with Error # " << GetLastError() << endl; }
	//else { cout << "Server: CreateNamedPipe Success." << endl; }

	//ConnectNamedPipe
	bConnectNamedPipe = ConnectNamedPipe(hCreateNamedPipe, NULL);
	if (bConnectNamedPipe == FALSE) { cout << "Server: ConnectNamedPipe Failed with
Error # " << GetLastError() << endl; }
	//else { cout << "Server: ConnectNamePipe Success." << endl; }



	//WriteFile Operation
	bWritefile = WriteFile(
		hCreateNamedPipe,
		dataChar,
		dwWiteBufferSize,
		&dwNoBytesWrite,
		NULL
	);
	if (bWritefile == FALSE) { cout << "Server: WriteFile Failed with Error # " <<
GetLastError() << endl; }
	//else { cout << "Server: WriteFile Success." << endl; }


	//Flush the File Buffer
	bFlushFileBuffer = FlushFileBuffers(hCreateNamedPipe);
	if (bFlushFileBuffer == FALSE) { cout << "Server: FlushFileBuffer Failed with
Error # " << GetLastError() << endl; }
	//else { cout << "Server: FlushFileBuffer Success." << endl; }


	//ReadFile Operation
	bReadfile = ReadFile(
		hCreateNamedPipe,
		szReadFileBuffer,
		dwReadBufferSize,
		&dwNoBytesRead,
		NULL
```

```
	);
	if (bReadfile == FALSE) { cout << "Server: ReadFile Failed with Error # " <<
GetLastError() << endl; }
	else
	{
		cout << endl << "\t\t....PRODUCER SERVER...." << endl;
		cout << "DATA READING FROM CLIENT -> " << szReadFileBuffer << endl;
	}

	//Disconnect NamedPipe
	DisconnectNamedPipe(hCreateNamedPipe);

	//CloseHandle
	CloseHandle(hCreateNamedPipe);

	return 0;
}

vector<int> Producer()
{
	cout << "Producer Process Entered.\n";

	int size = 100;			// Number of randomly generated integers
	int upLim = 100;	// Upper limit to range of randomly generated integers.
	string writeLocation =
"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolutions\\PIPES
";
	string file = writeLocation + "\\ProducerFile.txt"; //File produced, saving the
data
	vector<int> data(size);

	newfile(file); //Create a new file to save data
	data = generate(upLim, size); //Generate size number of random integers between 0
and uplim
	write(file, data); //Print data and Write it to a file

	return data;
}

void newfile(string path1)
{
	ofstream a;
	a.open(path1);
	a.close();
}
vector<int> generate(int lim, int sz)
{
	vector<int> data(sz);

	srand(time(NULL));
	for (int idx = 0; idx < sz; idx++)
	{
		data[idx] = rand() % lim + 1;
	}

	return data;
}
void write(string path, vector<int> data)
```

```
{
	fstream file;
	file.open(path);

	int idx = 0;
	int sz = data.size();
	cout << "Data Written to Pipe" << endl;
	while (idx < sz)
	{
		file << data[idx] << endl;
		cout << data[idx] << " ";
		idx++;
	} cout << endl;
	file.close();

}
```

PipeClient.cpp

```
#pragma once

#include <iostream>
#include <Windows.h>
#include <fstream>
#include <vector>
#include <string>
#include <stdlib.h>
#include <time.h>
#include <stdexcept>

using namespace std;

int Consumer(vector<int>);
void newfile(string);
void write(string, vector<int>);

int main(int argc, char* argv[])
{
	cout << "\n\t\t....CONSUMER CLIENT...." << endl;

	//Local Variables
	HANDLE hCreateFile;

	//ReadFile Local Variables
	BOOL   bReadFile;
	DWORD  dwNoBytesRead;
	char   szReadFileBuffer[1023];
	DWORD  dwReadFileBufferSize = sizeof(szReadFileBuffer);

	//WriteFile Local Variables
	BOOL   bWriteFile;
	DWORD  dwNoBytesWrite;
	char   szWriteFileBuffer[1023] = "Client Consumed Data from Pipe";
	DWORD  dwWriteFileBufferSize = sizeof(szWriteFileBuffer);

	//CreatFile for Pipe
	hCreateFile = CreateFile(
```

```
		L"\\\\.\\pipe\\MYNAMEDPIPE",
		GENERIC_READ | GENERIC_WRITE,
		0,
		NULL,
		OPEN_EXISTING,
		FILE_ATTRIBUTE_NORMAL,
		NULL
	);
	if (hCreateFile == INVALID_HANDLE_VALUE) { cout << "Client: CreateFile Failed with
Error # " << GetLastError() << endl; }
	//else { cout << "Client: CreateFile Success" << endl; }

	//ReadFile Operation
	bReadFile = ReadFile(
		hCreateFile,
		szReadFileBuffer,
		dwReadFileBufferSize,
		&dwNoBytesRead,
		NULL
	);
	if (bReadFile == FALSE) { cout << "Client: ReadFile Failed with Error # " <<
GetLastError() << endl; }
	else
	{
		//cout << "Client: ReadFile Success" << endl;
	}

	//Convert Characters to Integer Values (get Data from pipe)
	char bufferMsgAdjusted[1023];
	int size = 0;
	for (int i = 0; i < sizeof(szReadFileBuffer); i++)
	{
		bufferMsgAdjusted[i] = (szReadFileBuffer[i] - '0');
		if (bufferMsgAdjusted[i] != -48) { size++; }
	}
	//Print Data Read from Pipe
	vector<int> data(size);                                          cout << "Data
Read from Pipe  -> ";
	for (int idx = 0; idx < size; idx++)
	{
		data[idx] = (bufferMsgAdjusted[idx]);          cout << data[idx] << " ";

	}
	cout << endl;


	//Consume Data
	Consumer(data);

	//WriteFile Operation
	bWriteFile = WriteFile(
		hCreateFile,
		szWriteFileBuffer,
		dwWriteFileBufferSize,
		&dwNoBytesWrite,
		NULL
	);
```

```
	if (bWriteFile == FALSE) { cout << "Client: WriteFile Failed with Error # " << 
GetLastError() << endl; }
	//else { cout << "Client: WriteFile Success" << endl; }

	//Close Handles
	CloseHandle(hCreateFile);

	return 0;

}

int Consumer(vector<int> data)
{
	cout << "Consumer Process Entered.\n";

	string writeLocation = 
"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolutions\\PIPES
";
	string pfile = writeLocation + "\\ProducerFile.txt"; //File read
	string cfile = writeLocation + "\\ConsumerFile.txt"; //File produced


	newfile(cfile); //Open File to save data read from pipe
	write(cfile, data); //Write data to file from pipe

	return 0;
}

void newfile(string path1)
{
	ofstream a;
	a.open(path1);
	a.close();
}
void write(string path, vector<int> data)
{
	fstream file;
	file.open(path);

	int idx = 0;
	int sz = data.size();
	while (idx < sz)
	{
		file << data[idx] << endl;
		idx++;
	}
	file.close();

