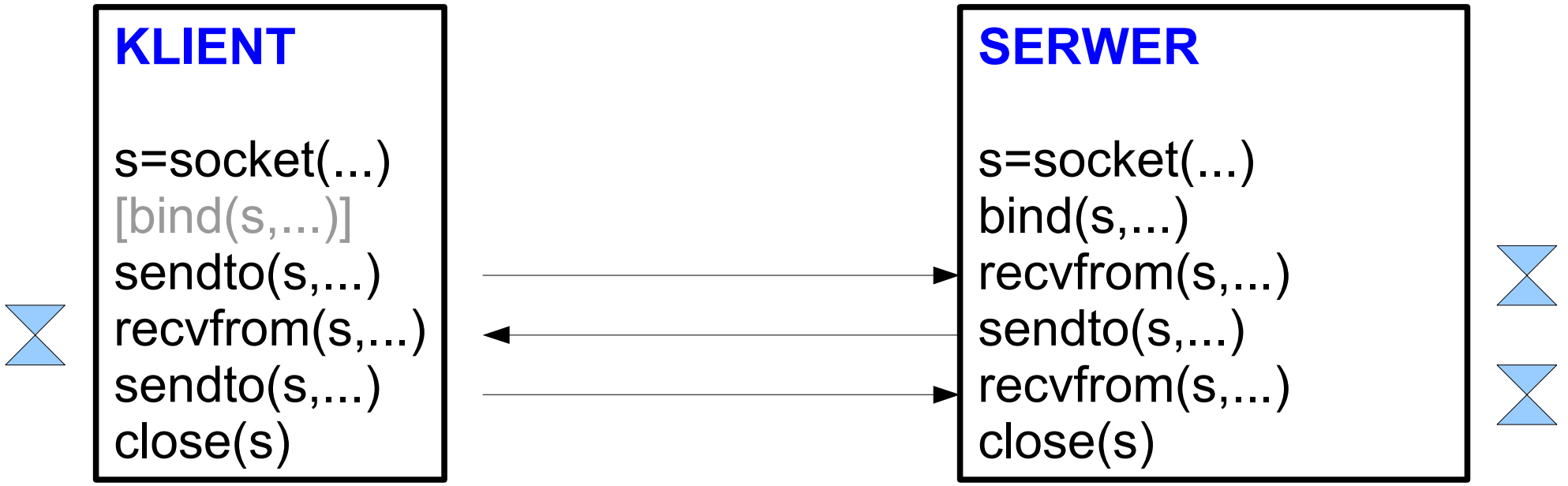
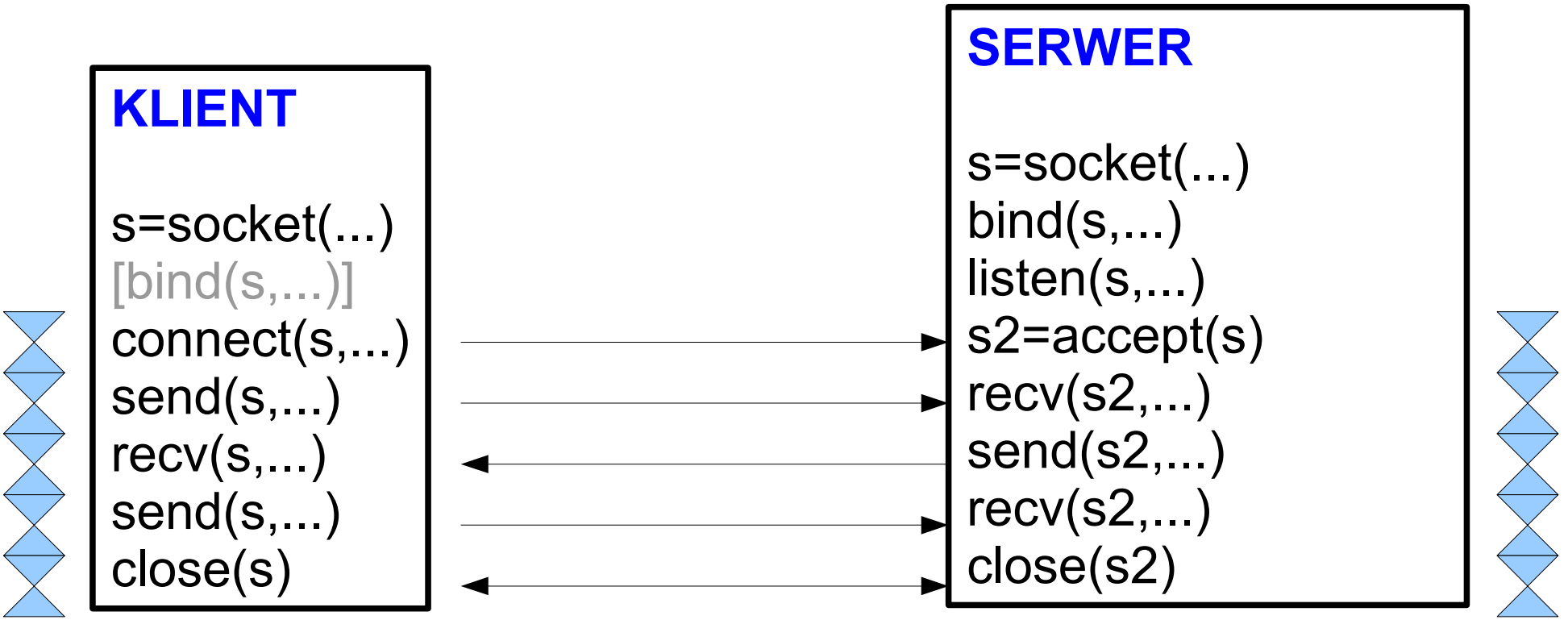
Schemat dla UDP

