Instytut Teleinformatyki

Wydział Inżynierii Elektrycznej i Komputerowej
Politechnika Krakowska

programowanie usług sieciowych

„Dziedzina Unix”

laboratorium: 06

Kraków, 2014

Spis treści

1. Wiadomości wstępne

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące **protokołów dziedziny Unix**. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

1.1. Tematyka laboratorium

Tematyką laboratorium jest programowanie aplikacji klient-serwer w oparciu o **protokoły dziedziny Unix**. Protokoły te nie stanowią rzeczywistej rodziny protokołów, lecz jedynie pewien sposób prowadzenia komunikacji między procesami klienta i serwera wykonywanymi **w tej samej stacji**. Umożliwiają stosowanie tego samego interfejsu API (ang. A*pplication Program Interface*), który służy do programowania klientów i serwerów wykonywanych w różnych stacjach.

Istnieją dwa rodzaje gniazd w dziedzinie Unix:

- *strumieniowe* – przypominające gniazda TCP,
- *datagramowe* – przypominające gniazda UDP.

Gniazda identyfikujemy za pomocą nazw ścieżkowych w obrębie zwykłego systemu plików. Nazwy te nie odnoszą się do zwykłych plików systemu Unix. Nie można pobierać danych z tych plików ani do nich zapisywać w zwyczajny sposób. Może to robić jedynie program, który powiązał taką nazwę ścieżkową z gniazdem w dziedzinie Unix.

1.2. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi aplikacji typu **klient - serwer**:

- Jak wygląda połączenie.

A także z zagadnieniami dotyczącymi **gniazd**:

- Co to jest gniazdo;
- Jak wygląda gniazdo w dziedzinie Unix;
- Działanie funkcji `socet()`, `bind()`, `listen()`, `accept()`, `connect()` oraz `close()`.

Literatura:

[1] W.R. Stevens, *„Programowanie Usług Sieciowych”*, *„API: gniazda i XTI”*.

[2] N. Matthew, R.Stones *„Linux programowanie”*.

1.3. Cel laboratorium

Celem laboratorium jest zapoznanie się z możliwym wykorzystaniem gniazd w dziedzinie Unix. Podczas realizacji tego laboratorium zapoznasz się z:

- sposobem komunikacji pomiędzy klientem i serwerem (uruchomionych na tej samej stacji) przy użyciu gniazd dziedziny Unix,
- z możliwościami gniazd w dziedzinie Unix,
- z technikami używania gniazd w dziedzinie Unix.

2. Przebieg laboratorium

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

2.1. Przygotowanie laboratorium

Przed przystąpieniem do dalszej części laboratorium należy skopiować cała zawartość katalog `/home/shared/pus/pus_06_unixdomain` do katalogu `~/pus/pus_06_unixdomain`.

2.2. Zadanie 1. Dowiązanie nazwy ścieżkowej do gniazda w dziedzinie Unix.

Treścią zadania jest przeanalizowanie programu wykorzystującego funkcję `bind` do dowiązania nazwy ścieżkowej podanej w linii poleceń do gniazda w dziedzinie Unix.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Przejdź do katalogu ze źródłem programu:
   ```
   $ cd ~/pus/pus_06_unixdomain
   ```
2. Skompiluj źródło:
   ```
   $ g++ unixbind.c –o unixbind
   ```
3. Uruchom program podając w wierszu poleceń nazwę ścieżkową:
   ```
   $ ./unixbind /tmp/b
   ```
4. Wynikiem działania tego programu jest wypisanie na ekran naszej dowiązanej ścieżki i rozmiaru jej struktury gniazdowej:

   ```
   bound name = /tmp/b, returned len = 9
   ```
5. Analiza kodu:


```
(01)#include        "unp.h"
(02)int
(03)main(int argc, char **argv)
(04) {
(05)    int             sockfd;
(06)    socklen_t       len;
(07)    struct sockaddr_un      addr1, addr2;


(10)    sockfd = socket(AF_LOCAL, SOCK_STREAM, 0);

(11)    unlink(argv[1]);                /* OK if this fails */

(12)    bzero(&addr1, sizeof(addr1));

(13)    addr1.sun_family = AF_LOCAL;
(14)    strncpy(addr1.sun_path, argv[1], sizeof(addr1.sun_path)-1);
(15)    bind(sockfd, (SA *) &addr1, SUN_LEN(&addr1));

(16)    len = sizeof(addr2);
(17)    getsockname(sockfd, (SA *) &addr2, &len);
(18)    printf("bound name = %s, returned len =d\n",addr2.sun_path,
len);
(19)       exit(0);
(20) }
```

Numer	Funkcja	Opis
11	`unlink`	Usuwa nazwę ścieżkową
14	`strncpy`	Kopiuje wiersz poleceń
15	`bind`	Dowiązuje nazwę ścieżkową do gniazda
17	`getsockname`	Pobiera tą samą nazwę ścieżkową, którą przed chwilą dowiązaliśmy

Tabela Opis funkcji wykorzystanych w programie

2.3. Zadanie 2. Datagramowa obsługa komunikacji k-s w dziedzinie Unix.

Przeanalizuj program i działanie klienta i serwera echa UDP, korzystający z gniazd w dziedzinie Unix.

