BÀI 3.
CÁC CHẾ ĐỘ VÀO RA TRÊN WINSOCK(TIẾP)

Nội dung

- Chế độ vào ra blocking và non-blocking
- Kỹ thuật đa luồng
- Kỹ thuật thăm dò
- Kỹ thuật vào ra theo thông báo
- Kỹ thuật vào ra theo sự kiện
- Kỹ thuật Overlapped
- Kỹ thuật vào ra trên Completion Port

5. KỸ THUẬT VÀO RA THEO SỰ KIỆN

Kỹ thuật vào ra theo sự kiện

- Vào ra bất đồng bộ tương tự WSAAsyncSelect
- Hàm *WSAEventSelect()* được sử dụng để gắn một bộ bắt sự kiện WSAEVENT với mỗi socket
- Khi sự kiện xảy ra, đối tượng *WSAEVENT* chuyển từ trạng thái chưa báo hiệu(non-signaled) sang đã báo hiệu(signaled)
- Tạo đối tượng *WSAEVENT*

```
WSAEVENT WSACreateEvent(void);
```

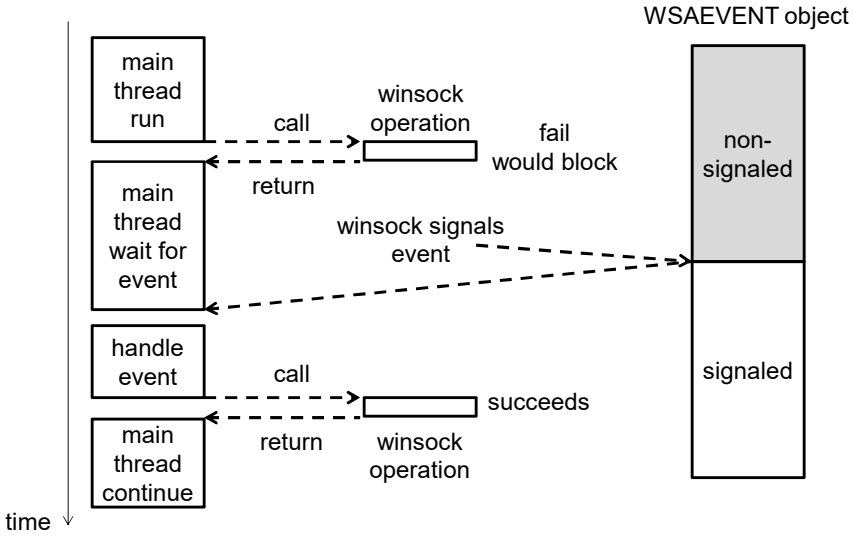
- Sau khi xử lý sự kiện, cần chuyển đối tượng *WSAEVENT* trở lại trạng thái chưa báo hiệu:

```
BOOL WSAResetEvent(WSAEVENT hEvent);
```

- Hủy đối tượng sự kiện

```
BOOL WSACloseEvent(WSAEVENT hEvent);
```

Kỹ thuật vào ra theo sự kiện



5

Hàm *WSAEventSelect()*

- Gắn bộ bắt sự kiện vào socket: *WSAEventSelect()*
- Chuyển socket sang chế độ vào ra không chặn dừng(non-blocking)
- Trả về:
  - Thành công: 0
  - Lỗi: SOCKET_ERROR

```
int WSAEventSelect(
    SOCKET s,           // [IN] Socket được theo dõi sự kiện
    WSAEVENT hEventObject,   // [IN] Bộ bắt sự kiện
                             // WSAEVENT gắn với socket
    long lEvent         // [IN] Mặt nạ xác định các sự
                        // kiện cần theo dõi
);
```

6

Hàm *WSAWaitForMultipleEvents()*

- Đợi các sự kiện xảy ra trên các đối tượng *WSAEVENT*
- Trả về khi có một bộ bắt sự kiện bất kỳ chuyển sang trạng thái báo hiệu hoặc có time-out, hoặc thủ tục xử lý vào ra thực thi

```
DWORD WSAWaitForMultipleEvents(
   DWORD cEvents,     // [IN] Số lượng bộ bắt sự kiện cần đợi
   const WSAEVENT FAR * lphEvents,// [IN] Các bộ bắt sự kiện
   BOOL fWaitAll,     //[IN] Đợi tất cả các bộ bắt sự kiện?
   DWORD dwTimeout,   //[IN] Thời gian chờ tối đa (ms)
   BOOL fAlertable    //[IN]  Thiết lập là FALSE
```