}
```

(2.3) Pipe Method Results:

The following image shows the results of the source code above.

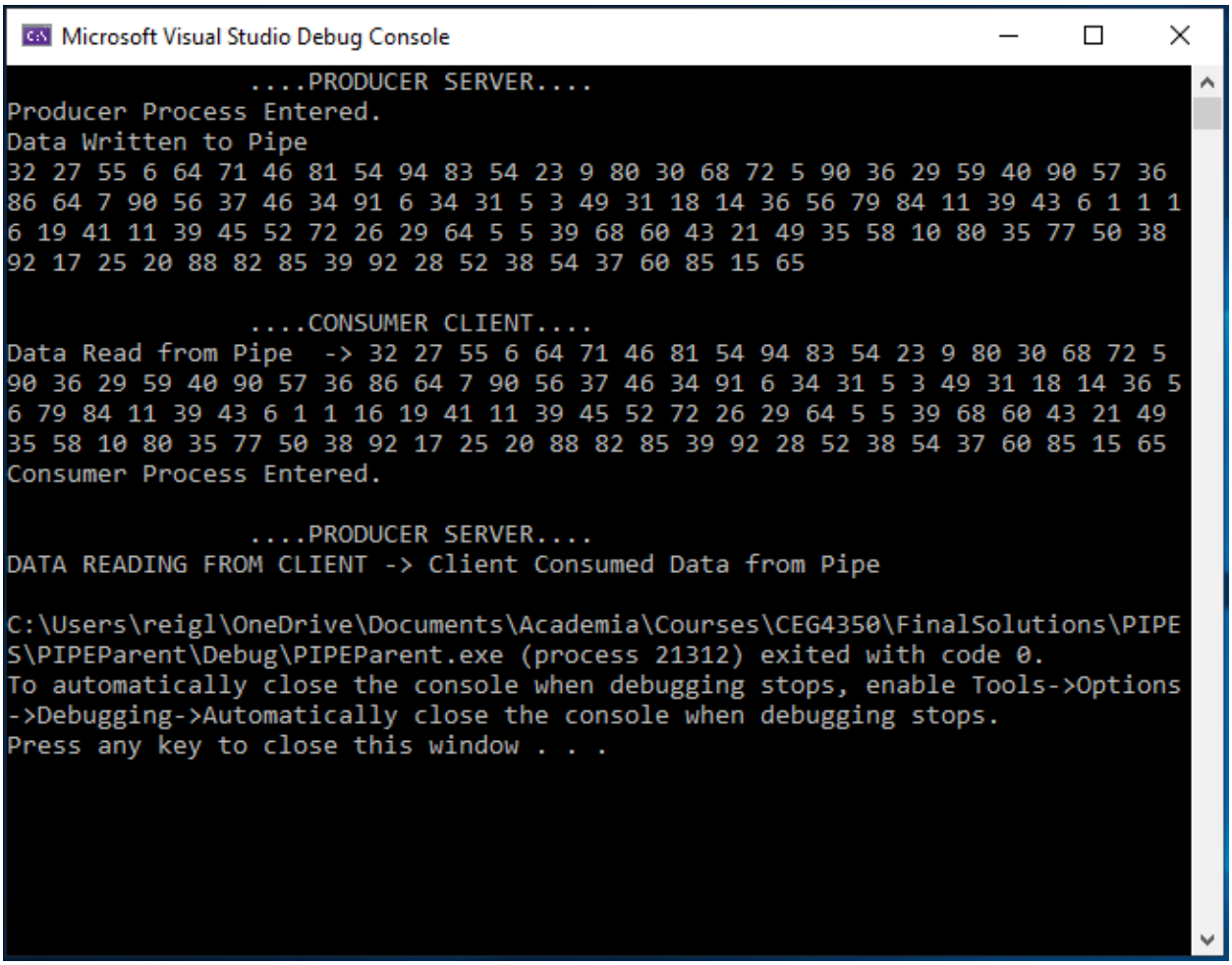

Figure 1: Results of IPC Method 1: Named Pipe Method

(3) Implementation Two: Mailbox Method

(3.1) Mailbox Method Design:

This program is written in C++ using the IDE Visual Studio 2019 Community on a Windows 10 Operating System. This program consists of three source files, one parent and two children. The IPC method implemented in this program is indirect message passing by way of a mailbox, `MYMAILSLOT`. The owner process, `MailslotServer.cpp`, receives a message from the user process, `MailslotClient.cpp`.

In this program a parent process, `MailslotParent.cpp`, creates two child processes that will pass data from the mailbox user (the producer) process to the mailbox owner (the consumer) process. The data produced is a vector of one hundred integers with random values. The data is converted from integer to character data type and placed in the buffer. The producer then prints, saves, and writes the data to the buffer before completing and terminating. The consumer then reads the message from the mailbox and converts it into a vector of integers. Then, the consumer prints and saves the data before completing and terminating.

(3.2) Mailbox Method Source Code:

MailslotParent.cpp

```
#pragma once

#include <iostream>
#include <stdio.h>
#include <windows.h>
#include <fstream>
#include <vector>
#include <string>
#include <stdlib.h>
#include <time.h>
#include <stdexcept>

int Producer();
int Consumer();

int main()
{
    Consumer();
    Sleep(1000);
    Producer();

    return 0;
}

int Producer()
{
```

```
    //Initialize Process
    HANDLE hProcess = NULL;
    HANDLE hThread = NULL;
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    DWORD dwProcessId = 0;
    DWORD dwThreadId = 0;
    ZeroMemory(&si, sizeof(si));
    ZeroMemory(&pi, sizeof(pi));
    BOOL bCreateProcess = NULL;

    bCreateProcess = CreateProcess(

L"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolutions\\MAIL
BOXES\\MailslotClient\\Debug\\MailslotClient.exe", //App Name
        NULL, // Command Line
        NULL, // Process Attribute
        NULL, // Thread Attribute
        FALSE,// Inherit Handle
        0,    // Creation Flag
        NULL, // Environment Variable
        NULL, // Current Directory
        &si,  // Startup Info
        &pi   // Process Info
    );
    if (bCreateProcess == FALSE)
    {
        std::cout << "Create Producer Process Failed & Error No - " << GetLastError() <<
std::endl;
    }


    WaitForSingleObject(pi.hProcess, INFINITE); //Wait for process to complete and close
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);

    return 0;
}
int Consumer()
{

    //Initialize Process
    HANDLE hProcess = NULL;
    HANDLE hThread = NULL;
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    DWORD dwProcessId = 0;
    DWORD dwThreadId = 0;
    ZeroMemory(&si, sizeof(si));
    ZeroMemory(&pi, sizeof(pi));
    BOOL bCreateProcess = NULL;

    bCreateProcess = CreateProcess(

L"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolutions\\MAIL
BOXES\\MailslotServer\\Debug\\MailslotServer.exe", //App Name
        NULL, // Command Line
```

