Sieci komputerowe i programowanie sieciowe

3. Komunikacja między procesami na różnych maszynach (gniazda)

Gniazdo (socket) – abstrakcyjny, dwukierunkowy (wysyłanie i odbieranie) punkt końcowy połączenia.

Gniazda są używane, aby procesy na różnych maszynach mogły się komunikować przez sieć. Możliwa jest także komunikacja w ramach tej samej maszyny.

W praktyce pojęcia gniazd używa się najczęściej w kontekście protokołów IP i TCP/UDP (w dalszej części referatu traktuje o takich gniazdach).

Gniazdo posiada
- typ
- adres lokalny
- opcjonalnie lokalny numer portu

gniazdo = {adres, port, typ_protokołu}

Na czas trwania komunikacji gniazdo może posiadać dodatkowo:
- adres zdalny
- zdalny numer portu

Wyjaśnienie:
- adres – węzeł sieci
- port – oznacz proces w węźle (liczba od 0 do 65535)
- typ gniazda – protokół (sposób) wymiany danych

Na jednej maszynie można utworzyć dwa gniazda o tym samym porcie (w takim przypadku jedno z nich musi być TCP, a drugie UDP).

Gniazdem może być też specjalny plik (wtedy gniazdo nie posiada adresu i portu, do komunikacji potrzebna jest nazwa pliku).

Jeżeli gniazdo używa portów, to lokalny numer portu może być przydzielony automatycznie (efemeryczny numer portu), lub określony (bind) przez twórcę aplikacji.

W systemach Unix obsługa gniazd jest zaimplementowana w jądrze.
Proces może odwołać się do gniazda za pomocą deskryptora. Do obsługi gniazd system operacyjny udostępnia API (plik nagłówkowy <sys/socket.h>).

Porty
- 0 - 1023 zarezerwowane na standardowo przypisane do nich usługi (*well known ports*) np. SSH – 22/TCP, HTTPS - 443/TCP, TFTP - 69/UDP.
  Plik /etc/services zawiera informacje jakich portów używają usługi.
- 1024 - 49151 - porty zarejestrowane
- 49152 - 65535 - dynamiczne/prywatne – do dowolnego użytku

Scenariusz komunikacji

1. Serwer tworzy gniazdo

   int socket(int domain, int type, int protocol);

   oraz (w przypadku protokołu TCP) rejestruje gniazdo w systemie

   int bind(int sockfd, struct sockaddr * my_addr, int addrlen);

Następnie zaczyna nasłuchiwać int listen(int sockfd, int backlog);

2. Klient tworzy gniazdo i nawiązuje połączenie

int connect(int sockfd, const struct sockaddr * addr, socklen_t addrlen);

3. Serwer akceptuje połączenie (funkcja blokująca)

int accept(int s, struct sockaddr * addr, socklen_t * addrlen);

4. Przesyłanie danych

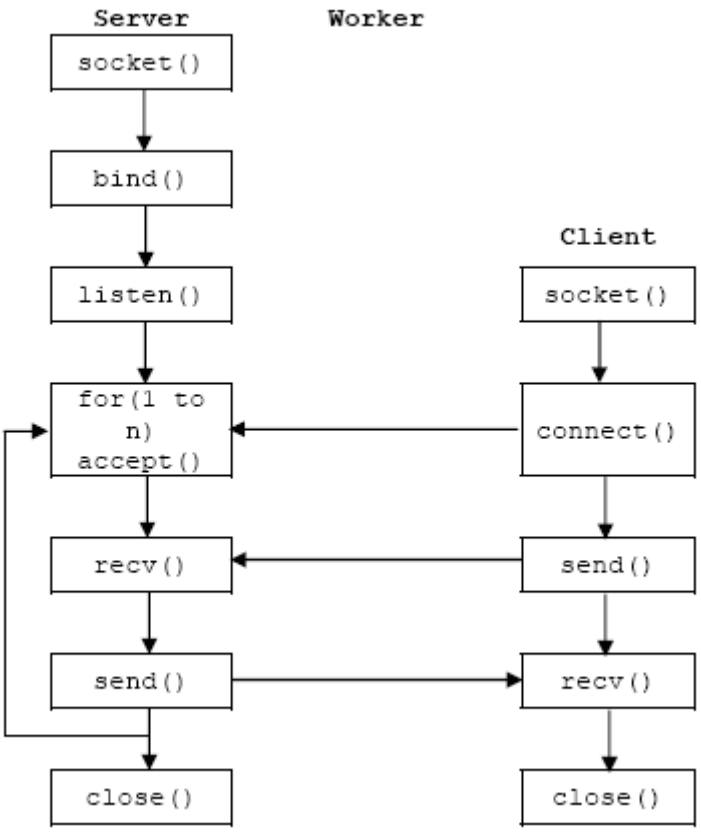
TCP (domyślnie blokujące)

ssize_t send(int sockfd, const void * buf, size_t len, int flags);
ssize_t recv(int sockfd, void * buf, size_t len, int flags);

UDP i TCP (domyślnie nieblokujące)

ssize_t sendto(int sockfd, const void * buf, size_t len, int flags, const struct sockaddr * dest_addr, socklen_t addrlen);
ssize_t recvfrom(int sockfd, void * buf, size_t len, int flags, struct sockaddr * src_addr, socklen_t * addrlen);

5. Zamknięcie połączenia int shutdown(int sockfd, int flag);



http://www.cs.put.poznan.pl/ddwornikowski/sieci/sieci2/bsdsockets.html
https://pl.wikipedia.org/wiki/Gniazdo_(telekomunikacja)
https://pl.wikipedia.org/wiki/Port_protoko%C5%82u
https://en.wikipedia.org/wiki/Network_socket
http://cpp0x.pl/artykuly/?id=66
https://en.wikipedia.org/wiki/Berkeley_sockets
https://pl.wikipedia.org/wiki/Protok%C3%B3%C5%82_sterowania_transmisj%C4%85
https://pl.wikipedia.org/wiki/User_Datagram_Protocol
http://home.agh.edu.pl/~balis/dydakt/sr/lab/gniazda2009.pdf