- Số bộ bắt sự kiện tối đa: *WSA_MAXIMUM_WAIT_EVENTS*(=64)
- Giá trị trả về:
  - Thất bại: *WSA_WAIT_FAILED*
  - Time-out: *WSA_WAIT_TIMEOUT*
  - Thành công: Chỉ số bộ bắt sự kiện nhỏ nhất đã chuyển trạng thái + WSA_WAIT_EVENT_0

7

Hàm *WSAEnumNetworkEvents()*

- Xác định các sự kiện xảy ra trên socket

```
int WSAEnumNetworkEvents(
   SOCKET s,               //[IN] Socket muốn thăm dò
   WSAEVENT hEventObject,  //[IN] Bộ bắt sự kiện gắn với
                           // socket
   LPWSANETWORKEVENTS lpNetworkEvents //[OUT] Cấu trúc chứa
                                      //mã sự kiện
);
```

- Cấu trúc *WSANETWORKEVENTS*

```
typedef struct _WSANETWORKEVENTS{
    long lNetworkEvents; //Mặt nạ xác định sự kiện xảy ra
    int  iErrorCode[FD_MAX_EVENTS]; //Mảng các mã lỗi
} WSANETWORKEVENTS, FAR * LPWSANETWORKEVENTS;
```

  - Chỉ số kiểm tra mã lỗi có dạng FD_XXX_BIT
  - Nếu iErrorCode[FD_XXX_BIT] != 0 có lỗi xảy ra với sự kiện FD_XXX

8

Sử dụng kỹ thuật vào ra theo sự kiện

- `socks[]`: Mảng chứa giá trị các socket
- `events[]`: Mảng các bộ nghe sự kiện gắn với socket
  - Bộ nghe `events[i]` gắn với socket `socks[i]`

9

Sử dụng *WSAEventSelect()*

```
DWORD nEvents = 0;
DWORD i, index;
SOCKET socks[WSA_MAXIMUM_WAIT_EVENTS];
WSAEVENT events [WSA_MAXIMUM_WAIT_EVENTS], newEvent;
WSANETWORKEVENTS sockEvent;
//Construct listenning socket
SOCKET listenSock;
listenSock = socket(...);

//create new events
newEvent = WSACreateEvent();

// Associate event types FD_ACCEPT and FD_CLOSE
// with the listening socket and newEvent
WSAEventSelect(listenSock, NewEvent, FD_ACCEPT | FD_CLOSE);
socks[0] = listenSock;
events[0] = newEvent;
nEvents ++;

// Call bind(), listen()
```

10

Sử dụng *WSAEventSelect()*

```
while(1){
   //wait for network events on all socket
   index = WSAWaitForMultipleEvents(nEvents, events, FALSE,
                                                    WSA_INFINITE, FALSE);
   if(index == WSA_WAIT_FAILED) break;
   index = index - WSA_WAIT_EVENT_0;
   // Iterate through all events and enumerate
   // if the wait does not fail.
   for(i = index; i < nEvents, i++){
       index = WSAWaitForMultipleEvents(1, &events[i],
                                                    FALSE, 500, FALSE);
       if(index!= WSA_WAIT_FAILED && index!=WSA_WAIT_TIMEOUT){
          WSAEnumNetworkEvents(socks[i],events[i],
                                                    &sockEvent);
          if(sockEvent.lNetworkEvents & FD_ACCEPT){
              if(sockEvent.iErrorCode[FD_ACCEPT_BIT] != 0){
                 printf("Error!");
                 break;
              }
              //call accept() and process accepted socket
          }
```

11

Sử dụng *WSAEventSelect()*

```
          if(sockEvent.lNetworkEvents & FD_READ){
              //...
          }

          if(sockEvent.lNetworkEvents & FD_WRITE){
              //...
          }

          if(sockEvent.lNetworkEvents & FD_CLOSE){
              //...
          }

          //reset event
          WSAResetEvent(events[i]);

       }
   }//end for
}//end while
```

12

5. KỸ THUẬT VÀO RA OVERLAPPED

13

Kỹ thuật vào ra Overlapped

- Sử dụng cấu trúc WSAOVERLAPPED chứa thông tin về các thao tác vào ra
- Để sử dụng kỹ thuật overlapped, socket phải được khởi tạo với cờ điều khiển tương ứng
- Một số hàm sử dụng kỹ thuật overlapped: WSASend(), WSASendTo(), WSARecv(), WSARecvFrom(), WSAIoctl(), WSARecvMsg(), AcceptEx(), ConnectEx()
- Hiệu năng cao hơn do có thể gửi đồng thời nhiều yêu cầu vào ra tới hệ thống
- Các hàm sử dụng kỹ thuật overlapped sẽ trả về kết quả ngay. Các phương pháp xử lý kết quả:
  - Đợi thông báo từ một sự kiện
  - Thực hiện một thủ tục dạng CALLBACK (completion routine)

