Übungsblatt 6

Übung zur Betriebssystemarchitektur
WS 2004/05

Dipl.-Inf. Bernhard Rabe
Betriebssysteme & Middleware

Inhalt

- Übungsblatt 6
  - FTP Protokoll RFC 959
- Socket API
- Programmbeispiele

FTP Protokoll

- RFC 959 , 1985
- Kontroll-Verbindung auf Serverport 21
- Datenverbindungen werden ausgehandelt
  - Passiv: Der Server öffnet den Server Socket
    - funktioniert nicht bei Servern hinter Firewall oder NAT
    - Aktiv: Der Klient öffnet den Server Socket
      - funktioniert nicht bei Klienten hinter Firewall oder NAT

FTP Kommandos

- 3-4 Zeichen lang
- Parameter mit einem Leerzeichen getrennt
- mit CRLF abgeschlossen
- PWD\r\n
- TYPE<Space>A\r\n
- Groß- Kleinschreibung nicht spezifiziert!

FTP Replies

- 3 –Stellige Zahl als Status
- Zusätzliche Beschreibung mit einem Leerzeichen oder Bindestrich getrennt
- Abschluß mit CRLF
- `220-Welcome\r\n` weitere Zeilen folgen
- `220 FTP Server ready.\r\n` letzte Zeile

FTP Replies II

- Informationen in Replies Beispiel PWD
- `257 “/” is current directory.\r\n`
- Pfadangabe in Anführungszeichen
- Pfadangaben im UNIX-Format /....

Statuscodes

- RFC 959 Empfehlung
- **1yz** Aktion erfolgreich initziert
- **2yz** Aktion erfolgreich ausgeführt
- **3yz** Aktion erfolgreich aber weitere Aktionen notwendig
- **4yz** Kommando nicht ausgeführt, eventuell zeitlich begrenzt
- **5yz** Kommando dauerhaft nicht ausführbar
- **y** Gruppierung der Antwort
- **z** feingranulare Differenzierung

Kontroll-Verbindung

- Kommandos und Antworten

Server	Klient
220 Welcome	
	USER xxx
331 Password required	
	PASS xxx
230 Logged in	

Aushandeln einer “aktiven” Datenverbindung

Server Klient

TYPE A

200 Type set to A

PORT 192,168,0,10,16,51

200 Command ok

LIST

150 Opening ASCII Con...

Datenverbindung ist aktiv

226 Closing Daten Connection.

Datenverbindung

- ASCII
  - Jede Zeile ist mit CRLF abgeschlossen
  - so kann im plattformabhängigen Newline-Format gespeichert werden
- Binary
  - Daten werden so wie gelesen, gesendet
  - keine Konvertierung notwendig
- Ende der Übertragung durch Schließen des Sockets an der Datenquelle und Empfang einer Meldung auf dem Kontrollkanal

Hilfsfunktionen

```
char *strtok( char *strToken,
              const char *strDelimit );
```

- findet Token in Strings
- *strToken* String bzw. NULL
  - NULL weitersuchen im letzten angegebenen String
- *strDelimit* Trennzeichen zwischen Token

strtok

```
char *s="CWD /folder with spaces\r\n";
char *ptr;
ptr=strtok(s, " \r");//Leerzeichen und \r
```

- ptr enthält “CWD”

```
ptr=strtok(NULL, "\r"); //weitersuchen
```

- ptr enthält “/folder with spaces”

Socket API

```
TYPE socket(int af,int type,int protocol);
```

- TYPE
  - **SOCKET** unter Win32
  - **int** unter Unix
- Win32: gegen ws2_32.lib Bibliothek linken

Windows Socket API

```
int WSAStartup(WORD wVersionRequested,
               LPWSADATA lpWSAData );
```

- Winsock Bibliothek laden: ws2_32.dll
- MAKEWORD(Major,Minor)
  - z.B. MAKEWORD(1,1) -> 1.1
  - MAKEWORD(2,0) -> Winsock 2
  - MAKEWORD(2,2) -> MSDN
- im Fehlerfall kein WSAGetLastError() bzw. GetLastError()
- `int WSACleanup(void)`

Winsock-Versions

bind/connect/accept

- **int** bind(TYPE s,const struct sockaddr* name, int len);
- **int** connect(TYPE *s*, **const struct** sockaddr* *name*, **int** *namelen* **);**
- socktype accept(TYPE *s*, **struct sockaddr*** *addr*, **int*** *addrlen* **);**

struct sockaddr

```
struct sockaddr {
    unsigned short sa_family;
    char sa_data[14];
};
```

- Template für spezielle sockaddr-Strukturen
  - sockaddr_in, sockaddr_in6

```
struct sockaddr_in{
    short sin_family;
    unsigned short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};
```

struct in_addr

```
typedef struct in_addr {
      union {
        struct {
          u_char s_b1,s_b2,s_b3,s_b4;
        } S_un_b;
        struct {
          u_short s_w1,s_w2;
        } S_un_w;
        u_long S_addr;
      } S_un;
} in_addr;
```