```
        NULL, // Process Attribute
        NULL, // Thread Attribute
        FALSE,// Inherit Handle
        0,    // Creation Flag
        NULL, // Environment Variable
        NULL, // Current Directory
        &si,  // Startup Info
        &pi   // Process Info
    );

    if (bCreateProcess == FALSE)
    {
        std::cout << "Consumer Process Failed & Error No - " << GetLastError() <<
std::endl;
    }


    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);

    return 0;
}
```

MailslotServer.cpp

```
#pragma once
#include <Windows.h>
#include <iostream>
#include <vector>
#include <string>
#include <fstream>

using namespace std;

void Consumer(vector<int>);
void newfile(string);
int write(string, vector<int>);

int main()
{
    cout << "\t\t-----MAILSLOTS SERVER-----" << endl;
    //CreateMailslots Local Variables
    HANDLE hSlots;
    //ReadFile Local Variables
    BOOL    bReadFile;
    DWORD   dwNoBytesRead;
    char    szReadBuffer[1023];
    DWORD   dwReadFileBufferSize = sizeof(szReadBuffer);

    //CreateMailslots
    hSlots = CreateMailslot(
        L"\\\\.\\mailslot\\MYMAILSLOT",
        0,
        MAILSLOT_WAIT_FOREVER,
        NULL
    );
```

```
    if (hSlots == INVALID_HANDLE_VALUE) { cout << "CreateMailslot Failed with Error # "
<< GetLastError() << endl; }
    else { cout << "CreateMailslot Success." << endl; }


    //ReadFile
    bReadFile = ReadFile(
        hSlots,
        szReadBuffer,
        dwReadFileBufferSize,
        &dwNoBytesRead,
        NULL
    );
    if (bReadFile == FALSE) { cout << "ReadFile Failed with Error # " << GetLastError()
<< endl; }
    else { cout << "ReadFile Success." << endl; }

    //Convert char[] -> vector<int>
    vector<int> data(100);
    for (int i = 0; i < data.size(); i++)
    {
        data[i] = szReadBuffer[i];
    }

    cout << "\t\t-----MAILSLOTS SERVER-----" << endl;

    //Consume
    Consumer(data);

    CloseHandle(hSlots);
    return 0;
}

void Consumer(vector<int> data)
{
    string file =
"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolutions\\MAILB
OXES\\Consumer.txt";
    newfile(file); //Create file for saving data
    write(file, data); //Save and Print data
}
int write(string path, vector<int> data)
{
    newfile(path);
    fstream file;
    file.open(path);

    int idx = 0;
    int sz = data.size();
    cout << "Consumer Data: " << endl;
    while (idx < sz)
    {
        cout << data[idx] << " ";
        file << data[idx] << endl;
        idx++;
    }cout << endl;
    file.close();
```

```
    return 0;
}
void newfile(string path)
{
    ofstream a;
    a.open(path);
    a.close();
}
```

MailslotClient.cpp

```
#pragma once
#include <Windows.h>
#include <iostream>
#include <vector>
#include <time.h>
#include <fstream>

using namespace std;

vector<int> Producer(int, int);
void newfile(string);
int write(string, vector<int>);

int main()
{
    cout << "\t\t-----MAILSLOTS CLIENT-----" << endl;
    //CreateFile Local Variable
    HANDLE hCreateFile;

    //WriteFile Local Variables
    BOOL    bWriteFile;
    DWORD   dwNoBytesWrite;


    //CreateFile
    hCreateFile = CreateFile(
        L"\\\\.\\mailslot\\MYMAILSLOT",
        GENERIC_READ | GENERIC_WRITE,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL
    );
    if (hCreateFile == INVALID_HANDLE_VALUE) { cout << "CreateFile Failed with Error # "
<< GetLastError() << endl; }
    else { cout << "CreateFile Success." << endl; }

    //Produce Data
    vector<int> data(100);
    data = Producer(100, 100);
    char    szWriteFileBuffer[1023];
    DWORD   dwWriteFileBufferSize = sizeof(szWriteFileBuffer);
    for (int i = 0; i < data.size(); i++)
    {
```

```
        szWriteFileBuffer[i] = data[i]; //Write data to message to be placed in mailslot
    }
    string file =
"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolutions\\MAILB
OXES\\Producer.txt";
    write(file, data); //Save and Print data

    //WriteFile
    bWriteFile = WriteFile(
        hCreateFile,
        szWriteFileBuffer,
        dwWriteFileBufferSize,
        &dwNoBytesWrite,
        NULL
    );
    if (bWriteFile == FALSE) { cout << "WriteFile Failed with Error # " << GetLastError()
<< endl; }
    else { cout << "WriteFile Success." << endl; }

    CloseHandle(hCreateFile);
    return 0;
}

vector<int> Producer(int sz, int lim)
{
    vector<int> data(100);
    srand(time(NULL));
    for (int idx = 0; idx < sz; idx++)
    {
        data[idx] = rand() % lim + 1;
    }


    return data;
}

int write(string path, vector<int> data)
{
    newfile(path);
    fstream file;
    file.open(path);

    int idx = 0;
    int sz = data.size();
    cout << "Producer Data: " << endl;
    while (idx < sz)
    {
        cout << data[idx] << " ";
        file << data[idx] << endl;
        idx++;
    }cout << endl;
    file.close();

