CS311 - FA13: Final

Trevor Bramwell

December 11, 2013

1 Overview

This paper compares and contrasts Stream Sockets, Anonymous Pipes, and Multiprocessing, between the Windows and POSIX APIs. For each section I will provide a sample piece of code for each interface, using as many API functions as possible. I will first give an overview of what the example does, provide the example, then explain how the APIs differ within each example. When these interfaces are placed side by side, this should allow the reader to easily see the similarites and differences between them.

2 Sockets

The first API I will be comparing is Sockets. In Windows these are referred to as *WinSock*. WinSock has the same commands for creating and accepting connections as POSIX sockets, with the addition of 'closesocket'. The difference is in Window's use of macros over *file descriptors* (fds).

```c
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>

#define BUF_SIZE 500

int
main(int argc, char *argv[])
{
    struct addrinfo hints;
    struct addrinfo *result, *rp;
    int sfd, s;
    struct sockaddr_storage peer_addr;
    socklen_t peer_addr_len;
    ssize_t nread;
    char buf[BUF_SIZE];

    if (argc != 2) {
        fprintf(stderr, "Usage: %s port\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_UNSPEC;     /* Allow IPv4 or IPv6 */
    hints.ai_socktype = SOCK_DGRAM; /* Datagram socket */
    hints.ai_flags = AI_PASSIVE;     /* For wildcard IP address */
```

```
31      hints.ai_protocol = 0;          /* Any protocol */
32      hints.ai_canonname = NULL;
33      hints.ai_addr = NULL;
34      hints.ai_next = NULL;
35
36      s = getaddrinfo(NULL, argv[1], &hints, &result);
37      if (s != 0) {
38          fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
39          exit(EXIT_FAILURE);
40      }
41
42      /* getaddrinfo() returns a list of address structures.
43         Try each address until we successfully bind(2).
44         If socket(2) (or bind(2)) fails, we (close the socket
45         and) try the next address. */
46
47      for (rp = result; rp != NULL; rp = rp->ai_next) {
48          sfd = socket(rp->ai_family, rp->ai_socktype,
49                  rp->ai_protocol);
50          if (sfd == -1)
51              continue;
52
53          if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
54              break;                  /* Success */
55
56          close(sfd);
57      }
58
59      if (rp == NULL) {               /* No address succeeded */
60          fprintf(stderr, "Could not bind\n");
61          exit(EXIT_FAILURE);
62      }
63
64      freeaddrinfo(result);           /* No longer needed */
65
66      /* Read datagrams and echo them back to sender */
67
68      for (;;) {
69          peer_addr_len = sizeof(struct sockaddr_storage);
70          nread = recvfrom(sfd, buf, BUF_SIZE, 0,
71                  (struct sockaddr *) &peer_addr, &peer_addr_len);
72          if (nread == -1)
73              continue;               /* Ignore failed request */
74
75          char host[NI_MAXHOST], service[NI_MAXSERV];
76
77          s = getnameinfo((struct sockaddr *) &peer_addr,
78                          peer_addr_len, host, NI_MAXHOST,
79                          service, NI_MAXSERV, NI_NUMERICSERV);
80         if (s == 0)
81              printf("Received %ld bytes from %s:%s\n",
82                      (long) nread, host, service);
83          else
84              fprintf(stderr, "getnameinfo: %s\n", gai_strerror(s));
85
86          if (sendto(sfd, buf, nread, 0,
87                      (struct sockaddr *) &peer_addr,
88                      peer_addr_len) != nread)
89              fprintf(stderr, "Error sending response\n");
90      }
91  }
```