= możliwe zablokowanie aplikacji



Schemat dla TCP



Funkcja select

Problem:

Funkcje send, recv, sendto, recvfrom i accept są blokujące. Nie jest możliwa implementacja programu, który nasłuchuje jednocześnie na TCP i UDP i nie jest wielowątkowy.

Rozwiązanie:

Funkcja select.

Funkcja select c.d.

```
#include <sys/select.h>
```

lub

```
#include <unistd.h>
#include <sys/time.h>
int select( int n,
            fd_set *readfds,
            fd_set *writefds,
            fd_set *exceptfds,
            struct timeval *timeout)
```

Funkcja select c.d.

FD_CLR(int fd, fd_set *set)
Usuń deskryptor fd ze zbioru set.

FD_ISSET(int fd, fd_set *set)
Czy fd znajduje się w zbiorze set?

FD_SET(int fd, fd_set *set)
Dodaj fd do zbioru set.

FD_ZERO(fd_set *set)
Wyczyść cały zbiór set.

Funkcja select c.d.

```
struct timeval {
    long tv_sec;  /* sekundy */
    long tv_usec; /* mikrosekundy */
}
```

Funkcja select c.d.

- Funkcja select zwraca liczbę deskryptorów, na których zaszły zdarzenia.
- Dany zbiór deskryptorów może być ignorowany (podajemy NULL).
- Podanie NULL jako czasu oczekiwania powoduje natychmiastowy powrót.
- Podanie zerowego czasu oczekiwania oznacza czekanie do skutku.

Funkcja select c.d.

- Zwracana wartość to liczba deskryptorów, na których wystąpiło zdarzenie. 0 oznacza przeterminowanie (minął czas a nic się nie wydarzyło), a -1 to błąd (np. nieprawidłowe deskryptory w zbiorach).
- Dzięki select możliwy jest nasłuch na wielu gniazdach jednocześnie.