    return 0;
}
void newfile(string path)
{
    ofstream a;
```

```
    a.open(path);
    a.close();
}
```

(3.3) Mailbox Method Results:

The following image shows the results of the source code above.

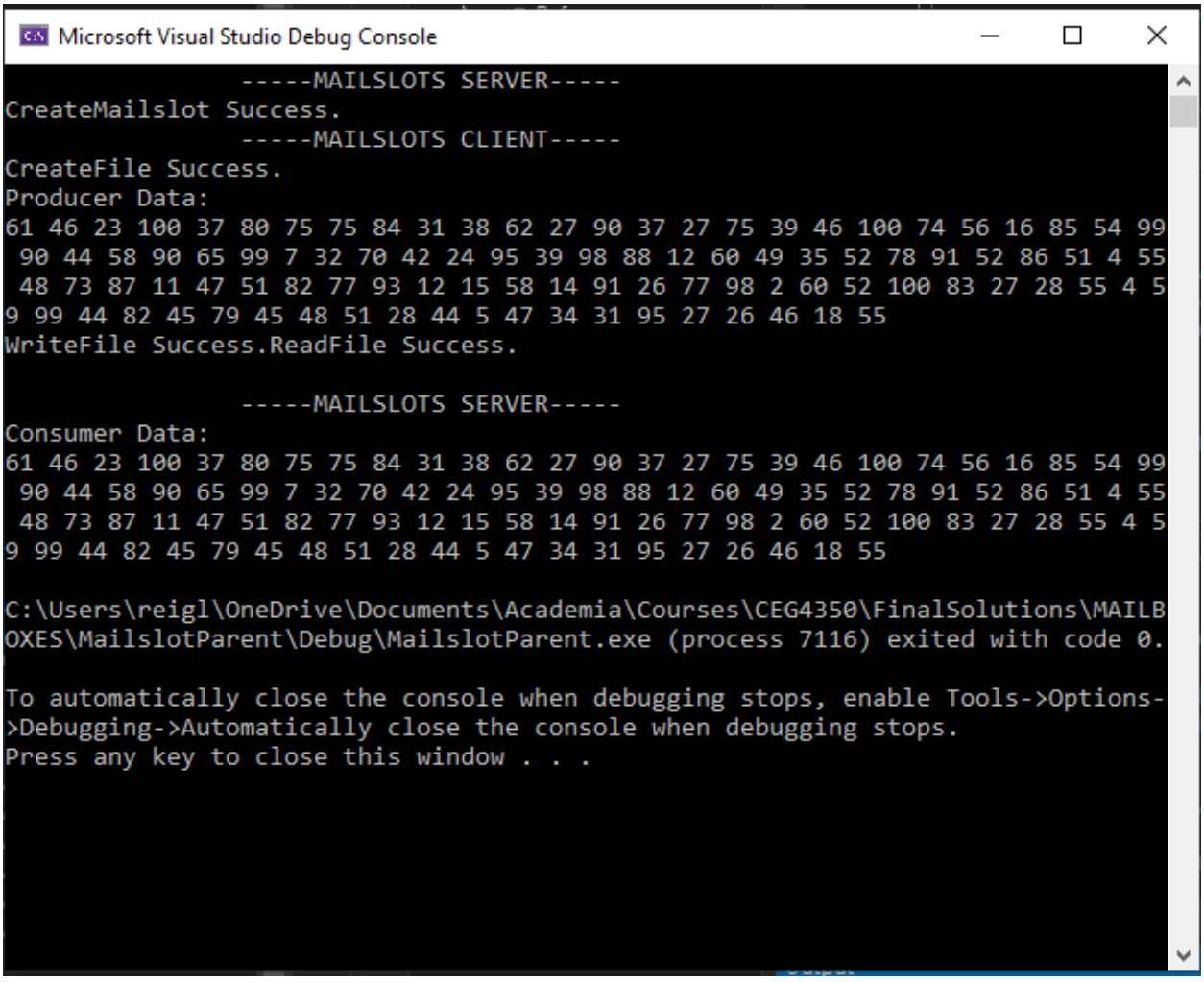

Figure 2: Results of IPC Method 2: Indirect Message Passing via Mailbox Method

(4) Implementation Three: Socket Method

(4.1) Socket Method Design:

This program is written in C++ using the IDE Visual Studio 2019 Community on a Windows 10 Operation System. This program consists of three source files, one parent and two children. The IPC method implemented in this program is a user datagram protocol (UDP) socket, where the data is passed from the socket client, `SocketClient.cpp`, (the producer process) to the socket server, `SocketServer.cpp`, (the consumer process). Since UDP sockets do not have an initial send and receive communication to ensure a valid data path (like TCP sockets), I have implemented an error handling functionality with `GetLastError()` and `SOCKET_ERROR`.

In this program a parent process, `SocketParent.cpp`, creates two child processes that establish themselves as the UDP socket server and UDP socket client with the same port value (port 1025). The socket client (producer) passes data to the server client (consumer). The data consists of a vector of one hundred random integers that are converted to character values while they are stored in the datagram buffer and converted back once the information has been received. The producer passes the data to the consumer, prints and saves the data, and then completes and terminates. Similarly, the consumer initializes the buffer and receives the data from the socket client before printing and saving the data and, finally, completing and terminating the process.

(4.2) Socket Method Source Code:

SocketParent.cpp

```
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <stdlib.h>
#include <time.h>
#include <stdexcept>
#include <Windows.h>

using namespace std;

int CallProducerChild();
int CallConsumerChild();

int main()
{
	CallConsumerChild();
	CallProducerChild();
```

```
	return 0;
}

int CallProducerChild()
{
	//Initialize Process
	HANDLE hProcess = NULL;
	HANDLE hThread = NULL;
	STARTUPINFO si;
	PROCESS_INFORMATION pi;
	DWORD dwProcessId = 0;
	DWORD dwThreadId = 0;
	ZeroMemory(&si, sizeof(si));
	ZeroMemory(&pi, sizeof(pi));
	BOOL bCreateProcess = NULL;

	bCreateProcess = CreateProcess(

	L"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolution
s\\SOCKET\\SocketClient\\Debug\\SocketClient.exe", //App Name
		NULL, // Command Line
		NULL, // Process Attribute
		NULL, // Thread Attribute
		FALSE,// Inherit Handle
		0,    // Creation Flag
		NULL, // Environment Variable
		NULL, // Current Directory
		&si,  // Startup Info
		&pi   // Process Info
	);

	if (bCreateProcess == FALSE)
	{
		std::cout << "Consumer Process Failed & Error No - " << GetLastError() <<
std::endl;
	}


	//WaitForSingleObject(pi.hProcess, INFINITE); //wait for process to complete and
close
	CloseHandle(pi.hProcess);
	CloseHandle(pi.hThread);

	return 0;
}
int CallConsumerChild()
{
	//Initialize Process
	HANDLE hProcess = NULL;
	HANDLE hThread = NULL;
	STARTUPINFO si;
	PROCESS_INFORMATION pi;
	DWORD dwProcessId = 0;
	DWORD dwThreadId = 0;
	ZeroMemory(&si, sizeof(si));
	ZeroMemory(&pi, sizeof(pi));
	BOOL bCreateProcess = NULL;
```

```
	bCreateProcess = CreateProcess(

	L"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolution
s\\SOCKET\\SocketServer\\Debug\\SocketServer.exe", //App Name
		NULL, // Command Line
		NULL, // Process Attribute
		NULL, // Thread Attribute
		FALSE,// Inherit Handle
		0,    // Creation Flag
		NULL, // Environment Variable
		NULL, // Current Directory
		&si,  // Startup Info
		&pi   // Process Info
	);

	if (bCreateProcess == FALSE)
	{
		std::cout << "Consumer Process Failed & Error No - " << GetLastError() <<
std::endl;
	}


	//WaitForSingleObject(pi.hProcess, INFINITE); //wait for process to complete and
close
	CloseHandle(pi.hProcess);
	CloseHandle(pi.hThread);

	return 0;
}
```

SocketServer.cpp

```
#pragma once
#include <Windows.h>
#include <iostream>
#include <winsock.h>
#include <fstream>
#include <vector>
#include <string>
#include <stdlib.h>
#include <time.h>

using namespace std;

int Consumer(vector<int>);
void newfile(string);
void write(string, vector<int>);

int main()
{
    cout << "\t\t ------UDP SERVER------" << endl << endl;

    //Local Variable Definitions
    WSADATA     WinSockData;
    int         iWsaStartup;
    int         iWsaCleanup;
```