posix_sockets_server.c

```
1  #undef UNICODE
2
```

```
3  #define WIN32_LEAN_AND_MEAN
4
5  #include <windows.h>
6  #include <winsock2.h>
7  #include <ws2tcpip.h>
8  #include <stdlib.h>
9  #include <stdio.h>
10
11 // Need to link with Ws2_32.lib
12 #pragma comment (lib, "Ws2_32.lib")
13 // #pragma comment (lib, "Mswsock.lib")
14
15 #define DEFAULT_BUFLEN 512
16 #define DEFAULT_PORT "27015"
17
18 int __cdecl main(void)
19 {
20     WSADATA wsaData;
21     int iResult;
22
23     SOCKET ListenSocket = INVALID_SOCKET;
24     SOCKET ClientSocket = INVALID_SOCKET;
25
26     struct addrinfo *result = NULL;
27     struct addrinfo hints;
28
29     int iSendResult;
30     char recvbuf[DEFAULT_BUFLEN];
31     int recvbuflen = DEFAULT_BUFLEN;
32
33     // Initialize Winsock
34     iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
35     if (iResult != 0) {
36         printf("WSAStartup failed with error: %d\n", iResult);
37         return 1;
38     }
39
40     ZeroMemory(&hints, sizeof(hints));
41     hints.ai_family = AF_INET;
42     hints.ai_socktype = SOCK_STREAM;
43     hints.ai_protocol = IPPROTO_TCP;
44     hints.ai_flags = AI_PASSIVE;
45
46     // Resolve the server address and port
47     iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result);
48     if ( iResult != 0 ) {
49         printf("getaddrinfo failed with error: %d\n", iResult);
50         WSACleanup();
51         return 1;
52     }
53
54     // Create a SOCKET for connecting to server
55     ListenSocket = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
56     if (ListenSocket == INVALID_SOCKET) {
57         printf("socket failed with error: %ld\n", WSAGetLastError());
58         freeaddrinfo(result);
59         WSACleanup();
60         return 1;
61     }
62
63     // Setup the TCP listening socket
64     iResult = bind( ListenSocket, result->ai_addr, (int)result->ai_addrlen);
65     if (iResult == SOCKET_ERROR) {
66         printf("bind failed with error: %d\n", WSAGetLastError());
67         freeaddrinfo(result);
```

```
68          closesocket(ListenSocket);
69          WSACleanup();
70          return 1;
71      }
72
73      freeaddrinfo(result);
74
75      iResult = listen(ListenSocket, SOMAXCONN);
76      if (iResult == SOCKET_ERROR) {
77          printf("listen failed with error: %d\n", WSAGetLastError());
78          closesocket(ListenSocket);
79          WSACleanup();
80          return 1;
81      }
82
83      // Accept a client socket
84      ClientSocket = accept(ListenSocket, NULL, NULL);
85      if (ClientSocket == INVALID_SOCKET) {
86          printf("accept failed with error: %d\n", WSAGetLastError());
87          closesocket(ListenSocket);
88          WSACleanup();
89          return 1;
90      }
91
92      // No longer need server socket
93      closesocket(ListenSocket);
94
95      // Receive until the peer shuts down the connection
96      do {
97
98          iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
99          if (iResult > 0) {
100             printf("Bytes received: %d\n", iResult);
101
102         // Echo the buffer back to the sender
103             iSendResult = send( ClientSocket, recvbuf, iResult, 0 );
104             if (iSendResult == SOCKET_ERROR) {
105                 printf("send failed with error: %d\n", WSAGetLastError());
106                 closesocket(ClientSocket);
107                 WSACleanup();
108                 return 1;
109             }
110             printf("Bytes sent: %d\n", iSendResult);
111         }
112         else if (iResult == 0)
113             printf("Connection closing...\n");
114         else  {
115             printf("recv failed with error: %d\n", WSAGetLastError());
116             closesocket(ClientSocket);
117             WSACleanup();
118             return 1;
119         }
120
121     } while (iResult > 0);
122
123     // shutdown the connection since we're done
124     iResult = shutdown(ClientSocket, SD_SEND);
125     if (iResult == SOCKET_ERROR) {
126         printf("shutdown failed with error: %d\n", WSAGetLastError());
127         closesocket(ClientSocket);
128         WSACleanup();
129         return 1;
130     }
131
132     // cleanup
```

```
133     closesocket(ClientSocket);
134     WSACleanup();
135
136     return 0;
137 }
```

win32_sockets_server.c

Client Socket Example: `http://msdn.microsoft.com/en-us/library/windows/desktop/ms737591(v=vs.85).aspx`