Funkcja select - przykład

```
[...]
int dtcp, dudp, maxd, ret;
struct timeval tv;
fd_set zbior;
dtcp = socket(AF_INET, SOCK_STREAM, 0);
dudp = socket(AF_INET, SOCK_DGRAM, 0);

[ bind, listen na dtcp
  ale NIE accept]
```

Funkcja select c.d.

```
maxd = (dtcp > dudp ? dtcp : dudp);
tv.tv_sec = tv.tv_usec = 0;
FD_ZERO(&zbior);
FD_SET(dtcp, &zbior);
FD_SET(dudp, &zbior);
ret = select( maxd, &zbior, NULL,
                    NULL, &tv );
```

Funkcja select c.d.

```
if (ret > 0) {
    if (FD_ISSET(dtcp, &zbior)) {
        [accept, send, recv, ...]
    }
    if (FD_ISSET(dudp, &zbior)) {
        [recvfrom, sendto, ...]
    }
}
```

Biblioteka WinSock

Implementacja w wsock32.dll.
Konieczna inicjalizacja i zwolnienie zasobów.

#include <winsock2.h>

WSAStartup(WORD wersja, WSADATA *info)

info.iMaxSockets – maksymalna liczba gniazd
info.iMaxUdpDg – maksymalna długość datagramu

WSACleanup()

Winsock - kompilacja

Należy linkować z wsock32:

- Kompilatory GNU (np. MinGW):
  gcc -o wykonywalny zrodlo.c -lwsock32
- Kompilator Borland C++:
  bcc32 zrodlo.c wsock32.lib
- Kompilator Visual C++:
  cl /Fewykonywalny zrodlo.c wsock32.lib

Winsock

Typ deskryptora gniazda: SOCKET
Tworzenie gniazda: socket
Kojarzenie z portem: bind
Nasłuch TCP: listen
Akceptowanie połączenia TCP: accept
Nawiązanie połączenia TCP: connect
Wysyłanie / odbieranie danych: send, recv
Komunikacja UDP: sendto, recvfrom
Zamknięcie gniazda: closesocket

Winsock

BSD / Unix:
Deskryptory gniazd sieciowych są traktowane tak samo jak inne deskryptory w systemie (możliwe wysyłanie i odbieranie danych za pomocą read i write, zamknięcie gniazda przez close itd.).

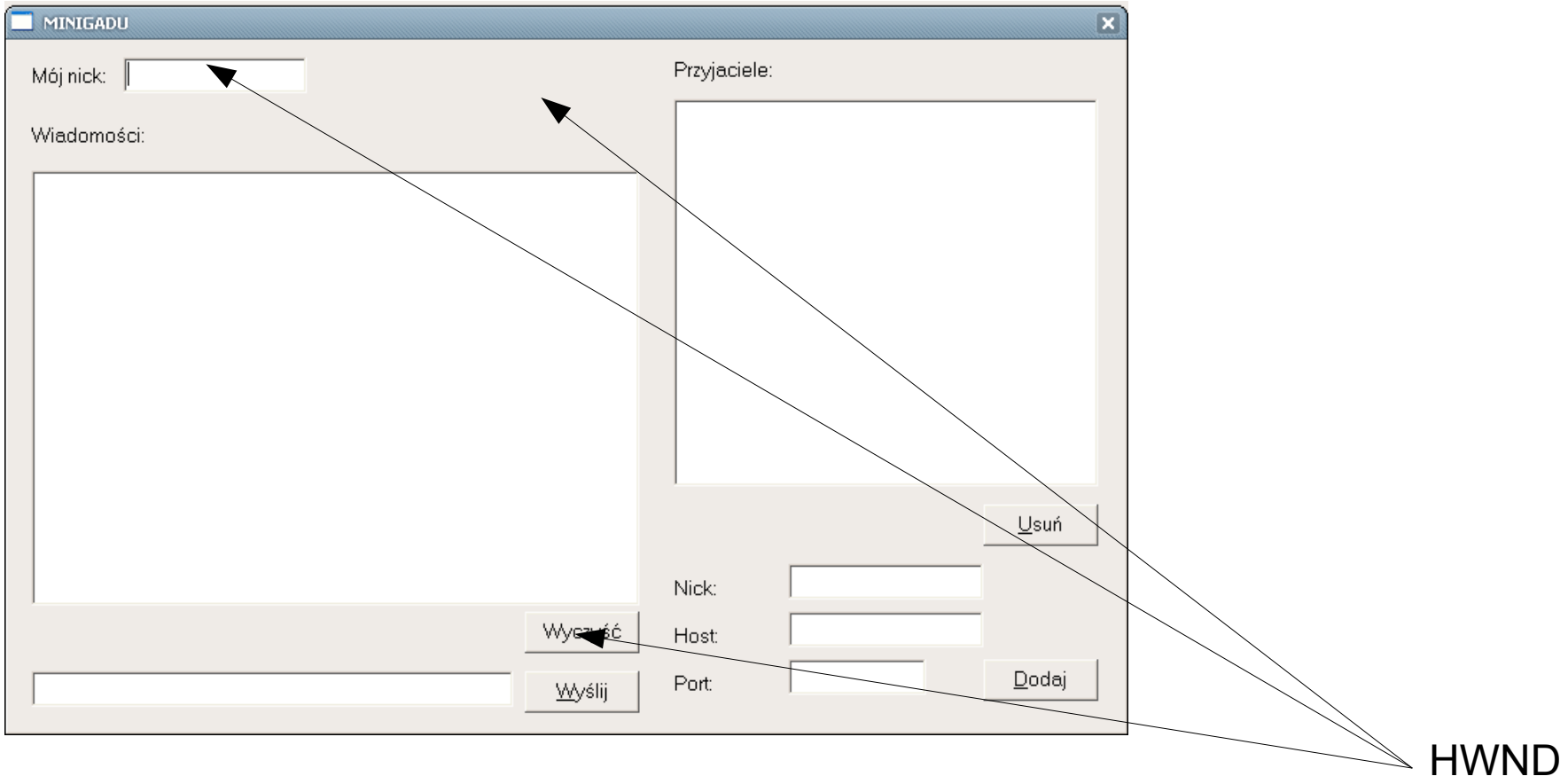
Winsock:
Deskryptory gniazd sieciowych i deskryptory plików zachowują się różnie (dlatego np. nie można zamykać gniazd za pomocą close).