14

Hàm WSASocket

- Khởi tạo một socket gắn với provider cung cấp dịch vụ của tầng giao vận(tương tự socket())
- Để socket làm việc ở chế độ Overlapped, gán cờ WSA_FLAG_OVERLAPPED cho tham số dwFlags

```
SOCKET WSASocket(
   int                af,     //Họ giao thức
   int                type,   //Loại socket
   int                protocol, //Giao thức TCP/UDP
   LPWSAPROTOCOL_INFO lpProtocolInfo, //Cấu trúc thông tin
                                      //giao thức
   GROUP              g,       //Nhóm socket, thường là 0
   DWORD              dwFlags  //Cờ khởi tạo
);
```

15

Cấu trúc WSAOVERLAPPED

```
typedef struct WSAOVERLAPPED
{
    DWORD    Internal;
    DWORD    InternalHigh;
    DWORD    Offset;
    DWORD    OffsetHigh;
    WSAEVENT hEvent;
} WSAOVERLAPPED, FAR * LPWSAOVERLAPPED;
```

- Các trường `Internal`, `InternalHigh`, `Offset`, `OffsetHigh` được sử dụng trong nội bộ WinSock
- Trường `hEvent` là một bộ bắt sự kiện chứa các sự kiện khi thao tác vào ra hoàn tất.

16

WSASend()

```
int WSASend(
   SOCKET           s,                  //Socket
   LPWSABUF         lpBuffers,          //Bộ đệm chứa dữ liệu
   DWORD            dwBufferCount,      //Số bộ đệm sử dụng
   LPDWORD          lpNumberOfBytesSent, //Số byte đã gửi
   DWORD            dwFlags,            //Cờ điều khiển
   LPWSAOVERLAPPED lpOverlapped,        //Biến cấu trúc Overlapped
   LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);
```

- Gửi dữ liệu với cơ chế overlapped trên socket
- Gửi dữ liệu trên nhiều bộ đệm
- Trả về:
  - Dữ liệu được gửi đi ngay: 0
  - Thao tác vào ra đang chờ được hoàn thành: WSA_IO_PENDING

17

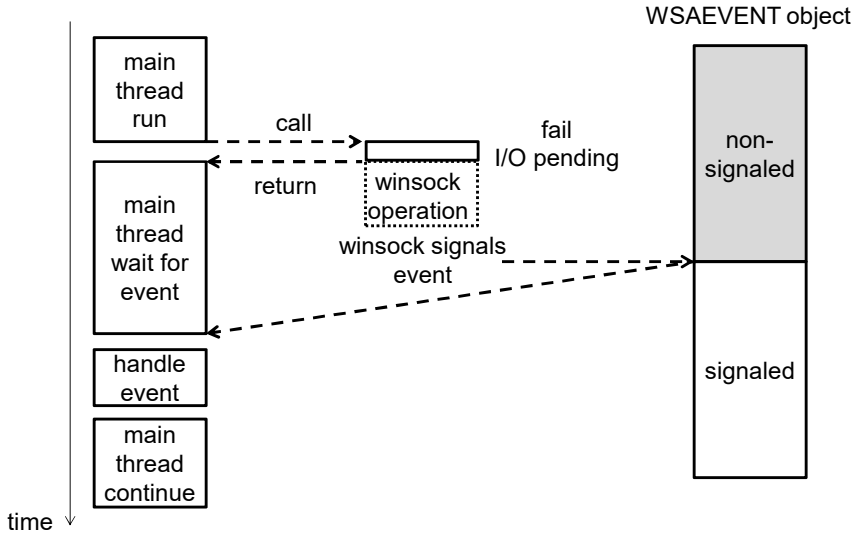
WSARecv()

```
int WSARecv(
   SOCKET           s,                  //Socket
   LPWSABUF         lpBuffers,          //Bộ đệm chứa dữ liệu
   DWORD            dwBufferCount,      //Số bộ đệm sử dụng
   LPDWORD          lpNumberOfBytesSent, //Số byte đã nhận
   LPDWORD          dwFlags,            //Cờ điều khiển
   LPWSAOVERLAPPED lpOverlapped,        //Biến cấu trúc Overlapped
   LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);
```

