جامعة المنارة

جامعة المنارة

كلية: الهندسة

قسم: المعلوماتية

اسم المقرر: نظم تشغيل ٢

رقم الجلسة (٧)

عنوان الجلسة

نظم التشغيل الموزعة

(برمجة المقابس)

الفصل الدراسي

الأول

العام الدراسي

٢٠٢٣_٢٠٢٤

جدول المحتويات

Contents

الغاية من الجلسة:

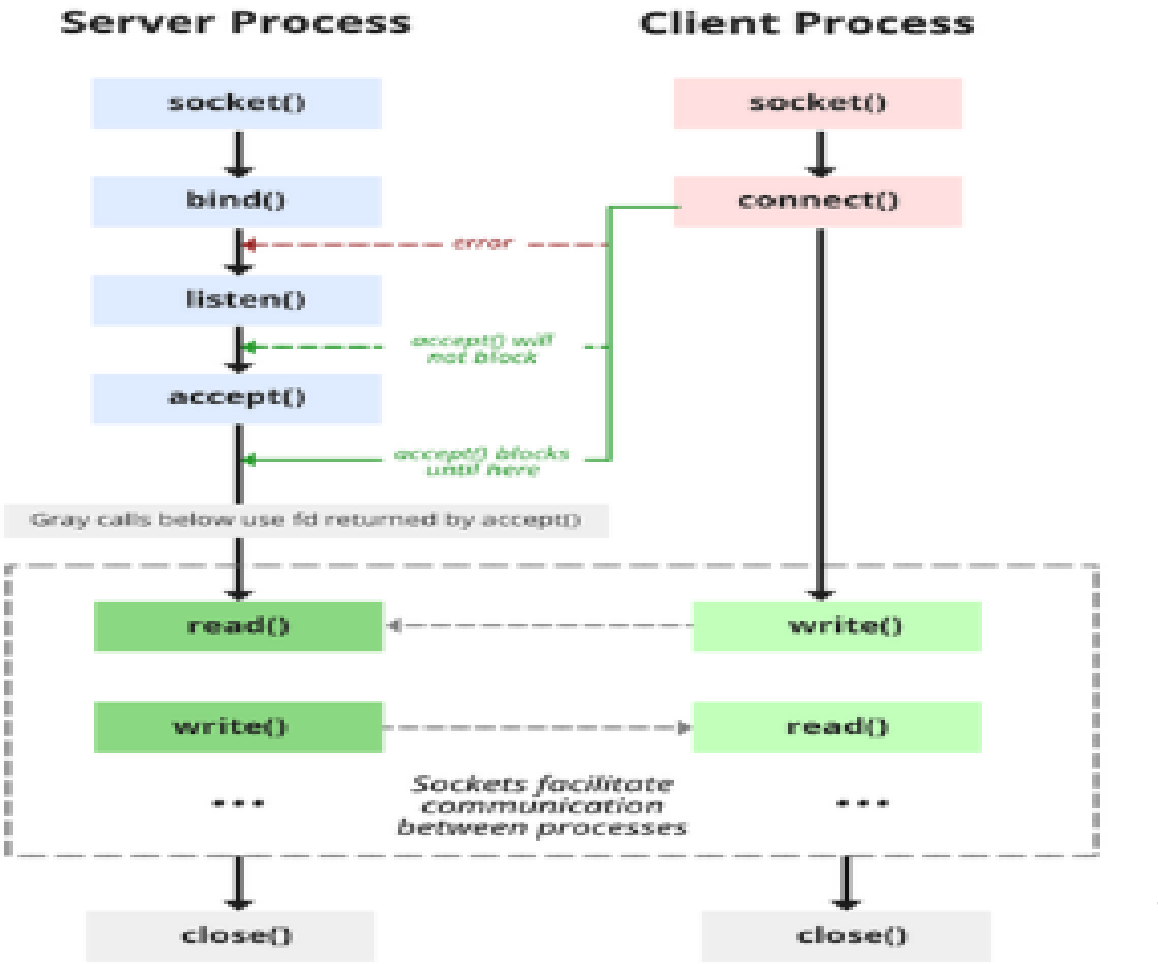
تعريف الطلاب ب

- ✓ الربط بين العقد بواسطة المقابس
- ✓ انشاء مقبس من جهة المخدم يعمل كمستمع للطلبات القادمة من الزبائن
- ✓ انشاء مقبس من جهة طرقية يعمل كزبون يرسل الطلبات إلى المقيس من جهة الخادم

برمجة المقبس:

هي وسيلة لربط عقدتين على الشبكة للتواصل مع بعضهما البعض. يستمع أحد طرفي المقبس (العقدة) إلى منفذ معين عند عنوان IP، بينما يتصل طرف المقبس الآخر لتكوين الاتصال.

يشكل الخادم مقبس المستمع بينما يتصل العميل بالخادم.



على البيانات المستلمة.