3 Anonymous Pipes

```
1  #include <sys/types.h>
2  #include <sys/wait.h>
3  #include <sys/types.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #include <string.h>
8
9  int
10 main(int argc, char *argv[])
11 {
12     int pipefd[2];
13     pid_t cpid;
14     char buf;
15
16     if (argc != 2) {
17      fprintf(stderr, "Usage: %s <string>\n", argv[0]);
18      exit(EXIT_FAILURE);
19     }
20
21     if (pipe(pipefd) == -1) {
22         perror("pipe");
23         exit(EXIT_FAILURE);
24     }
25
26     cpid = fork();
27     if (cpid == -1) {
28         perror("fork");
29         exit(EXIT_FAILURE);
30     }
31
32     if (cpid == 0) {    /* Child reads from pipe */
33         close(pipefd[1]);          /* Close unused write end */
34
35         while (read(pipefd[0], &buf, 1) > 0)
36             write(STDOUT_FILENO, &buf, 1);
37
38         write(STDOUT_FILENO, "\n", 1);
39         close(pipefd[0]);
40         _exit(EXIT_SUCCESS);
41
42     } else {            /* Parent writes argv[1] to pipe */
43         close(pipefd[0]);          /* Close unused read end */
44         write(pipefd[1], argv[1], strlen(argv[1]));
45         close(pipefd[1]);          /* Reader will see EOF */
46         wait(NULL);                /* Wait for child */
47         exit(EXIT_SUCCESS);
48     }
49 }
```

