Gniazda:

Jak identyfikować procesy na różnych maszynach? Nie można za pomocą numeru procesu, gdyż jest to liczba zmienna. Takowosz stworzono gniazda.

Gniazda to pojęcie abstrakcyjne reprezentujące dwukierunkowy punkt końcowy połączenia, którego ideą jest dostarczenie komunikacji miedzy dwoma procesami znajdującymi się na tej samej, bądź różnych maszynach. Można je identyfikować jako zbiór cech procesu komunikującego się z innym procesem:

Adres lokalny – identyfikator maszyny na której działa nasz proces (IPv4, IPv6, UNIX)
Port lokalny – liczba dodatnia identyfikująca proces na maszynie, 0 dla gniazd UNIX
Protokół – typ gniazda identyfikujący sposób wymiany danych (UDP, TCP, surowy)

W trakcie komunikacji dodatkowo mogą dojść dwa atrybuty:

Adres zdalny – identyfikator maszyny, z która nawiązane jest połączenie
Port zdalny – port przez który identyfikujemy proces na dalekiej maszynie

3 rodzaje adresów i protokołów dają 9 możliwych scenariuszy komunikacji. Dwa procesy są w stanie się połączyć wtedy i tylko wtedy gdy używają tej samej rodziny gniazd i tego samego protokołu.

Mamy dwa rodzaje komunikacji sieciowej:

1. Połączeniowa – działa za pomocą protokołu TCP
2. Bezpołączeniowa – z użyciem protokołu UDP

Schematy komunikacji
Ad 1:

Pierwszym krokiem jest utworzenie, zarówno przez serwer jak i klienta, własnych punktów końcowych transportu do nawiązania komunikacji (gniazd) będących uchwytami do utworzenia połączenia sieciowego (lokalnego) pomiędzy procesami [socket()], serwer musi także stowarzyszyć gniazdo z adresem maszyny na jakim będzie oczekiwał zgłoszeń [bind()]. Następnie serwer przechodzi w stan oczekiwania tworząc kolejkę klientów [listen()]. Wtedy klient może wysłać do serwera prośbę o nawiązanie połączenia [connect()]. Gdy w kolejce serwera znajdują się jakieś zgłoszenia wybiera on jedno z nich (pierwsze) i tworzy dla niego nowe gniazdo służące do obsługi połączenia tylko i wyłącznie z wybranym procesem [accept()].W ten sposób poprzednie gniazdo dalej może być wykorzystywane do przyjmowania połączeń.

Jak już mamy połączone procesy możemy swobodnie wymieniać między nimi dane [read()/ write()] w obie strony bez obawy, że coś nam ucieknie. W momencie rezygnacji z dalszej wymiany bitów ładnie jest zamknąć używane gniazda [close()/shutdown()].

Serwer	Klient	Funkcje UNIX
Tworzenie gniazda	Tworzenie gniazda	socket()
Nałożenie adresu na gniazdo		bind()
Oczekiwanie na połączenie		listen()
	Prośba o połączenie	connect()
Akceptacja zgłoszenia		accept()
Wymiana danych	Wymiana danych	read()/write()

Ad 2:

W tym schemacie także niezbędne jest stworzenie gniazd lokalnych po obu stronach komunikacji i nałożenie na nie adresów lokalnych. W tej wersji serwer nie tworzy kolejki dla połączeń oczekujących. Już w tym momencie można zacząć transmisje danych [sendto()/recvfrom()]. Serwer każdorazowo chcąc wysyłając odpowiedź musi zapamiętać z jakiego adresu przyszła do niego informacja.

Serwer	Klient	Funkcje UNIX
Założenie gniazda	Założenie gniazda	socket()
Nałożenie adresu na gniazdo	Nałożenie adresu na gniazdo	bind()
	Wysłanie danych	sendto()
Odebranie danych		recvfrom()
Wysłanie odpowiedzi		sendto()
	Odebranie odpowiedzi	recvfrom()

Alternatywnie istnieje możliwość utworzenia gniazda UDP połączonego (wywołaniem funkcji [connect()], adres zdalny użyty podczas wywołania tej funkcji staje się naszym jedynym adresem zdalnym do wymiany danych. Ponowne wywołanie funkcji connect() prowadzi do określenia nowego punktu docelowego lub rozłączenia gniazda w zależności od wybranych parametrów.

Błąd asynchroniczny – sytuacja w której klient próbuje się połączyć z serwerem, który jeszcze nie został uruchomiony. Klient wysyła datagram, a system odpowiada błędem ICMP port unreachable, jednak klient nie otrzyma go gdyż funkcja sendto() kończy swoje działanie w momencie wysłania datagramu. Błąd ten nie występuje w podczas używania gniazd UDP połączonych.

W wypadku gniazd surowych dane które transmitujemy nie są pakowanie/rozpakowywane w nagłówki protokołów, które trzeba samemu stworzyć/zinterpretować.

Serwery:
iteracyjny – obsługa tylko jednego klienta w danej chwili, każde zapytanie czeka, aż wykonane zostanie poprzednie
współbieżny – wielu klientów może być obsługiwanych w tym samym czasie, dla każdego klienta tworzony osobny wątek/proces zajmujący się obsługa danego zgłoszenia

xinetd - superserwer, który wykonuje początkowe fragmenty kodów wspólne dla wielu programów i czeka w uśpieniu nasłuchując na wielu portach [select()] na zgłoszenia klientów, gdy takowe zgłoszenie się pojawi demon odpala odpowiednią usługę przekazuje jej obsługę klienta.

W odwodzie gniazd mamy XTI i TLI. Wykorzystują one strumienie.
Strumienie – takie cudo zapewniające dwukierunkową komunikację między procesem użytkownika a procedurą jądra obsługującą urządzenie (np. karta sieciowa) lub pseudo-urządzenie (program – procedura jądra). Dostęp do urządzenia mamy dzięki czołu strumienia, do którego ślemy odpowiednio sformatowane dane.