Winsock

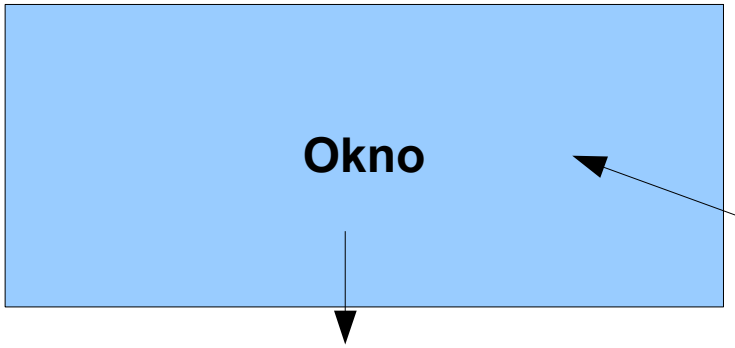
Dwa zestawy funkcji:

- zgodne z gniazdami BSD (socket, connect, listen, inet_addr, htonl, ...) - zgodność prawie w 100% (wyjątki: typ SOCKET, funkcja closesocket, wartości zwracane przez niektóre funkcje)
- specyficzne dla API Win32 (nazwy zaczynają się od WSA, np. WSAStartup, WSAAsyncSelect)

Win32 - okna


MINIGADU
Mój nick:
Przyjaciele:
Wiadomości:
Usuń
Nick:
Host:
Port:
Wyczyść
Wyślij
Dodaj
HWND

Obsługa komunikatów



PĘTLA KOMUNIKATÓW

```
while (GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

OBSŁUGA KOMUNIKATÓW OKNA