posix_pipes.c

```c
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include <strsafe.h>

#define BUFSIZE 4096

HANDLE g_hChildStd_IN_Rd = NULL;
HANDLE g_hChildStd_IN_Wr = NULL;
HANDLE g_hChildStd_OUT_Rd = NULL;
HANDLE g_hChildStd_OUT_Wr = NULL;

HANDLE g_hInputFile = NULL;

void CreateChildProcess(void);
void WriteToPipe(void);
void ReadFromPipe(void);
void ErrorExit(PTSTR);

int _tmain(int argc, TCHAR *argv[])
{
   SECURITY_ATTRIBUTES saAttr;

   printf("\n->Start of parent execution.\n");

// Set the bInheritHandle flag so pipe handles are inherited.

   saAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
   saAttr.bInheritHandle = TRUE;
   saAttr.lpSecurityDescriptor = NULL;

// Create a pipe for the child process's STDOUT.

   if ( ! CreatePipe(&g_hChildStd_OUT_Rd, &g_hChildStd_OUT_Wr, &saAttr, 0) )
      ErrorExit(TEXT("StdoutRd CreatePipe"));

// Ensure the read handle to the pipe for STDOUT is not inherited.

   if ( ! SetHandleInformation(g_hChildStd_OUT_Rd, HANDLE_FLAG_INHERIT, 0) )
      ErrorExit(TEXT("Stdout SetHandleInformation"));

// Create a pipe for the child process's STDIN.

   if (! CreatePipe(&g_hChildStd_IN_Rd, &g_hChildStd_IN_Wr, &saAttr, 0))
      ErrorExit(TEXT("Stdin CreatePipe"));

// Ensure the write handle to the pipe for STDIN is not inherited.

   if ( ! SetHandleInformation(g_hChildStd_IN_Wr, HANDLE_FLAG_INHERIT, 0) )
      ErrorExit(TEXT("Stdin SetHandleInformation"));

// Create the child process.

   CreateChildProcess();

// Get a handle to an input file for the parent.
// This example assumes a plain text file and uses string output to verify data flow.

   if (argc == 1)
      ErrorExit(TEXT("Please specify an input file.\n"));

   g_hInputFile = CreateFile(
```

```
63          argv[1],
64          GENERIC_READ,
65          0,
66          NULL,
67          OPEN_EXISTING,
68          FILE_ATTRIBUTE_READONLY,
69          NULL);
70
71      if ( g_hInputFile == INVALID_HANDLE_VALUE )
72          ErrorExit(TEXT("CreateFile"));
73
74  // Write to the pipe that is the standard input for a child process.
75  // Data is written to the pipe's buffers, so it is not necessary to wait
76  // until the child process is running before writing data.
77
78      WriteToPipe();
79      printf( "\n->Contents of %s written to child STDIN pipe.\n", argv[1]);
80
81  // Read from pipe that is the standard output for child process.
82
83      printf( "\n->Contents of child process STDOUT:\n\n", argv[1]);
84      ReadFromPipe();
85
86      printf("\n->End of parent execution.\n");
87
88  // The remaining open handles are cleaned up when this process terminates.
89  // To avoid resource leaks in a larger application, close handles explicitly.
90
91      return 0;
92  }
93
94  void CreateChildProcess()
95  // Create a child process that uses the previously created pipes for STDIN and STDOUT.
96  {
97      TCHAR szCmdline[]=TEXT("child");
98      PROCESS_INFORMATION piProcInfo;
99      STARTUPINFO siStartInfo;
100     BOOL bSuccess = FALSE;
101
102 // Set up members of the PROCESS_INFORMATION structure.
103
104     ZeroMemory( &piProcInfo, sizeof(PROCESS_INFORMATION) );
105
106 // Set up members of the STARTUPINFO structure.
107 // This structure specifies the STDIN and STDOUT handles for redirection.
108
109     ZeroMemory( &siStartInfo, sizeof(STARTUPINFO) );
110     siStartInfo.cb = sizeof(STARTUPINFO);
111     siStartInfo.hStdError = g_hChildStd_OUT_Wr;
112     siStartInfo.hStdOutput = g_hChildStd_OUT_Wr;
113     siStartInfo.hStdInput = g_hChildStd_IN_Rd;
114     siStartInfo.dwFlags |= STARTF_USESTDHANDLES;
115
116 // Create the child process.
117
118     bSuccess = CreateProcess(NULL,
119         szCmdline,     // command line
120         NULL,          // process security attributes
121         NULL,          // primary thread security attributes
122         TRUE,          // handles are inherited
123         0,             // creation flags
124         NULL,          // use parent's environment
125         NULL,          // use parent's current directory
126         &siStartInfo,  // STARTUPINFO pointer
127         &piProcInfo);  // receives PROCESS_INFORMATION
```

```
128
129     // If an error occurs, exit the application.
130     if ( ! bSuccess )
131        ErrorExit(TEXT("CreateProcess"));
132     else
133     {
134        // Close handles to the child process and its primary thread.
135        // Some applications might keep these handles to monitor the status
136        // of the child process, for example.
137
138        CloseHandle(piProcInfo.hProcess);
139        CloseHandle(piProcInfo.hThread);
140     }
141 }
142
143 void WriteToPipe(void)
144
145 // Read from a file and write its contents to the pipe for the child's STDIN.
146 // Stop when there is no more data.
147 {
148    DWORD dwRead, dwWritten;
149    CHAR chBuf[BUFSIZE];
150    BOOL bSuccess = FALSE;
151
152     for (;;)
153     {
154        bSuccess = ReadFile(g_hInputFile, chBuf, BUFSIZE, &dwRead, NULL);
155        if ( ! bSuccess || dwRead == 0 ) break;
156
157        bSuccess = WriteFile(g_hChildStd_IN_Wr, chBuf, dwRead, &dwWritten, NULL);
158        if ( ! bSuccess ) break;
159     }
160
161 // Close the pipe handle so the child process stops reading.
162
163     if ( ! CloseHandle(g_hChildStd_IN_Wr) )
164        ErrorExit(TEXT("StdInWr CloseHandle"));
165 }
166
167 void ReadFromPipe(void)
168
169 // Read output from the child process's pipe for STDOUT
170 // and write to the parent process's pipe for STDOUT.
171 // Stop when there is no more data.
172 {
173    DWORD dwRead, dwWritten;
174    CHAR chBuf[BUFSIZE];
175    BOOL bSuccess = FALSE;
176    HANDLE hParentStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
177
178     for (;;)
179     {
180        bSuccess = ReadFile( g_hChildStd_OUT_Rd, chBuf, BUFSIZE, &dwRead, NULL);
181        if( ! bSuccess || dwRead == 0 ) break;
182
183        bSuccess = WriteFile(hParentStdOut, chBuf,
184                             dwRead, &dwWritten, NULL);
185        if (! bSuccess ) break;
186     }
187 }
188
189 void ErrorExit(PTSTR lpszFunction)
190
191 // Format a readable error message, display a message box,
192 // and exit from the application.
```

```
{
    LPVOID lpMsgBuf;
    LPVOID lpDisplayBuf;
    DWORD dw = GetLastError();

    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        dw,
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
        (LPTSTR) &lpMsgBuf,
        0, NULL );

    lpDisplayBuf = (LPVOID)LocalAlloc(LMEM_ZEROINIT,
        (lstrlen((LPCTSTR)lpMsgBuf)+lstrlen((LPCTSTR)lpszFunction)+40)*sizeof(TCHAR));
    StringCchPrintf((LPTSTR)lpDisplayBuf,
        LocalSize(lpDisplayBuf) / sizeof(TCHAR),
        TEXT("%s failed with error %d: %s"),
        lpszFunction, dw, lpMsgBuf);
    MessageBox(NULL, (LPCTSTR)lpDisplayBuf, TEXT("Error"), MB_OK);

    LocalFree(lpMsgBuf);
    LocalFree(lpDisplayBuf);
    ExitProcess(1);
}
```