- Nhận dữ liệu với cơ chế overlapped trên socket
- Nhận dữ liệu trên nhiều bộ đệm
- Trả về:
  - Nhận được dữ liệu ngay: 0
  - Thao tác vào ra đang chờ được hoàn thành: WSA_IO_PENDING

18

Kỹ thuật overlapped – Xử lý qua sự kiện



19

Kỹ thuật overlapped – Xử lý qua sự kiện

- B1: Tạo socket và nghe yêu cầu kết nối
- B2: Goi hàm accept() để tạo socket xử lý 1 kết nối
- B3: Tạo 1 biến cấu trúc WSAOVERLAPPED cho socket kết nối với client ở B2 và gán bộ bắt sự kiện cho cấu trúc
- B4: Gọi hàm vào ra với biến cấu trúc WSAOVERLAPPED
- B5: Gọi hàm **`WSAWaitForMultipleEvents ()`** bắt sự kiện
- B6: Gọi hàm **`WSAGetOverlappedResult()`** xác định kết quả hoàn thành thao tác vào ra
- B7: Gọi hàm **`WSAResetEvent()`** để thiết lập lại bộ nghe sự kiện
- B8: Gọi lại hàm vào ra để trao đổi dữ liệu tiếp theo
- B9: Lặp lại các bước từ 5-8

20

Hàm WSAGetOverlappedResult()

- Lấy kết quả thực hiện thao tác vào ra trên socket
- Trả về:
  - TRUE: Thao tác vào ra hoàn tất thành công
  - FALSE: Thao tác không hoàn tất hoặc có lỗi

```
BOOL WSAGetOverlappedResult(
    SOCKET s,                //Socket cần kiểm tra kết quả
    LPWSAOVERLAPPED lpOverlapped, //cấu trúc WSAOVERLAPPED
    LPDWORD lpcbTransfer,          //Tổng số byte đã trao đổi
    BOOL fWait,              //Chờ thao tác vào ra hoàn tất?
    LPDWORD lpdwFlags        //Cờ báo trạng thái hoàn tất
);
```

21

Kỹ thuật Overlapped – Xử lý qua sự kiện

```
// Simple server application that is capable of managing
// overlapped I/O on one socket using the event notification

SOCKET connSocket, listenSocket;
WSAEVENT events[WSA_MAXIMUM_WAIT_EVENTS];
DWORD flags = 0, nEvents = 0;
WSAOVERLAPPED acceptOverlapped;

// Step 1: Start Winsock, and set up a listening socket

// Step 2: Accept a new connection
connSocket = accept(…);

//Step 3: Set up an overlapped structure
events[nEvents] = WSACreateEvent();
ZeroMemory(&acceptOverlapped, sizeof(WSAOVERLAPPED));
acceptOverlapped.hEvent = events[nEvents];
nEvents++;

//Step 4: Call I/O functions
```

22

Kỹ thuật Overlapped – Xử lý qua sự kiện(tiếp)

```
 // Process overlapped receives on the socket
while(TRUE){
   DWORD    index;

   // Step 5: Wait for the overlapped I/O call to complete
   index = WSAWaitForMultipleEvents(nEvents, events, FALSE,
                                          WSA_INFINITE, FALSE);

   // Step 6: Determine the status of the overlapped request
   WSAGetOverlappedResult(connSocket, &acceptOverlapped,
                     &bytesTransferred, FALSE, &flags);

   // First check to see whether the peer has closed
   // the connection, and if so, close the socket
   if (bytesTransferred == 0){
       //...
       return;
   }

   //do something with the received data...
```

23

Kỹ thuật Overlapped – Xử lý qua sự kiện(tiếp)

```
   // Step 7: Reset the signaled event
   WSAResetEvent(events[index - WSA_WAIT_EVENT_0]);
   ZeroMemory(&acceptOverlapped, sizeof(WSAOVERLAPPED));
   acceptOverlapped.hEvent = events[index - WSA_WAIT_EVENT_0];

   // Step 8: Call I/O function again ...

} // end while
```

Làm cách nào để sử dụng kỹ thuật Overlapped cho nhiều kết nối?