```
    SOCKET      UDPSocketServer;
    struct      sockaddr_in UDPClient;

    char        Buffer[512];
    int         iBufferLen = strlen(Buffer) + 1;

    int         iBind;
    int         iReceiverFrom;

    int         iUDPClientLen = sizeof(UDPClient);
    int         iCloseSocket;

    //WSAStartUp
    iWsaStartup = WSAStartup(MAKEWORD(2, 2), &WinSockData);
    if (iWsaStartup != 0) { cout << "WSAStartUp function Failed." << endl; }
    //else { cout << "WSAStartup function Success." << endl; }

    //Fill UDPClient(socket address) Struct
    UDPClient.sin_family = AF_INET;
    UDPClient.sin_addr.s_addr = inet_addr("127.0.01");
    UDPClient.sin_port = htons(1025);     //Convert Port Number from Litle Endian to Big
Endian

    //Socket Creation
    UDPSocketServer = socket(
        AF_INET,     //Address Family Type (Internet)
        SOCK_DGRAM,  //Type of Socket (Datagram)
        IPPROTO_UDP  //Protocol Name (UDP)
    );
    if (UDPSocketServer == INVALID_SOCKET) { cout << "Socket Creation Failed with Error #
" << WSAGetLastError() << endl; }
    //else { cout << "Socket Creation Success." << endl; }

    //Bind server to socket
    iBind = bind(
        UDPSocketServer,        //nameof socket
        (SOCKADDR*)&UDPClient,  //type cast of the address of the socket (since it
resides client-side)
        sizeof(UDPClient)
    );
    if (iBind == SOCKET_ERROR) { cout << "Binding Failed with Error # " <<
WSAGetLastError() << endl; }
    //else { cout << "Binding Success." << endl; }

    //Receive Data from Client
    iReceiverFrom = recvfrom(
        UDPSocketServer,
        Buffer,
        iBufferLen,
        MSG_PEEK,
        (SOCKADDR*)&UDPClient,
        &iUDPClientLen
    );
    if (iReceiverFrom == SOCKET_ERROR) { cout << "Receiving Failed with Error # " <<
WSAGetLastError() << endl; }
    //else { cout << "Receiving Success." << endl; }
```

```
    //Convert Characters to Integer Values (get Data from pipe)
    char bufferMsgAdjusted[512];
    int size = 0;
    for (int i = 0; i < sizeof(Buffer); i++)
    {
        bufferMsgAdjusted[i] = (Buffer[i] - '0');
        if (bufferMsgAdjusted[i] != -48 && bufferMsgAdjusted[i] != -100) { size++; }
    }
    vector<int> data(size);
    for (int idx = 0; idx < size; idx++)
    {
        data[idx] = (bufferMsgAdjusted[idx]);

    }


    //Consume Data
    Consumer(data);

    //closesocket
    iCloseSocket = closesocket(UDPSocketServer);
    if (iCloseSocket == SOCKET_ERROR) { cout << "Close Socket Failed with Error # " <<
WSAGetLastError() << endl; }
    //else { cout << "Close Socket Success." << endl; }

    //WSACleanUp
    iWsaCleanup = WSACleanup();     //Uses Winsock 2 DLL (Ws2_32.dll)
    if (iWsaCleanup == SOCKET_ERROR) { cout << "WSA CleanUp Failed with Error # " <<
WSAGetLastError() << endl; }
    //else { cout << "WSA CleanUp Success." << endl; }

    return 0;
}

int Consumer(vector<int> data)
{
    cout << "Consumer Process Entered.\n";

    string writeLocation =
"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolutions\\SOCKE
T";
    string pfile = writeLocation + "\\ProducerFile.txt"; //File read
    string cfile = writeLocation + "\\ConsumerFile.txt"; //File produced


    newfile(cfile); //Open File to save data read from pipe
    write(cfile, data); //Write to file & Print data from pipe

    return 0;
}

void newfile(string path1)
{
    ofstream a;
    a.open(path1);
    a.close();
}
void write(string path, vector<int> data)
```

```
{
    fstream file;
    file.open(path);

    int idx = 0;
    int sz = data.size();

    cout << "Data Read From Socket" << endl;
    while (idx < sz)
    {
        file << data[idx] << endl;
        cout << data[idx] << " ";
        idx++;
    } cout << endl;
    file.close();

}
```

SocketClient.cpp

```
#pragma once
#include <Windows.h>
#include <iostream>
#include <winsock.h>
#include <fstream>
#include <vector>
#include <string>
#include <time.h>

using namespace std;

vector<int> Producer();

void newfile(string);
vector<int> generate(int, int);
void write(string, vector<int>);

int main()
{
    cout << "\t\t------UDP Client------" << endl << endl;

    //Produce the Data
    vector<int> data = Producer();
    char dataChar[512] = "";
    for (int i = 0; i < data.size(); i++)
    {
        dataChar[i] = ('0' + data[i]);          //Convert data into character values to
be printed and transfered across pipe
    }

    //Local Variables
    WSADATA     WinSockData;
    int         iWsaStartup;
    int         iWsaCleanup;

    SOCKET      UDPSocketClient;
    struct      sockaddr_in     UDPServer;
```

```
    //char        Buffer[512]     = "Hello from Client!";
    int         iSendto;

    //int         iBufferLen      = strlen(Buffer) + 1;
    int         iBufferLen = strlen(dataChar) + 1;
    int         iUDPServerLen   = sizeof(UDPServer);
    int         iCloseSocket;

    //WSAStartUp
    iWsaStartup = WSAStartup(MAKEWORD(2, 2), &WinSockData);
    if (iWsaStartup != 0) { cout << "WSAStartUp function Failed." << endl; }
    //else { cout << "WSAStartup function Success." << endl; }

    //Fill UDPServer(socket address) Struct
    UDPServer.sin_family = AF_INET;
    UDPServer.sin_addr.s_addr = inet_addr("127.0.01");
    UDPServer.sin_port = htons(1025);     //Convert Port Number from Litle Endian to Big
Endian


    //Socket Creation
    UDPSocketClient = socket(
        AF_INET,
        SOCK_DGRAM,
        IPPROTO_UDP
    );
    if (UDPSocketClient == INVALID_SOCKET) { cout << "Socket Creation Failed with Error #
" << WSAGetLastError() << endl; }
   // else { cout << "Socket Creation Success." << endl; }

    //Send Data to Server
    iSendto = sendto(
        UDPSocketClient,
        dataChar, //Buffer,
        iBufferLen,
        MSG_DONTROUTE,
        (SOCKADDR*)&UDPServer,
        sizeof(UDPServer)
    );
    if(iSendto == SOCKET_ERROR) { cout << "Sending Failed with Error # " <<
WSAGetLastError() << endl; }
    //else { cout << "Sending Success." << endl; }

    //closesocket
    iCloseSocket = closesocket(UDPSocketClient);
    if (iCloseSocket == SOCKET_ERROR) { cout << "Close Socket Failed with Error # " <<
WSAGetLastError() << endl; }
    //else { cout << "Close Socket Success." << endl; }