Hilfsfunktionen

- `unsigned long htonl(unsigned long);`
- `unsigned short htons(unsigned short);`
  - Host-Ordnung zu Netzwerk-Ordnung

- `unsigned long ntohl(unsigned long);`
- `unsigned short ntohs(unsigned short);`
  - Netzwerk-Ordnung zu Host-Ordnung

Hilfsfunktionen II

- **unsigned long** inet_addr(
  const char* *cp*);
- liefert eine IP-Adresse als 4 Byte Zahl
- *cp* IP Adresse im X.X.X.X Format

```
addr.sin_addr.S_un.S_addr=
        inet_addr("127.0.0.1");
```

- s_addr -> S_un.S_addr
- addr.sin_addr.s_addr=inet_addr(...);

- ...s_addr Konstanten
  - INADDR_ANY jede Netzwerkinterface-Adresse → bind
  - INADDR_LOOPBACK → 127.0.0.1

```
char* inet_ntoa( struct in_addr in );
```

- liefert IP Adresse im X.X.X.X Format

getaddrinfo

```
int getaddrinfo(const char* nodename,
                const char* servname,
                const struct addrinfo* hints,
                struct addrinfo** res );
```

- liefert eine Liste von *addrinfo*-Strukturen
- *nodename* -> IP Adresse oder Name
- *servname* Protokolltyp (http, ftp,.. oder Port)
- *hints* gewünschte Verbindungstyen (Familie, Typ, Protokoll)

struct addrinfo

```
typedef struct addrinfo {
  int ai_flags;
  int ai_family;
  int ai_socktype;
  int ai_protocol;
  size_t ai_addrlen;
  char* ai_canonname;
  struct sockaddr* ai_addr;
  struct addrinfo* ai_next;
} addrinfo;
```

echoc.c

getaddrinfo.c

Hilfsfunktionen

- **int** getpeername**(TYPE** *s*, **struct sockaddr*** *name*, **int*** *namelen* **);**
  - liefert die verbundene Gegenstelle des sockets
- **int** getsockname**(TYPE** *s*, **struct sockaddr*** *name*, **int*** *namelen* **);**
  - liefert lokal verbundene Adresse des Sockets
- und viele mehr

Programm-Beispiele

- Datei kopieren
  - Win32
  - Linux/POSIX
  - Plattformübergreifend
  - Rechnerübergreifend

Datei Kopieren Win32 I

- **BOOL CopyFile(LPCTSTR** *lpExistingFileName*,
  LPCTSTR *lpNewFileName*,
  BOOL *bFailIfExists* **);**
- Netzwerkfähig \\<server>\<share>\name
- Pfad ist auf MAX_PATH Zeichen begrenzt
- 32k Unicode Zeichen mit \\?\UNC\<server>\<share>

Datei Kopieren Win32 II

```
hRead=CreateFile(Pfad,....OPEN_EXISTING);
hWrite=CreateFile(Pfad,...,CREATE_ALWAYS);
do
{
   ReadFile(hRead,buf,255,&read,NULL);
   WriteFile(hWrite,buf,read,&written);
}while(255==read);
CloseHandle(hRead);
CloseHandle(hWrite);
```

Mini-Shell

- Starten von Programmen aus stdin oder einer Datei
- Namen mit \n abgeschlossen

```
FILE *fd;
if(argc>1)
   fd=fopen(argv[1],”r”);
else
   fd=stdin;
while(fgets(buf,255,fd)!=NULL)
{
   if(buf[strlen(buf)-1]=‘\n’)
     buf[strlen(buf)-1]=‘\0’;
   CreateProcess(NULL,buf,.....);
   WaitForSingleObject(hProcess,...)
}
fclose(fd);
```

shell.c

Pipe

- ls | more in C nachbilden
- ls und more seien 2 separate Programme
- CreatePipe/CreateProcess
- pipe/fork/dup2/execle

```
int hpipe[2];
pipe(&hpipe[0]);
if(0==fork())
{
   dup2(hpipe[1],1); /*close(1);dup(hpipe[1]);*/
   close(hpipe[1]); close(hpipe[0]);
   execl("/bin/ls", "ls",0); ....
}
dup2(hpipe[0],0); /*close(0); dup(hpipe[0]);*/
close(hpipe[0]); close(hpipe[1]);
execl("/bin/more", "more",0); ...
```

talk-Programm

- abwechselndes Lesen und Schreiben
- Server /Client
- Lösung mit NamedPipes/MailSlots/Sockets

talks.c

talkc.c

Fragestunde

- 31.01.2004
- Zur Vorlesungszeit in HS1
- Themenspezifische Fragen