24

Overlapped I/O – Completion routine

- Hệ thống sẽ thông báo cho ứng dụng biết thao tác vào ra kết thúc thông qua một hàm CALLBACK gọi là Completion Routine
- WinSock sẽ bỏ qua trường event trong cấu trúc OVERLAPPED, việc tạo đối tượng event và thăm dò là không cần thiết nữa.
- Lưu ý: chi phí tính toán trên completion routine không được quá cao

```
void CALLBACK CompletionROUTINE(
   DWORD dwError,                //[IN]Trạng thái hoàn thành của
                                 //thao tác vào ra
   DWORD cbTransferred,          //[IN]Số byte trao đổi
   LPWSAOVERLAPPED lpOverlapped,  //[IN] Trỏ tới cấu trúc
                                  // WSAOVERLAPPED tương ứng
   DWORD dwFlags                  // Cờ kết quả thao tác vào ra
);
```

25

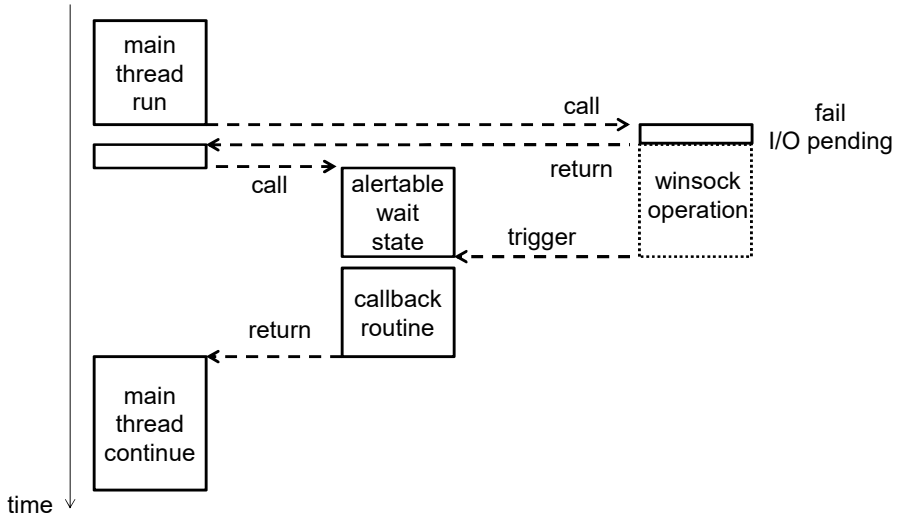
Xử lý completion routine

- Ứng dụng cần chuyển luồng sang trạng thái alertable ngay sau khi gửi yêu cầu vào ra.
- Các hàm có thể chuyển luồng sang trạng thái alertable: WSAWaitForMultipleEvents(), SleepEx()
- Nếu ứng dụng không có đối tượng event nào thì có thể sử dụng SleepEx()

```
DWORD SleepEx(
   DWORD dwMilliseconds,   // Thời gian đợi
   BOOL bAlertable         // Trạng thái alertable
);
```

26

Overlapped I/O – Completion routine



27

Overlapped I/O – Completion routine

- B1: Tạo socket và nghe yêu cầu kết nối
- B2: Goi hàm accept() để tạo socket xử lý 1 kết nối
- B3: Tạo 1 biến cấu trúc WSAOVERLAPPED cho socket kết nối với client ở B2
- B4: Gọi hàm vào ra với biến cấu trúc WSAOVERLAPPED
- B5: Gọi hàm `WSAWaitForMultipleEvents()` với tham số fAlertable là TRUE. Nếu thao tác thực thi hoàn thành, completion routine tự động thực thi, và hàm trả về WSA_IO_COMPLETION. Lưu ý: Trong completion routine gọi hàm vào ra để tiếp tục trao đổi dữ liệu
- B6: Kiểm tra giá trị trả về của `WSAWaitForMultipleEvents()`
- Lặp lại bước 5 và 6

28

Overlapped – Completion routine

```
// Simple server application that is capable of managing one
// socket request using completion routines

SOCKET acceptSocket, listenSocket;
WSAEVENT events[1];
DWORD flags, transferedBytes, index;

void main(void)
{
    WSAOVERLAPPED overlapped;
    // Step 1: Start Winsock, and set up a listening socket
    // Step 2: Accept a new connection

    acceptSocket = accept(...);
    //Step 3: Set up an overlapped structure
    ZeroMemory(&overlapped, sizeof(WSAOVERLAPPED));

    //Step 4: Call I/O functions...
    events[0] = WSACreateEvent();
```

29

Overlapped – Completion routine(tiếp)

```
   while(1){
       // Step 5:
       index = WSAWaitForMultipleEvents(1, events, FALSE,
                    WSA_INFINITE, TRUE);

       // Step 6:
       if (index == WAIT_IO_COMPLETION)
           continue;
       else
           break;
       WSAResetEvent(events[index - WSA_WAIT_EVENT_0]);
    }
}
```