    //WSAStartUp
    iWsaCleanup = WSACleanup();        //Uses Winsock 2 DLL (Ws2_32.dll)
    if (iWsaCleanup == SOCKET_ERROR) { cout << "WSA CleanUp Failed with Error # " <<
WSAGetLastError() << endl; }
    //else { cout << "WSA CleanUp Success." << endl; }

    return 0;
}
```

```
vector<int> Producer()
{
    cout << "Producer Process Entered.\n";

    int size = 100;         // Number of randomly generated integers
    int upLim = 100; // Upper limit to range of randomly generated integers.
    string writeLocation =
"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolutions\\SOCKE
T";
    string file = writeLocation + "\\ProducerFile.txt"; //File produced, saving the data
    vector<int> data(size);

    newfile(file); //Create a new file to save data
    data = generate(upLim, size); //Generate size number of random integers between 0 and
uplim
    write(file, data); //Print data and Write it to a file

    return data;
}

void newfile(string path1)
{
    ofstream a;
    a.open(path1);
    a.close();
}
vector<int> generate(int lim, int sz)
{
    vector<int> data(sz);

    srand(time(NULL));
    for (int idx = 0; idx < sz; idx++)
    {
        data[idx] = rand() % lim + 1;
    }

    return data;
}
void write(string path, vector<int> data)
{
    fstream file;
    file.open(path);

    int idx = 0;
    int sz = data.size();

    cout << "Data Written to Socket" << endl;
    while (idx < sz)
    {
        file << data[idx] << endl;
        cout << data[idx] << " ";
        idx++;
    } cout << endl;
    file.close();

}
```

(4.3) Socket Method Results:

The following image shows the results of the source code above.

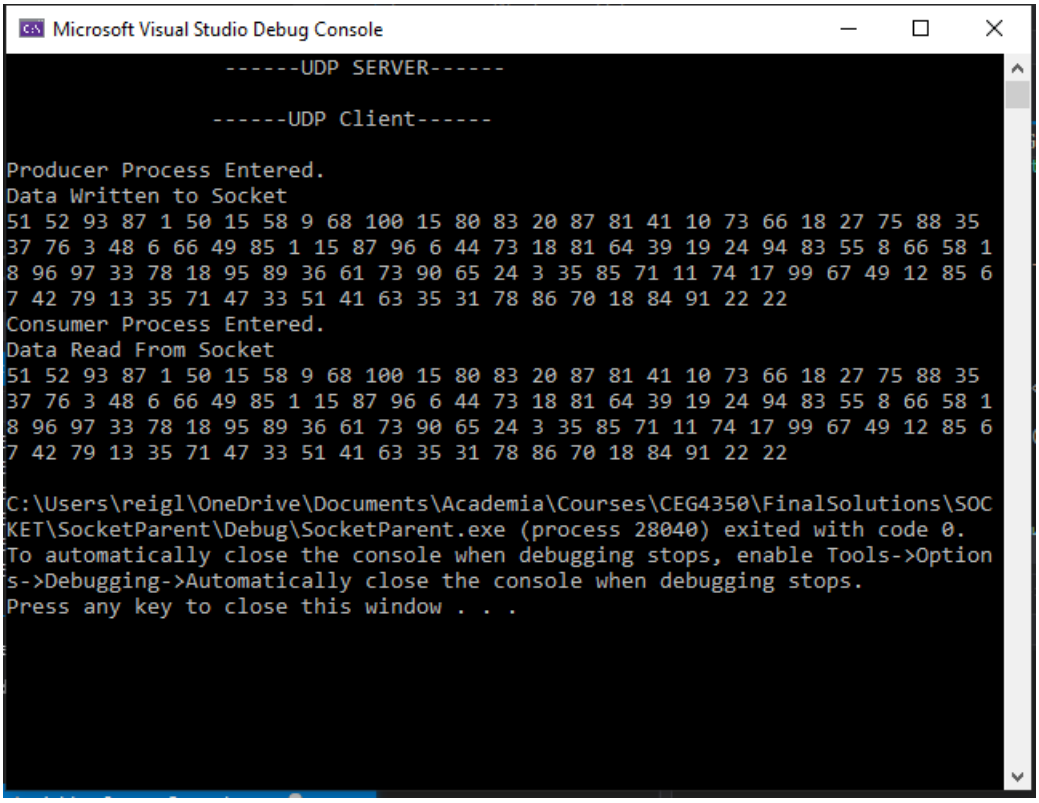

Figure 3: Results of IPC Method 3: User Datagram Protocol Socket Method

(5) Implementation Three: Semaphore Method

(5.1) Semaphore Method Design:

This program is written in C++ using the IDE Visual Studio 2019 Community on a Windows 10 Operating System. This program consists of only one source file that creates two threads. The IPC method implements a named binary semaphore and a logical ring-buffer, with the help of shared memory variables to ensure synchronization.

In this program two threads are created using `CreateThread()`, one thread serves as the producer (`ThreadProduce`) and the other serves as the consumer (`ThreadConsume`). The producer thread creates a random integer and places that value in the next open slot in the buffer. The open slot location (denoted as `head`) is then incremented to the next position in the circular buffer. The producer also places the value of the data item into a vector of integers for later saving and printing. The consumer thread reads an integer from the next slot of the buffer that was first filler and resets the slot's value to `-1`. (This implies that this is a first in first out IPC method.) The read location (denoted as `tail`) is then incremented to the next position in the circular buffer. The consumer also places the value of the data item into a vector of integers for later saving and printing. Once all one hundred data items have been consumed, both threads and the semaphore are closed and the data produced and consumed are then saved and printed before the completion of the program.

Due to the rate at which this program is able to execute both the producer and consumer thread functions, the random number generation capability begins to break down. This occurs because the `srand(time(NULL))` function (used in conjunction with the `rand()` function) is being seeded with a null-time value that is close enough in time to be negligible, and therefor the function produces the same number repeatedly. In this program, I have inserted the command `Sleep(1000)`, which delays the programs progress by one second (1000 milliseconds). This allows enough time to pass to ensure a more random variable is produced. This also results in the execution of the program taking a little more than 100 seconds.

(5.2) Semaphore Method Source Code:

SemaphoreServer.cpp

```
#pragma once
#include <Windows.h>
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <stdlib.h>
#include <time.h>
#include <stdexcept>
```

```
using namespace std;

//Global Variables
int ConsumedIntegers = 0;
int head, tail = 0;  //Logical Ring Buffer Variables
int buffer[10] = { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 };
vector<int> dataProduced(100); int P = 0;
vector<int> dataConsumed(100); int C = 0;


int Producer();
int Consumer(int);
void NewFile(string);
void WriteToFile(int, vector<int>);
void WriteToBuffer(int, int&);
int ReadFromBuffer(int&);
bool IsFull();
bool IsEmpty(int&, int&);

HANDLE hSemaphore;