يوفر قالبًا أساسيًا لتنفيذ خادم TCP بلغة C++ على نظام Windows الأساسي باستخدام مكتبة .Winsock و يمثل نقطة انطلاق لبناء تطبيقات شبكية تتضمن الاتصال بين الخادم وعملاء متعددين.

من جهة الخادم

```
#undef UNICODE
#define WIN32_LEAN_AND_MEAN
```


```
#include <windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdlib.h>
#include <stdio.h>
// Need to link with Ws2_32.lib
#pragma comment (lib, "Ws2_32.lib")
// #pragma comment (lib, "Mswsock.lib")

#define DEFAULT_BUFLEN 512
#define DEFAULT_PORT "27015"
int __cdecl main(void)
{
    WSADATA wsaData;
    int iResult;
    SOCKET ListenSocket = INVALID_SOCKET;
    SOCKET ClientSocket = INVALID_SOCKET;
    struct addrinfo *result = NULL;
    struct addrinfo hints;
    int iSendResult;
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;
    // Initialize Winsock
    iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup failed with error: %d\n", iResult);
        return 1;
    }

    ZeroMemory(&hints, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_protocol = IPPROTO_TCP;
    hints.ai_flags = AI_PASSIVE;

    // Resolve the server address and port
    iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result);
    if ( iResult != 0 ) {
        printf("getaddrinfo failed with error: %d\n", iResult);
        WSACleanup();
        return 1;
    }
    // Create a SOCKET for the server to listen for client connections.
    ListenSocket = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
    if (ListenSocket == INVALID_SOCKET) {
        printf("socket failed with error: %ld\n", WSAGetLastError());
        freeaddrinfo(result);
        WSACleanup();
        return 1;
```

```
}
// Setup the TCP listening socket
iResult = bind( ListenSocket, result->ai_addr, (int)result->ai_addrlen);
if (iResult == SOCKET_ERROR) {
    printf("bind failed with error: %d\n", WSAGetLastError());
    freeaddrinfo(result);
    closesocket(ListenSocket);
    WSACleanup();
    return 1;
}
freeaddrinfo(result);
iResult = listen(ListenSocket, SOMAXCONN);
if (iResult == SOCKET_ERROR) {
    printf("listen failed with error: %d\n", WSAGetLastError());
    closesocket(ListenSocket);
    WSACleanup();
    return 1;
}
// Accept a client socket
ClientSocket = accept(ListenSocket, NULL, NULL);
if (ClientSocket == INVALID_SOCKET) {
    printf("accept failed with error: %d\n", WSAGetLastError());
    closesocket(ListenSocket);
    WSACleanup();
    return 1;
}
// No longer need server socket
closesocket(ListenSocket);
// Receive until the peer shuts down the connection
do {

    iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
    if (iResult > 0) {
        printf("Bytes received: %d\n", iResult);

    // Echo the buffer back to the sender
        iSendResult = send( ClientSocket, recvbuf, iResult, 0 );
        if (iSendResult == SOCKET_ERROR) {
            printf("send failed with error: %d\n", WSAGetLastError());
            closesocket(ClientSocket);
            WSACleanup();
            return 1;
        }
        printf("Bytes sent: %d\n", iSendResult);
    }
    else if (iResult == 0)
        printf("Connection closing...\n");
    else  {
        printf("recv failed with error: %d\n", WSAGetLastError());
```

```
            closesocket(ClientSocket);
            WSACleanup();
            return 1;
        }
    } while (iResult > 0);

    // shutdown the connection since we're done
    iResult = shutdown(ClientSocket, SD_SEND);
    if (iResult == SOCKET_ERROR) {
        printf("shutdown failed with error: %d\n", WSAGetLastError());
        closesocket(ClientSocket);
        WSACleanup();
        return 1;
    }

    // cleanup
    closesocket(ClientSocket);
    WSACleanup();

    return 0;
}
```