```
switch (uMsg)
{
    case WM_INITDIALOG:
        return TRUE;

    case WM_COMMAND:
        switch (wParam)
        {
            case IDOK:
            case IDCANCEL:
                EndDialog(hDlg, TRUE);
                return TRUE;
        }
}
return FALSE;
```

Komunikat

(UINT typ, WPARAM par1, LPARAM par2)

typ: liczba całkowita (rodzaj komunikatu)
par1: liczba całkowita 16-bitowa
par2: liczba całkowita 32-bitowa (zwykle adres bufora)

Rodzaje komunikatów

- Komunikaty systemu dla okien, np. WM_PAINT, WM_CLOSE, WM_MOUSEDOWN, WM_COMMAND ...
- Komunikaty użytkownika: WM_USER, WM_USER+1, WM_USER+2, ...
- Komunikaty interakcji komponentów, np. (dla komponentu typu lista) LB_GETTEXT, LB_GETCURSEL, LB_ADDSTRING ...

Źródła komunikatów

Skąd komunikaty biorą się w kolejce komunikatów okna?

- Wstawiane przez system (np. WM_KEYDOWN, WM_MOUSEDOWN)
- Wysyłane przez aplikację za pomocą SendMessage (np. LB_ADDSTRING, WM_SETFONT)

Pętla komunikatów

dopóki są komunikaty w kolejce

    m := pierwszy komunikat z kolejki
    w := okno, w którym wystąpił komunikat
    wynik := false

    dopóki wynik=false i w jest oknem
        wynik := funkcja obsługi komunikatów w
                  oknie w dla komunikatu m
        jeśli wynik = false to w := właściciel(w)

Funkcje blokujące w Win32

Jeśli aplikacja okienkowa Win32 uruchomi funkcję blokującą (np. recv, recvfrom), to pętla komunikatów nie jest obsługiwana do momentu zakończenia zablokowanej funkcji.

Skutek: program nie reaguje ani na zdarzenia użytkownika (kliknięcia, wciskanie klawiszy), ani na zdarzenia systemu (odświeżanie zawartości, zamknięcie). Dla użytkownika program wygląda jakby się „zawiesił”.

Funkcja WSAAsyncSelect

WSAAsyncSelect(gniazdo, okno, komunikat, zdarzenie)

SOCKET gniazdo – gniazdo, które będzie monitorowane

HWND okno – okno, które otrzyma komunikat gdy wystąpi zdarzenie

UINT komunikat – rodzaj wysyłanego komunikatu (zwykle WM_USER+x)

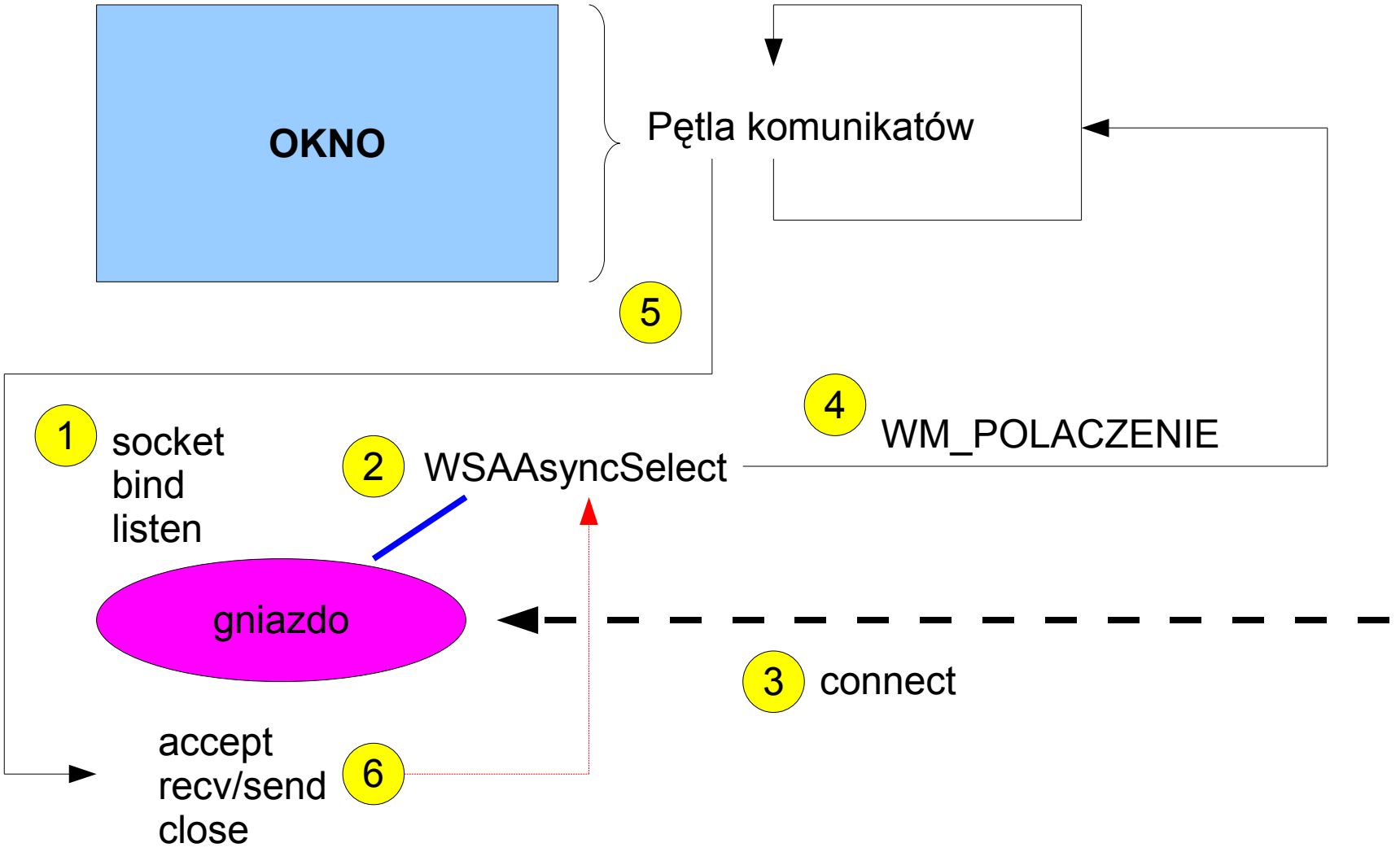