DWORD WINAPI ThreadProduce(LPVOID lpParam)
{
    while (ConsumedIntegers < 100)
    {
        WaitForSingleObject(hSemaphore, INFINITE);
        if (!IsFull()) //Check that there are slots to place produced data
        {
            //Produce Data
            int dataP = Producer();
            WriteToBuffer(dataP, head);
            //cout << "Entered Producer: " << ConsumedIntegers << " head = " << head << "
tail = " << tail << " Data Produced:" << dataP << endl;
            Sleep(1000);                    //The Sleep() function is used to delay the
program by 1 second each time the Producer() is called.
        }                                   //This gives time(NULL), which is the seed to
srand(), time to change value and produce a more random int.
        ReleaseSemaphore(hSemaphore, 1, 0);
    }
    return 0;
}
DWORD WINAPI ThreadConsume(LPVOID lpParam)
{
    while (ConsumedIntegers < 100)
    {
        WaitForSingleObject(hSemaphore, INFINITE);
        if (!IsEmpty(head, tail)) //Check that there is data to consume
        {
            int dataC = ReadFromBuffer(tail);

            //Cosume Data
            Consumer(dataC);
            ConsumedIntegers++;
            //cout << "Entered Consumer: " << ConsumedIntegers << " head = " << head << "
tail = " << tail << endl;
        }
        ReleaseSemaphore(hSemaphore, 1, 0);
    }
```

```
    return 0;
}

int main()
{
    cout << "\t\t----- SEMAPHORE IPC -----" << endl;

    //Local Variable
    HANDLE hThreadP, hThreadC;

    //Create Semaphore
    hSemaphore = CreateSemaphore(
        NULL,           //Security Attribute
        1,              //Initial Count
        1,              //Max Count
        L"MYSEMAPHORE"  //Semaphore Name
    );
    if (hSemaphore == NULL) { cout << "CreateSemaphore Failed with Error # " <<
GetLastError() << endl; }
   // else { cout << "CreateSemaphore Success." << endl; }

    //Create Threads
    hThreadP = CreateThread(
        NULL,           //Security Attribute
        0,              //Stack Size (Default)
        &ThreadProduce, //Start Function
        NULL,           //Thread Parameter
        0,              //Creation Flags
        0               //ThreadID
    );
    hThreadC = CreateThread(NULL, 0, &ThreadConsume, NULL, 0, 0 );

    //Wait for Signaled Object
    WaitForSingleObject(hThreadP, INFINITE);
    WaitForSingleObject(hThreadC, INFINITE);

    //Close Thread Handles
    CloseHandle(hThreadP);
    CloseHandle(hThreadC);

    //Close Semaphore Handle
    CloseHandle(hSemaphore);

    WriteToFile(1, dataProduced);
    WriteToFile(2, dataConsumed);
    return 0;
}

int Producer()
{
    srand(time(NULL));
    int data = rand() % 100 + 1;
    if (P > 99) {} //Index of data must be in range
    else
    { //Produced Data
        dataProduced[P] = data;
        P++;
    }
```

```
    return data;
}
int Consumer(int val)
{
    if (C > 99) {} //Index of data must be in range
    else
    { //Consumed Data
        dataConsumed[C] = val;
        C++;
    }

    return 0;
}

void WriteToFile(int Flag, vector<int> data)
{ //Save and Print the data
    if (Flag == 1)
    { //Producer Data
        string path =
"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolutions\\SEMAP
HORE\\Producer.txt";
        NewFile(path);
        fstream file;
        file.open(path);
        int sz = data.size();
        cout << "\nData Produced: " << endl;
        for (int i = 0; i < sz; i++)
        {
            cout << data[i] << " ";
            file << data[i] << endl;

        } cout << endl;
        file.close();
    }
    else if (Flag == 2)
    {//Consumer Data
        string path =
"C:\\Users\\reigl\\OneDrive\\Documents\\Academia\\Courses\\CEG4350\\FinalSolutions\\SEMAP
HORE\\Consumer.txt";
        NewFile(path);
        fstream file;
        file.open(path);
        int sz = data.size();
        cout << "\nData Consumed: " << endl;
        for (int i = 0; i < sz; i++)
        {
            cout << data[i] << " ";
            file << data[i] << endl;

        } cout << endl;
        file.close();
    }
    else
    {
        cout << "WriteToFile Error - Incorrect File Path." << endl;
    }
}
void NewFile(string file)
```

```
{
    ofstream a;
    a.open(file);
    a.close();
}


void WriteToBuffer(int val, int& head)
{
    buffer[head] = val;
    head = (head+1)%10;
}
int ReadFromBuffer(int& tail)
{
    int val = buffer[tail];
    buffer[tail] = -1;
    tail = (tail +1)%10;
    return val;
}
bool IsFull()
{
    int emptyslots = 0;
    for (int i = 0; i < sizeof(buffer); i++)
    {
        if (buffer[i] == -1) { emptyslots++; }
    }

    if (emptyslots == 0) { return true; }
    else { return false; }
}
bool IsEmpty(int &head, int &tail)
{
    int emptyslots = 0;
    for (int i = 0; i < sizeof(buffer); i++)
    {
        if (buffer[i] == -1) { emptyslots++; }
    }

