
Programmieren für das Internet

<http://www-pu.informatik.uni-tuebingen.de/pfi-0506>

Übungsblatt 1

Abgabe: bis 9:00 Uhr am 3.11.2005

1. [20 Punkte] Programmiere einen Server für das *Calculate-Protokoll*! Der Server soll TCP-Verbindungen auf Port 9000¹ entgegennehmen. Der Server liest Befehle zeilenweise ein. Alle Zeilen werden durch Newline („\n“) beendet. Zum Protokoll gehören die folgenden Befehle:
 - Mit „ADD□*int*□*int*“ fordert der Client den Server auf, die beiden gegebenen positiven Ganzzahlen zu addieren.
 - Mit „MUL□*int*□*int*“ fordert der Client den Server auf, die beiden gegebenen positiven Ganzzahlen zu multiplizieren.
 - Auf MUL- und ADD-Befehle antwortet der Server mit „RESULT□*int*“, wobei *int* das Ergebnis der angeforderten Berechnung ist. Der Server ist danach bereit weitere Befehle entgegenzunehmen.
 - Mit „QUIT“ beendet der Client die Verbindung zum Server.

Verwende zur Implementierung die Socket-Primitiva `socket()`, `bind()`, `listen()` und `accept()`. Eine nützliche Hilfsfunktion `readline()`, die Daten zeilenweise aus einer Socket liest, findest du auf der Homepage. Die gelesenen Zeilen lassen sich mit Hilfe von `sscanf()` leicht zerlegen, die Ergebnis-Zeile kann mit `sprintf()` erzeugt werden. Gehe bei der Implementierung davon aus, dass der Client wirklich nur ganzzahlige Integerzahlen sendet.

Hinweis 1: Mit `telnet <servername> <portnr>` kann man die Funktionalität des Servers testen.

Hinweis 2: Verwende eine Programmiersprache deiner Wahl! Es müssen jedoch alle oben genannten Socket-Primitiva im Code verwendet werden.

Lösung: Hier eine Lösung in ANSI C:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <strings.h>
```

¹Die Portnummer kann frei gewählt werden. Wir empfehlen allerdings Ports oberhalb von 1024 zu verwenden, da dafür keine Superuser-Rechte benötigt werden.

```

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/uio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>

#define BUFFERLEN 255
#define SERV_TCP_PORT 9000
#define UNKNOWN_COMMAND "UNKNOWN COMMAND\n"
int panic(char *where)
{
    fprintf(stderr, "error at %s '%s'\n", where, strerror(errno));
    exit(127);
}

/* liest zeilenweise aus dem Socketdeskriptor sd und schreibt die
   Zeile in das char-Array buf. buflen gibt die Laenge von buf an.
   Rueckgabewert ist die Anzahl der gelesenen Zeichen */
int readline(int sd, char *buf, int buflen)
{
    int n, rc;
    char c;

    for (n = 1; n < buflen; n++) {
        if ((rc = read(sd, &c, 1)) == 1) {
            *buf++ = c;
            if (c == '\n')
                break;
        }
        else if (rc == 0) {
            if (n == 1)
                return 0;
            else
                break;
        }
        else
            return -1;
    }
    *buf = 0;
    return n;
}

void write_safe(int fd, const void *buf, size_t nbytes)
{
    if (write(fd, buf, nbytes) < 0)
        panic("write()");
}

```

```

void process_requests(int fd, struct sockaddr_in *client_addr)
{
    char buf[BUFFERLEN];
    char answer[BUFFERLEN];
    int arg_1, arg_2;

    for(;;)
    {
        bzero((void *) buf, BUFFERLEN);
        bzero((void *) answer, BUFFERLEN);

        if (readline(fd, buf, BUFFERLEN) < 0)
            panic("readline()");

        if (sscanf(buf, "ADD %i %i", &arg_1, &arg_2) == 2)
        {
            sprintf(answer, "RESULT %i\n", arg_1 + arg_2);
            write_safe(fd, answer, strlen(answer));
        } else

        if (sscanf(buf, "MUL %i %i", &arg_1, &arg_2) == 2)
        {
            sprintf(answer, "RESULT %i\n", arg_1 * arg_2);
            write_safe(fd, answer, strlen(answer));
        } else

        if (strncmp("INFO", buf, 4) == 0)
        {
            sprintf(answer, "YOURIP %s\n", inet_ntoa(client_addr->sin_addr));
            write_safe(fd, answer, strlen(answer));
        } else

        if (strncmp("QUIT", buf, 4) == 0)
        {
            close(fd);
            break;
        }
        else write_safe(fd, UNKNOWN_COMMAND, strlen(UNKNOWN_COMMAND));
    }
}

int main(int argc, char argv[])
{
    int sock_fd, client_fd, addr_len;
    struct sockaddr_in client_addr, server_addr;

    if ((sock_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        panic("socket()");
}

```