30

Overlapped – Completion routine

```
void CALLBACK completionRoutine(DWORD error,
                                DWORD bytesTransferred,
                                LPWSAOVERLAPPED overlapped,
                                DWORD inFlags)
{
    if (error != 0 || bytesTransferred == 0)  {
        // Either a bad error occurred on the socket or the
        // socket was closed by a peer
        closesocket(acceptSocket);
        return;
    }

    //do something with received data

    ZeroMemory(&overlapped, sizeof(WSAOVERLAPPED));
    // Step 8: Call I/O function again ...
}
```

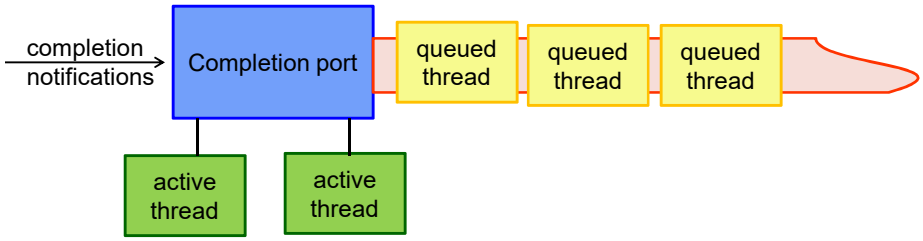
31

6. KỸ THUẬT VÀO RA COMPLETION PORT

32

Kỹ thuật vào ra Completion Port

- Sử dụng completion port để thực hiện các thao tác vào ra overlapped
- Completion port tổ chức một hàng đợi cho các luồng và giám sát các sự kiện vào ra trên các socket
- Mỗi khi thao tác vào ra hoàn thành trên socket, completion port kích hoạt một luồng để xử lý.



33

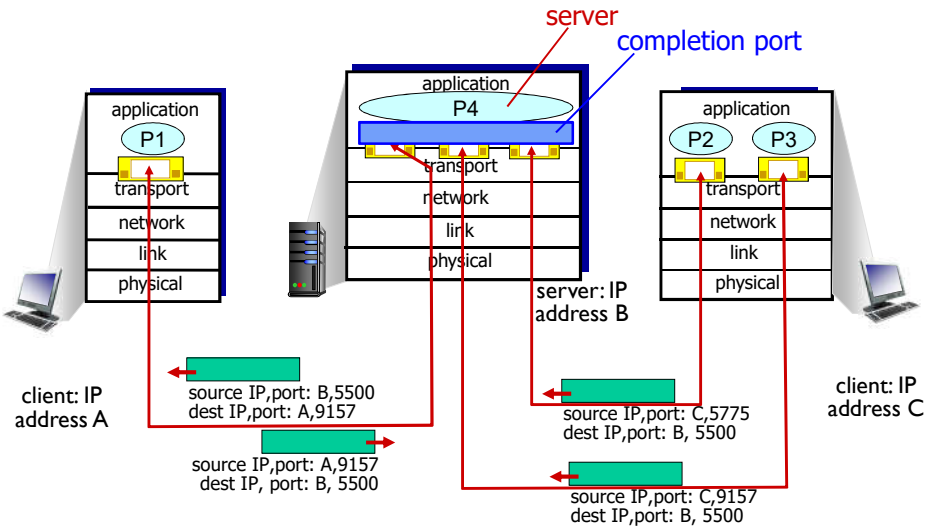
Hàm `CreateIoCompletionPort()`

- Tạo một completion port
- Nếu chưa xác định socket, truyền INVALID_HANDLE_VALUE
- Trả về: NULL nếu có lỗi, ngược lại phụ thuộc vào các giá trị truyền cho tham số
- Nếu completionPort là NULL: Completion port mới
- Nếu completionPort khác NULL: completion port với handle giống giá trị tham số
- Nếu s hợp lệ, socket được liên kết với completion port

```
HANDLE CreateIoCompletionPort (
   HANDLE s,                  //[IN]handle của đối tượng thực
                              // hiện thao tác vào ra
   HANDLE completionPort,     //[IN] Completion port xử lý vào ra
   DWORD completionKey,       //[IN] Định danh cho socket trên
                              //completion port
   DWORD numberOfThread       //[IN] Số luồng sử dụng trên
                              // completion port, thường là 0
);
```

34

Sử dụng completion port



35

Completion Port

- B1: Tạo Completion Port với tham số thứ tư là 0(số luồng thực thi đồng thời bằng số nhân của CPU)
- B2: Xác định số nhân CPU của hệ thống
- B3: Tạo worker thread để xử lý khi thao tác vào ra hoàn thành(sử dụng số nhân CPU đã xác định ở trên)
- B4: Tạo socket và đặt vào trạng thái nghe yêu cầu
- B5: Chấp nhận yêu cầu
- B6: Tạo cấu trúc chứa dữ liệu có ý nghĩa định danh cho socket. Lưu giá trị socket ở bước 5 vào cấu trúc
- B7: Liên kết socket với completion port, truyền cấu trúc ở B6 cho tham số completionKey
- B8: Thực hiện các thao tác vào ra trên socket
- B9: Lặp các thao tác từ 5 đến 8