long zdarzenie – rodzaj zdarzenia (FD_ACCEPT, FD_CONNECT, FD_READ, FD_WRITE, FD_CLOSE)

Funkcja WSAAsyncSelect

WSAAsyncSelect(s, hWnd, POLACZENIE,
FD_ACCEPT)

Skutek:
W momencie, gdy na gnieździe s wystąpi zdarzenie FD_ACCEPT (czyli gdy pojawi się przychodzące połączenie sieciowe), do okna o uchwycie hWnd zostanie wysłany komunikat POLACZENIE. Komunikat ten powinien zostać obsłużony w pętli komunikatów okna hWnd. Reakcją na POLACZENIE może być np. wykonanie accept, wysłanie / odebranie danych i zamknięcie sesji TCP.

WSAAsyncSelect



Funkcja WSAAsyncSelect

Korzyść:

WSAAsyncSelect jest funkcją nieblokującą (wraca natychmiast). Pozwala na monitorowanie stanu gniazda i jednoczesne przetwarzanie kolejki komunikatów. Dzięki temu aplikacja oczekująca na połączenie sieciowe poprawnie reaguje na wszystkie komunikaty.

Gniazda w Javie

Pakiet java.net: klasy gniazd TCP i UDP, adresy internetowe, datagramy.

Pakiet java.io: strumienie wejściowe i wyjściowe.

Najważniejsze klasy

- java.net.InetAddress
- java.net.Socket
- java.net.ServerSocket
- java.net.DatagramSocket
- java.net.DatagramPacket
- java.io.InputStream
- java.io.OutputStream
- java.io.DataInputStream
- java.io.DataOutputStream

java.net.InetAddress

Nie posiada konstruktora.

Metody statyczne:
InetAddress getByName(String nazwa/adres)
InetAddress getLocalHost()

Inne metody:
String getHostName()
byte[] getAddress()

java.net.Socket

Reprezentuje gniazdo TCP strony klienta.

Konstruktor (jeden z wielu):

Socket(InetAddress adres, int port)

Tworzy gniazdo i wykonuje połączenie (connect) na podany adres i numer portu.

java.net.Socket

java.io.InputStream getInputStream()
java.io.OutputStream getOutputStream()

Metody pobierają strumienie: wejściowy i wyjściowy skojarzone z gniazdem.
Zwykle na InputStream i OutputStream budujemy wygodniejsze strumienie (np. DataInputStream i DataOutputStream lub ObjectInputStream i ObjectOutputStream).

java.net.Socket

Schemat:

```
InetAddress adres =
    InetAddress.getByName("atos.wmid.amu.edu.pl");
Socket s = new Socket(adres, port);
DataOutputStream dos =
    new DataOutputStream(s.getOutputStream());
DataInputStream dis =
    new DataInputStream(s.getInputStream());
dos.writeUTF("tekst");
double d = dis.readDouble();
dis.close(); dos.close();
s.close();
```

java.net.ServerSocket

Konstruktor (jeden z wielu):

ServerSocket(int port)

Tworzy gniazdo TCP nasłuchujące na porcie o numerze port.

java.net.ServerSocket

Schemat:

```
ServerSocket ss = new ServerSocket(4500);

while (warunek stopu) {
    Socket s = ss.accept();

    [ strumienie na s, operacje wejścia/wyjścia,
      jak dla gniazda klienta ]

    s.close();
}
ss.close();
```

java.net.DatagramSocket

Konstruktory:

DatagramSocket(int port)
Tworzy gniazdo UDP skojarzone z lokalnym portem o numerze port.

DatagramSocket()
Tworzy gniazdo UDP skojarzone z lokalnym portem o numerze nadanym przez system operacyjny.

java.net.DatagramSocket

send(DatagramPacket datagram)
receive(DatagramPacket datagram)

Wysyła / odbiera datagram. W obu przypadkach parametr musi być wcześniej utworzonym datagramem (patrz dalej).

java.net.DatagramPacket

Konstruktory:

DatagramPacket(byte[] bufor, int długość)
Tworzy obiekt, który może być użyty w DatagramSocket.receive(...).

DatagramPacket(byte[] bufor, int długość, InetAddress adres docelowy, int port docelowy)
Tworzy obiekt, który może być użyty w DatagramSocket.send(...)

java.net.DatagramPacket

Inne metody:
byte[] getData() - pobierz dane z datagramu
void setData(byte[] dane) – ustaw dane
void setAddress(InetAddress adr) – ustaw adres docelowy
void setPort(int port) – ustaw port docelowy
SocketAddress getSocketAddress() - pobierz adres nadawcy datagramu

int SocketAddress.getPort()
InetAddress SocketAddress.getAddress()

java.io.InputStream

int read() - czytaj jeden bajt

int read(byte[] bufor) – czytaj co najwyżej bufor.length bajtów

close() - zamknij strumień

java.io.OutputStream

write(int b) – zapisz jeden bajt do strumienia

write(byte[] bufor) – zapisz zawartość bufora do strumienia

close() - zamknij strumień

java.io.DataInputStream

DataInputStream(InputStream in)

boolean readBoolean()
byte readByte()
char readChar()
double readDouble()
int readInt()
String readUTF()

void close()

java.io.DataOutputStream

DataOutputStream(OutputStream out)

void writeBoolean(boolean v)
void writeByte(int v)
void writeChar(int v)
void writeDouble(double v)
void writeInt(int v)
void writeUTF(String v)

void close()

Wyjątki

Praktycznie wszystkie operacje na gniazdach i strumieniach mogą potencjalnie wyrzucić wyjątek w przypadku błędu.

Wyjątki takie należy przechwytywać i obsługiwać w aplikacji. W niektórych przypadkach program nie da się nawet skompilować z powodu nieprzechwyconych wyjątków.

Przechwytywanie wyjątków

```
try {
    [ operacje na gniazdach / strumieniach ]
} catch (Exception e) {
    [ obsługa wyjątku e ]
}
```