من جهة الزبون

```
#define WIN32_LEAN_AND_MEAN

#include <windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdlib.h>
#include <stdio.h>
// Need to link with Ws2_32.lib, Mswsock.lib, and Advapi32.lib
#pragma comment (lib, "Ws2_32.lib")
#pragma comment (lib, "Mswsock.lib")
#pragma comment (lib, "AdvApi32.lib")

#define DEFAULT_BUFLEN 512
#define DEFAULT_PORT "27015"

int __cdecl main(int argc, char **argv)
{
WSADATA wsaData;
SOCKET ConnectSocket = INVALID_SOCKET;
struct addrinfo *result = NULL,
*ptr = NULL,
hints;
const char *sendbuf = "this is a test";
char recvbuf[DEFAULT_BUFLEN];
int iResult;
int recvbuflen = DEFAULT_BUFLEN;
```


```
// Validate the parameters
if (argc != 2) {
printf("usage: %s server-name\n", argv[0]);
return 1;
}

// Initialize Winsock
iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
if (iResult != 0) {
printf("WSAStartup failed with error: %d\n", iResult);
return 1;
}

ZeroMemory( &hints, sizeof(hints) );
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;

// Resolve the server address and port
iResult = getaddrinfo(argv[1], DEFAULT_PORT, &hints, &result);
if ( iResult != 0 ) {
printf("getaddrinfo failed with error: %d\n", iResult);
WSACleanup();
return 1;
}

// Attempt to connect to an address until one succeeds
for(ptr=result; ptr != NULL ;ptr=ptr->ai_next) {

// Create a SOCKET for connecting to server
ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype,
ptr->ai_protocol);
if (ConnectSocket == INVALID_SOCKET) {
printf("socket failed with error: %ld\n", WSAGetLastError());
WSACleanup();
return 1;
}

// Connect to server.
iResult = connect( ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);
if (iResult == SOCKET_ERROR) {
closesocket(ConnectSocket);
ConnectSocket = INVALID_SOCKET;
continue;
}
break;
}
```

```
freeaddrinfo(result);

if (ConnectSocket == INVALID_SOCKET) {
printf("Unable to connect to server!\n");
WSACleanup();
return 1;
}

// Send an initial buffer
iResult = send( ConnectSocket, sendbuf, (int)strlen(sendbuf), 0 );
if (iResult == SOCKET_ERROR) {
printf("send failed with error: %d\n", WSAGetLastError());
closesocket(ConnectSocket);
WSACleanup();
return 1;
}

printf("Bytes Sent: %ld\n", iResult);

// shutdown the connection since no more data will be sent
iResult = shutdown(ConnectSocket, SD_SEND);
if (iResult == SOCKET_ERROR) {
printf("shutdown failed with error: %d\n", WSAGetLastError());
closesocket(ConnectSocket);
WSACleanup();
return 1;
}

// Receive until the peer closes the connection
do {

iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
if ( iResult > 0 )
printf("Bytes received: %d\n", iResult);
else if ( iResult == 0 )
printf("Connection closed\n");
else
printf("recv failed with error: %d\n", WSAGetLastError());

} while( iResult > 0 );

// cleanup
closesocket(ConnectSocket);
WSACleanup();

return 0;
}
```

شرح مكونات البرنامج

- **تعريفات :**

```
#define DEFAULT_BUFLEN 512
#define DEFAULT_PORT "27015"
```

تحدد هذه الثوابت طول المخزن المؤقت الافتراضي ورقم المنفذ

- **الوظيفة الأساسية**:

```
int __cdecl main(int argc, char **argv)
```

الوظيفة الرئيسية تأخذ وسائط سطر الأوامر. يقوم بتهيئة Winsock، وإعداد معلومات العنوان، والاتصال بالخادم، وإرسال البيانات، واستقبال البيانات، وإغلاق الاتصال.

- **تهيئة Winsock:**

```
WSADATA wsaData;
int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
```

يؤدي هذا إلى تهيئة مكتبة Winsock. يتم استخدام بنية WSADATA لتخزين تفاصيل حول تطبيق Windows Switches.

- **إعداد معلومات العنوان:**

```
struct addrinfo *result = NULL, *ptr = NULL, hints;
```

يتم استخدام هذه البنية لتخزين معلومات العنوان للخادم. توفر بنية التلميحات تلميحات حول نوع المقبس الذي يدعمه المتصل.

التحقق من صحة وسيطة سطر الأوامر:

```
if (argc != 2) {
    printf("usage: %s server-name\n", argv[0]);
    return 1;
}
```

يضمن تنفيذ البرنامج بالعدد الصحيح من وسائط سطر الأوامر.

- **حل عنوان الخادم:**

```
iResult = getaddrinfo(argv[1], DEFAULT_PORT, &hints, &result);
```

يتنقل عبر قائمة العناوين التي تم إرجاعها بواسطة getaddrinfo ويحاول الاتصال بكل منها حتى ينجح.

- **إرسال البيانات:**

```
iResult = send(ConnectSocket, sendbuf, (int)strlen(sendbuf), 0);
```

يرسل البيانات إلى الخادم باستخدام المقبس المتصل.

- **اغلاق الاتصال:**

```
iResult = shutdown(ConnectSocket, SD_SEND);
```

يقوم بإيقاف تشغيل الجانب المرسل من الاتصال.

جار استقبال البيانات:

```
do {
   iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
   // ...
} while (iResult > 0);
```

تلقى البيانات من الخادم في حلقة حتى يقوم الخادم بإغلاق الاتصال.

- **التنظيف:**

```
closesocket(ConnectSocket);
WSACleanup();
```

التنفيذ :

يتم تشغيل المقبس من جهة المخدم و يصبح جاهز لتلقي الطلبات بعد انشاء ملف التنفيذ و تشغيله

```
Path_to_exec_file> file_name.exe.
```

من جهة الزبون يتم تشغيل المقبس و ارسال الطلبات إلى المخدم بعد كتابة اسم المخدم أو عنوانه IP

```
Path_to_exec_file> file_name.exe server_IP
```