36

Worker Thread

- Gọi hàm `GetQueuedCompletionStatus()` đợi thao tác vào ra hoàn thành trên completion port và lấy kết quả thực hiện. Trả về FALSE nếu thao tác vào ra lỗi
- Tham số lpCompletionKey và lpOverlapped chứa dữ liệu và kết quả của thao tác vào ra:
  - lpCompletionKey: chứa thông tin có ý nghĩa định danh cho socket trên completion port(per-handle data)
  - lpOverlapped : cấu trúc chứa đối tượng OVERLAPPED và các thông tin để worker thread xử lý dữ liệu(per-I/O data)

```
BOOL GetQueuedCompletionStatus(
   HANDLE CompletionPort,
   LPDWORD lpNumberOfBytesTransferred, //[OUT] Số byte trao đổi
   PULONG_PTR lpCompletionKey,  //[OUT]Định danh cho dữ liệu
   LPOVERLAPPED * lpOverlapped, //[OUT]Cấu trúc WSAOVERLAPPED
   DWORD dwMilliseconds         //[IN]Thời gian đợi
);
```

37

Định nghĩa các cấu trúc

```
typedef struct{
   OVERLAPPED overlapped;           //Cấu trúc OVERLAPPED
   char       buffer[DATA_BUFSIZE]; //Buffer chứa dữ liệu
   int        bufferLen;            //Kích thước buffer
   int        operationType;        //Loại thao tác vào ra
} PER_IO_DATA, *LPPER_IO_DATA;
```

- Lưu ý: trường overlapped nên được khai báo đầu cấu trúc
- Trong trường hợp khác, sau khi gọi hàm **`GetQueuedCompletionStatus`**, cần sử dụng macro sau để xác định địa chỉ của vùng nhớ chứa dữ liệu cho con trỏ **`LPPER_IO_DATA`**

```
PCHAR CONTAINING_RECORD(
   PCHAR Address,
   TYPE  Type,
   PCHAR Field );
```

38

Overlapped – Completion port

```
// Simple server application that is capable of managing one
// socket request using completion routines

// Structure definition
typedef struct{
       WSAOVERLAPPED overlapped;
       WSABUF dataBuff;
       CHAR buffer[DATA_BUFSIZE];
       int bufLen;
       int recvBytes;
       int sentBytes;
       int operation;
} PER_IO_OPERATION_DATA, * LPPER_IO_OPERATION_DATA;

typedef struct{
       SOCKET socket;
} PER_HANDLE_DATA, * LPPER_HANDLE_DATA;

unsigned __stdcall serverWorkerThread(LPVOID CompletionPortID);
```

39

Overlapped – Completion port(tiếp)

```
int _tmain(int argc, _TCHAR* argv[])
{
   //Initiate Winsock 2.2...

   // Step 1: Setup an I/O completion port
   completionPort = CreateIoCompletionPort(INVALID_HANDLE_VALUE,
                       NULL, 0, 0);

   // Step 2: Determine how many processors are on the system
        GetSystemInfo(&systemInfo);

   // Step 3: Create worker threads
   for(i = 0; i < (int)systemInfo.dwNumberOfProcessors * 2; i++)
        _beginthreadex(0, 0, serverWorkerThread,
                       (void*)completionPort,  0, 0);

   // Step 4: Create a listening socket...

   // Step 5: Accept connections
```

40

Overlapped – Completion port(tiếp)

```
// Step 6: Create a socket information structure to associate with
the socket
perHandleData = (LPPER_HANDLE_DATA) GlobalAlloc(GPTR,
                                          sizeof(PER_HANDLE_DATA);

// Step 7: Associate the accepted socket with the original
completion port
perHandleData->socket = acceptSock;
CreateIoCompletionPort((HANDLE) acceptSock, completionPort, (DWORD)
                        perHandleData, 0);

// Step 8: Create per I/O socket information structure to associate
with the WSARecv call
ZeroMemory(&(perIoData->overlapped), sizeof(OVERLAPPED));
perIoData->dataBuff.len = DATA_BUFSIZE;
perIoData->dataBuff.buf = perIoData->buffer;
perIoData->operation = RECEIVE;
flags = 0;
WSARecv(acceptSock, &(perIoData->dataBuff), 1, &transferredBytes,
        &flags, &(perIoData->overlapped), NULL);

return 0;}//end main()
```