```

bzero((void *) &server_addr, sizeof(struct sockaddr_in));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(SERV_TCP_PORT);
server_addr.sin_addr.s_addr = INADDR_ANY;

if (bind(sock_fd,
        (struct sockaddr *) &server_addr,
        sizeof(struct sockaddr_in)) < 0)
    panic("bind()");

if (listen(sock_fd, SOMAXCONN) < 0)
    panic("listen()");

for(;;)
{
    addr_len = sizeof(struct sockaddr_in);
    client_fd =
        accept(sock_fd,
              (struct sockaddr *) &client_addr,
              &addr_len);
    if (client_fd < 0)
        panic("accept()");

    process_requests(client_fd, &client_addr);
}
close(sock_fd); /* close listening socket */
return 0;
}

```

Lösung: Hier eine Lösung in OCAML:

```

(* Compilieren mit:
   ocamlc -o serverml unix.cma server.ml
   oder
   ocamlpt-o serverml unix.cmxa echoserver.ml
   Starten mit :
   ./serverml
*)

open Unix
open Printf
open String

let err_unknown_cmd = "ERROR unknown command\n"

exception WriteFailure

let readline sd =
    let c = " " in
        let rec loop res =

```

```

    if (read sd c 0 1) = 1 then
      if c = "\n"
      then res^c
      else loop (res^c)
    else raise End_of_file
  in
  loop ""

let write_safe sd s =
  let len = String.length s in
  if (write sd s 0 len) < len
  then raise WriteFailure

let split_at_space s =
  if (contains s ' ') then
    let space = index s ' ' in
    (sub s 0 space, sub s space (length s - space))
  else (s, "")

let rec ltrim aStr = match aStr with
  "" -> ""
| (_) ->
  if (aStr.[0] = ' ') then ltrim (sub aStr 1 (length aStr -1))
  else aStr

let calculate_args op =
  match (split_at_space (ltrim args)) with
  (s, t) -> string_of_int (op (int_of_string (ltrim s)) (int_of_string (ltrim t)))

let chop aStr =
  if ( (aStr.[length aStr -2] = '\r') && (aStr.[length aStr -1] = '\n') )
  then ( sub aStr 0 (length aStr -2) )
  else if (aStr.[length aStr -1] = '\n') then
    sub aStr 0 (length aStr -1)
  else aStr

let get_ip sock_addr =
  match sock_addr with ADDR_INET
  (inet_addr, port) -> string_of_inet_addr inet_addr
  | _ -> failwith "get_ip"

let main =
  let sd = socket PF_INET SOCK_STREAM 0 in
  bind sd (ADDR_INET(inet_addr_any, 9000));
  listen sd 5;
  let rec server_loop () =
    let (csd, client_addr) = accept sd in
    let rec client_loop () =
      match split_at_space(chop(readline csd)) with

```

```

("INFO", _) -> begin
  write_safe csd ("YOURIP "^(get_ip client_addr) ^ "\n");
  client_loop ()
end
  | ("QUIT", _) -> begin
    close csd;
    server_loop ()
  end
  | ("ADD", args) -> begin
    write_safe csd ((calculate args ( + )) ^ "\n");
    client_loop ()
  end
  | ("MUL", args) -> begin
    write_safe csd ((calculate args ( * )) ^ "\n");
    client_loop ()
  end
  | _ -> begin
    write_safe csd err_unknown_cmd;
    client_loop ()
  end
in
  client_loop ()
in
  handle_unix_error
  server_loop ()

```

2. [3 Punkte] Erweitere den Server aus Aufgabe 1 um ein Kommando „INFO\n“. INFO sendet die IP-Adresse des Clients im Format „YOURIP<ip-address>\n“ zurück.

Lösung: Siehe Lösung zu Aufgabe 1.

3. [3 Punkte] Wozu gibt es UDP? Hätte es genügt, die Benutzerprozesse einfache rohe IP-Pakete senden zu lassen? Begründe deine Antwort!

Lösung: IP-Adressen identifizieren einen Host. UDP fügt diese Adresse Port-Nummern hinzu, womit sich unterschiedliche Prozesse und Dienste innerhalb eines Hostes adressieren lassen. UDP ermöglicht die Kommunikation zwischen Prozessen.

4. [4 Punkte] Kann das Protokoll aus Aufgabe 1 auch unter Verwendung von UDP implementiert werden? Welche Schwierigkeiten gibt es und wie können sie gelöst werden? (Antwort in Stichpunkten)

Lösung: Das *Calculate-Protokoll* aus Aufgabe 1 kann auch unter Verwendung von UDP implementiert werden. Jedoch gibt es einige Problem zu lösen:

- Zuverlässigkeit der Verbindung gewährleistet werden

- Reihenfolge der Pakete muss hergestellt werden

Dazu könnte man die Pakete fortlaufend nummerieren, bei Empfang zwischenspeichern und sortieren. Der Empfang von Paketen sollte jeweils bestätigt werden. Wenn die Empfangsbestätigung nicht eintrifft, ist das Paket noch mal zu senden.

Hinweis: Löse die Aufgaben in einem Team von zwei bis drei Teilnehmern!
Sende deine Lösung per E-Mail an singh@informatik.uni-tuebingen.de.
Bitte füge deiner E-Mail eine Liste aller Teammitglieder (jeweils Vor- und Nachname, Matrikelnummer und E-Mail-Adresse) bei.