    if (head == tail) { return true; }
    else if (emptyslots == 10) { return true; }
    else { return false; }
}
```

(5.3) Semaphore Method Results:

The following image shows the results of the source code above.

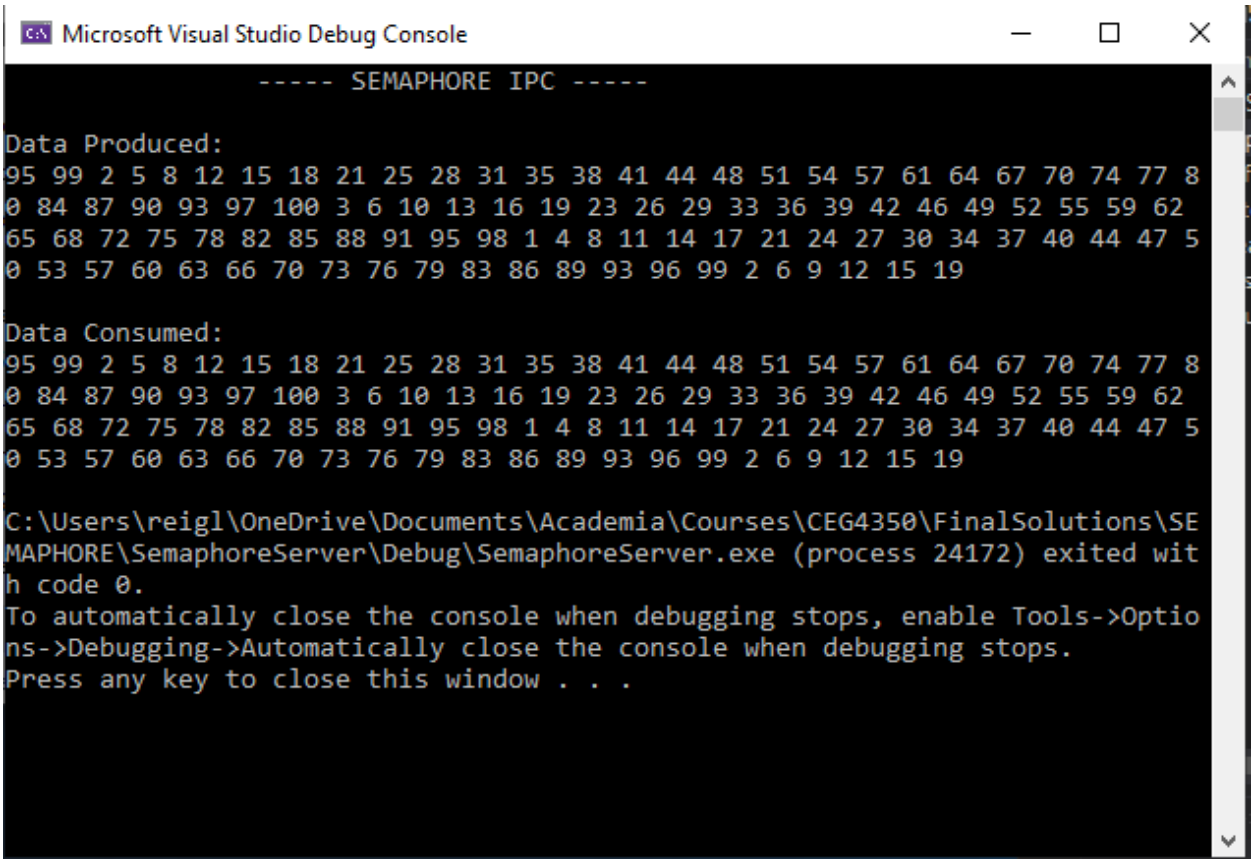

Figure 4: Results of IPC Method 4: Binary Semaphore with Circular Buffer Method

(6) General Discussion:

I have approached this project with an effort to gain experience with all or nearly all the methods that can be implemented both in this project and in my job as an engineer frequently working in computer simulation. To this point, I have implemented both multiple processes and threads. I have also implemented parent processes with multiple child processes to better represent applications that would be used in industry. Additionally, I have implemented both the socket method and semaphore method with simulation execution in mind. UDP sockets would perform better in simulation (than TCP sockets) due to the quicker availability of output. Multiply threaded semaphores would perform well in executing a single thread-function or a set of thread-functions a large number of times, a feature found in most simulations. I also have implemented vectors in the above code in an attempt to improve efficiency and minimize the operations required to execute at task – save the semaphore method due to the definition of the maximum buffer size. However, the semaphore method is most efficient implementation in this report, due to the threading. If the producer-consumer problem defined did not depend on a function limited by small changed in time (such as with the `rand()` function), the `Sleep()` function would not have been required. This project, while challenging, will prove greatly useful.

Appendix:

Output written to .txt files:

Pipe Output		Mailbox Output		Socket Output		Semaphore Output	
Producer	Consumer	Producer	Consumer	Producer	Consumer	Producer	Consumer
32	32	61	61	51	51	95	95
27	27	46	46	52	52	99	99
55	55	23	23	93	93	2	2
6	6	100	100	87	87	5	5
64	64	37	37	1	1	8	8
71	71	80	80	50	50	12	12
46	46	75	75	15	15	15	15
81	81	75	75	58	58	18	18
54	54	84	84	9	9	21	21
94	94	31	31	68	68	25	25
83	83	38	38	100	100	28	28
54	54	62	62	15	15	31	31
23	23	27	27	80	80	35	35
9	9	90	90	83	83	38	38
80	80	37	37	20	20	41	41
30	30	27	27	87	87	44	44
68	68	75	75	81	81	48	48
72	72	39	39	41	41	51	51
5	5	46	46	10	10	54	54
90	90	100	100	73	73	57	57
36	36	74	74	66	66	61	61
29	29	56	56	18	18	64	64
59	59	16	16	27	27	67	67
40	40	85	85	75	75	70	70
90	90	54	54	88	88	74	74
57	57	99	99	35	35	77	77
36	36	90	90	37	37	80	80
86	86	44	44	76	76	84	84
64	64	58	58	3	3	87	87
7	7	90	90	48	48	90	90
90	90	65	65	6	6	93	93

56	56	99	99	66	66	97	97
37	37	7	7	49	49	100	100
46	46	32	32	85	85	3	3
34	34	70	70	1	1	6	6
91	91	42	42	15	15	10	10
6	6	24	24	87	87	13	13
34	34	95	95	96	96	16	16
31	31	39	39	6	6	19	19
5	5	98	98	44	44	23	23
3	3	88	88	73	73	26	26
49	49	12	12	18	18	29	29
31	31	60	60	81	81	33	33
18	18	49	49	64	64	36	36
14	14	35	35	39	39	39	39
36	36	52	52	19	19	42	42
56	56	78	78	24	24	46	46
79	79	91	91	94	94	49	49
84	84	52	52	83	83	52	52
11	11	86	86	55	55	55	55
39	39	51	51	8	8	59	59
43	43	4	4	66	66	62	62
6	6	55	55	58	58	65	65
1	1	48	48	18	18	68	68
1	1	73	73	96	96	72	72
16	16	87	87	97	97	75	75
19	19	11	11	33	33	78	78
41	41	47	47	78	78	82	82
11	11	51	51	18	18	85	85
39	39	82	82	95	95	88	88
45	45	77	77	89	89	91	91
52	52	93	93	36	36	95	95
72	72	12	12	61	61	98	98
26	26	15	15	73	73	1	1
29	29	58	58	90	90	4	4
64	64	14	14	65	65	8	8
5	5	91	91	24	24	11	11

5	5	26	26	3	3	14	14
39	39	77	77	35	35	17	17
68	68	98	98	85	85	21	21
60	60	2	2	71	71	24	24
43	43	60	60	11	11	27	27
21	21	52	52	74	74	30	30
49	49	100	100	17	17	34	34
35	35	83	83	99	99	37	37
58	58	27	27	67	67	40	40
10	10	28	28	49	49	44	44
80	80	55	55	12	12	47	47
35	35	4	4	85	85	50	50
77	77	59	59	67	67	53	53
50	50	99	99	42	42	57	57
38	38	44	44	79	79	60	60
92	92	82	82	13	13	63	63
17	17	45	45	35	35	66	66
25	25	79	79	71	71	70	70
20	20	45	45	47	47	73	73
88	88	48	48	33	33	76	76
82	82	51	51	51	51	79	79
85	85	28	28	41	41	83	83
39	39	44	44	63	63	86	86
92	92	5	5	35	35	89	89
28	28	47	47	31	31	93	93
52	52	34	34	78	78	96	96
38	38	31	31	86	86	99	99
54	54	95	95	70	70	2	2
37	37	27	27	18	18	6	6
60	60	26	26	84	84	9	9
85	85	46	46	91	91	12	12
15	15	18	18	22	22	15	15
65	65	55	55	22	22	19	19