41

Overlapped – Completion port(tiếp)

```
unsigned __stdcall serverWorkerThread(LPVOID completionPortID)
{
   while(TRUE){
        HANDLE completionPort = (HANDLE) completionPortID;
        DWORD transferredBytes;
        LPPER_HANDLE_DATA perHandleData;
        LPPER_IO_OPERATION_DATA perIoData;

        GetQueuedCompletionStatus(completionPort, &transferredBytes,
                        (LPDWORD)&perHandleData,
                        (LPOVERLAPPED *) &perIoData, INFINITE);
        if (transferredBytes == 0 && (perIoData->operation == SEND
                || perIoData->operation == RECEIVE)){
        //close the socket

        //Process data and post other overlapped I/O Requets
}
```

42

Đóng completion port

- Mỗi completion port có thể sử dụng nhiều luồng điều khiển vào ra
- Tránh giải phóng cấu trúc OVERLAPPED trên một luồng, trong khi đang thực hiện vào ra

→Gọi hàm PostQueuedCompletionStatus() để gửi một packet có kích thước 0 tới completion port trên tất cả các luồng

→Gọi hàm CloseHandle() để đóng completion port

```
BOOL PostQueuedCompletionStatus(
    HANDLE CompletionPort,
    LPDWORD lpNumberOfBytesTransferred,//[IN] Số byte trao đổi
    PULONG_PTR lpCompletionKey,  //[IN]Định danh cho dữ liệu
    LPOVERLAPPED * lpOverlapped, //[IN]Cấu trúc WSAOVERLAPPED
);
```

43

TỔNG KẾT

44

Các kỹ thuật vào ra

- Kỹ thuật đa luồng:
  - Ưu điểm: đơn giản
  - Nhược điểm: Sử dụng tài nguyên không hiệu quả, không áp dụng cho ứng dụng phục vụ quá nhiều client
- Kỹ thuật thăm dò (select())
  - Ưu điểm: đơn giản
  - Nhược điểm:
    - Giới hạn bởi cấu trúc fd_set chỉ quản lý được 1024 socket
    - Quản lý nhiều socket: hàm select() không hiệu quả → không áp dụng cho ứng dụng phục vụ quá nhiều client

45

Các kỹ thuật vào ra(tiếp)

- Kỹ thuật vào ra theo thông báo:
  - Ưu điểm: đơn giản
  - Nhược điểm:
    - Yêu cầu ứng dụng phải có cửa sổ
    - Một cửa sổ trở thành nút thắt cổ chai trong ứng dụng nếu phải xử lý quá nhiều kết nối
- Kỹ thuật vào ra theo sự kiện:
  - Ưu điểm: đơn giản, không yêu cầu ứng dụng phải có cửa sổ
  - Nhược điểm: mỗi luồng chỉ quản lý được 64 bộ nghe sự kiện→ cần kết hợp kỹ thuật đa luồng để xử lý được nhiều kết nối hơn
- Kỹ thuật vào ra overlapped theo sự kiện:
  - Ưu điểm: hiệu năng cao
  - Hạn chế: mỗi luồng chỉ quản lý được 64 bộ nghe sự kiện

46

Các kỹ thuật vào ra(tiếp)

- Kỹ thuật vào ra overlapped, xử lý bằng completon routine
  - Ưu điểm: hiệu năng cao, không hạn chế số kết nối có thể xử lý
  - Hạn chế: completion routine không thực hiện được các tác vụ nặng
- Kỹ thuật vào ra overlapped theo completion port
  - Ưu điểm: hiệu năng cao, không hạn chế số kết nối có thể xử lý. Là mô hình phù hợp nhất cho các ứng dụng cần xử lý số kết nối lớn
  - Hạn chế: khó sử dụng ☺

47

Lựa chọn kỹ thuật nào?

- Client:
  - Overlapped I/O hoặc WSAEventSelect khi cần quản lý nhiều socket
  - Nếu ứng dụng có cửa sổ: WSAAsyncSelect là giải pháp tốt nhất
- Server: Overlapped I/O Completion port

48

Đọc thêm

- Which I/O Strategy Should I Use?

http://tangentsoft.net/wskfaq/articles/io-strategies.html

- The Lame List

http://tangentsoft.net/wskfaq/articles/lame-list.html

- Passing Sockets Between Processes

http://tangentsoft.net/wskfaq/articles/passing-sockets.html

49