win32_pipes.c

Pipes Example: `http://msdn.microsoft.com/en-us/library/windows/desktop/ms682499(v=vs.85).aspx`

4 Multiprocessing

```
#include <sys/wait.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

int
main(int argc, char *argv[])
{
    pid_t cpid, w;
    int status;

    cpid = fork();
    if (cpid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (cpid == 0) {            /* Code executed by child */
        printf("Child PID is %ld\n", (long) getpid());
        if (argc == 1)
            pause();                    /* Wait for signals */
        _exit(atoi(argv[1]));

    } else {                    /* Code executed by parent */
        do {
            w = waitpid(cpid, &status, WUNTRACED | WCONTINUED);
            if (w == -1) {
```

```
                perror("waitpid");
                exit(EXIT_FAILURE);
            }

            if (WIFEXITED(status)) {
                printf("exited, status=%d\n", WEXITSTATUS(status));
            } else if (WIFSIGNALED(status)) {
                printf("killed by signal %d\n", WTERMSIG(status));
            } else if (WIFSTOPPED(status)) {
                printf("stopped by signal %d\n", WSTOPSIG(status));
            } else if (WIFCONTINUED(status)) {
                printf("continued\n");
            }
        } while (!WIFEXITED(status) && !WIFSIGNALED(status));
        exit(EXIT_SUCCESS);
    }
}
```

posix_procs.c

```
#include <windows.h>
#include <stdio.h>
#include <tchar.h>

void _tmain( int argc, TCHAR *argv[] )
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    ZeroMemory( &si, sizeof(si) );
    si.cb = sizeof(si);
    ZeroMemory( &pi, sizeof(pi) );

    if( argc != 2 )
    {
        printf("Usage: %s [cmdline]\n", argv[0]);
        return;
    }

    // Start the child process.
    if( !CreateProcess( NULL,   // No module name (use command line)
        argv[1],        // Command line
        NULL,           // Process handle not inheritable
        NULL,           // Thread handle not inheritable
        FALSE,          // Set handle inheritance to FALSE
        0,              // No creation flags
        NULL,           // Use parent's environment block
        NULL,           // Use parent's starting directory
        &si,            // Pointer to STARTUPINFO structure
        &pi )           // Pointer to PROCESS_INFORMATION structure
    )
    {
        printf( "CreateProcess failed (%d).\n", GetLastError() );
        return;
    }

    // Wait until child process exits.
    WaitForSingleObject( pi.hProcess, INFINITE );

    // Close process and thread handles.
    CloseHandle( pi.hProcess );
    CloseHandle( pi.hThread );
}
```

win32_procs.c

Multiprocessing Example: `https://msdn.microsoft.com/en-us/library/windows/desktop/ms682512(v=vs.85).aspx`

5 References

5.1 Sockets

http://msdn.microsoft.com/en-us/library/windows/desktop/bb530741(v=vs.85).aspx

Windows API Reference: msdn.microsoft.com/en-us/library/windows/desktop/ms741394(v=vs.85).aspx POSIX References: http://pubs.opengroup.org/onlinepubs/9699919799/toc.htm

5.2 Pipes

http://msdn.microsoft.com/en-us/library/windows/desktop/aa365780(v=vs.85).aspx

http://msdn.microsoft.com/en-us/library/windows/desktop/aa365590(v=vs.85).aspx

5.3 Multiprocessing

http://msdn.microsoft.com/en-us/library/windows/desktop/ms684841(v=vs.85).aspx