W programie klienta:

- obie gniazdowe struktury mają typ danych `sockaddr_un`,
- pierwszy argument przekazywany do funkcji `socket` ma wartość `AF_LOCAL`, tworząc gniazdo datagramowe w dziedzinie Unix,
- zdefiniowana w pliku `unp.h` stała `UNIXDG_PATH` ma wartość `/tmp/unix.dg`,
- inicjujemy gniazdową strukturę adresową, którą przekazujemy do funkcji `bind`,
- wywołujemy funkcję `dg_echo`.

W programie serwera:

- gniazdową strukturą adresową jest struktura `sockaddr_un`,
- przekazujemy do funkcji `socket` argument AF_LOCAL,
- w sposób jawny wywołujemy funkcję `bind`, dowiązując nazwę ścieżkową do naszego gniazda, tak aby serwer otrzymał nazwę ścieżkową do której mógłby wysłać odpowiedzi,
- wypełniamy nazwą ścieżkową przydzieloną serwerowi gniazdową strukturę adresową,
- używamy funkcji `dg_cli`.

W celu uruchomienia programu:

1. Przejdź do katalogu `~/pus/pus_06_unixdomain/dg`:
   ```
   $ cd ~/pus/pus_06_unixdomain/dg
   ```
2. Uruchom plik do kompilacji `Makefile`:
   ```
   $ . Makefile
   ```
3. Uruchom serwer korzystający z gniazd w dziedzinie Unix;
   ```
   $ ./unixdgserv01
   ```
4. Uruchamiamy klienta UDP:
   ```
   $ ./unixdgcli01
   ```

Następnie wpiszmy dowolny tekst w programie klienta i zaobserwujmy co się stało podczas wysłania komunikatu do serwera.

2.4. Zadanie 3. Przykład przekazania deskryptora

Zadanie polega na przeanalizowaniu programu `mycat.c`, który będzie pobierał z wiersza poleceń nazwę ścieżkową, otwierał plik i kopiował zawartość do standardowego wyjścia. Zamiast wywoływać funkcję `open`, wywołujemy naszą funkcję `my_open`. Funkcja utworzy łącze strumieniowe i wywoła funkcje `fork` oraz `exec`, aby uruchomić inny program, który otworzy żądany plik. Ten inny program przekazuje otwarty deskryptor z powrotem do procesu macierzystego, używając łącza strumieniowego.

Uruchomienie przez proces innego programu w celu otworzenia pliku ma taką zaletę, że dla programu można ustawić identyfikator użytkownika jako root. Zatem nasz program może otrzymać uprawnienia do otwierania dowolnych plików, czyli takich, które normalnie nie mielibyśmy prawa otworzyć.

W celu uruchomienia programu:

1. Przejdź do katalogu `~/pus/pus_06_unixdomain/mycat`:

   ```
   $ cd ~/pus/pus_06_unixdomain/mycat
   ```

2. Uruchom plik do kompilacji `Makefile`:

   ```
   $ . Makefile
   ```

3. Uruchom program, który kopiuje plik do wyjścia standardowego:

   ```
   $ ./mycat sciezka_do_pliku
   ```


3. Opracowanie i sprawozdanie

Realizacja laboratorium pt. „Dziedzina Unix” polega na wykonaniu wszystkich zadań programistycznych podanych w drugiej części tej instrukcji. Wynikiem wykonania powinno być sprawozdanie w formie wydruku papierowego dostarczonego na kolejne zajęcia licząc od daty laboratorium, kiedy zadania zostały zadane.
Sprawozdanie powinno zawierać:

- opis metodyki realizacji zadań (system operacyjny, język programowania, biblioteki, itp.),
- algorytmy wykorzystane w zadaniach (zwłaszcza, jeśli zastosowane zostały rozwiązania nietypowe),
- opisy napisanych programów wraz z opcjami,
- trudniejsze kawałki kodu, opisane tak, jak w niniejszej instrukcji,
- uwagi oceniające ćwiczenie: trudne/łatwe, nie/realizowalne podczas zajęć, nie/wymagające wcześniejszej znajomości zagadnień (wymienić jakich),
- wskazówki dotyczące ewentualnej poprawy instrukcji celem lepszego zrozumienia sensu oraz treści zadań,
- itp.

