Dr. M. Grabert
Johannes Mayer

Systemnahe Software (SS 2001)

Abgabetermin: 18. Juni 2001

Beispiellösung

Die Lösung für eine der beiden folgenden Aufgaben müssen Sie schon bis zum 11. Juni 2001 abgeben. Welche Lösung Sie früher abgeben bleibt Ihnen überlassen.

9 Wir basteln mit „Rohren“! (10 Punkte)

Nehmen Sie zuerst Ihren persönlichen Vorlesungsbegleiter zur Hand und schlagen diesen auf den Seiten 67/68 auf. Dort finden Sie eine Erklärung der Bibliotheksbefehle `popen()` und `pclose()`. Sie würden nun, nachdem Sie die Beschreibung gelesen haben, gerne wissen, wie diese implementiert sind. Na das kriegen Sie doch raus! Sie programmieren `popen()` und `pclose()` einfach selbst! ;-) Übertreiben müssen wir es aber dann doch nicht, oder?! Also gut, nach einem `popen()` muss zunächst das zugehörige `pclose()` kommen, bevor `popen()` erneut aufgerufen wird. Das macht die Sache jetzt aber wirklich leichter!

Tipps: Mit `execl("/bin/sh", "sh", "-c", cmd, NULL);` können Sie das Kommando `cmd` von der Shell ausführen lassen (unter Berücksichtigung von PATH). Mit `fdopen()` kann aus einem File Descriptor ein File Pointer gewonnen werden. Den File Descriptor darf man dann aber nicht mehr schließen, sondern muss am Ende den File Pointer schließen!

Lösung:

```
/*
 * AI4 SS01 Blatt 6 (c) jm sai uni ulm 2001
 * Nicht reentrante Implementierung von popen und pclose,
 * d.h. popen-pclose-Bloecke muessen paarweise disjunkt sein.
 */
#include <stdio.h>
#include <unistd.h>
#include <wait.h>

int popen_pid = -1;    // PID des Prozesses, der das Kommando ausfuehrt
FILE *popen_fp;        // FP auf das Schreib- bzw. Leseende der Pipe
```

```
FILE *popen(const char *cmd, const char *mode) {
   int is_read;      // TRUE <=> Vater will von FP lesen
   int pid;          // PID des Sohnes, der das Kommando ausfuehrt
   FILE *fp;
   int pfd[2];

         /*
          * Eingabe ist fehlerhaft, wenn das Kommando fehlt
          * oder der Modus ungleich "r" und ungleich "w" ist.
          */
   if (cmd == NULL || (strcmp(mode, "r") && strcmp(mode, "w")))
      return NULL;

   is_read = (strcmp(mode, "r") == 0);   // lesender Modus ?

   if (pipe(pfd) < 0) {
      perror("pipe()");
      return NULL;
   }

   switch (pid = fork()) {
      case -1 :                 // Fehler beim forken
         perror("fork()");
         return NULL;
      case  0 :                 // Sohn-Prozess => Kommando ausfuehren
         if (is_read) {      // Vater will aus der Pipe lesen
            close(1);        // FD 1 auf das Schreibende ...
            dup(pfd[1]);     // ... der Pipe umlenken
            close(pfd[1]);   // restlichen Enden der ...
            close(pfd[0]);   // ... Pipe dicht machen
         }
         else {                 // Vater will in die Pipe schreiben
            close(0);        // FD 0 auf das Leseende ...
            dup(pfd[0]);     // ... der Pipe umlenken
            close(pfd[0]);   // restlichen Enden der ...
            close(pfd[1]);   // ... Pipe dicht machen
         }
            /* Kommando mit der sh-Shell ausfuehren */
         execl("/bin/sh", "sh", "-c", cmd, NULL);
         perror("execl()");  // Fehler bei exec()
         exit(1);
      default :                 // Vater-Prozess => Programmausf. fortsetzen
         if (is_read) {    // Vater will aus der Pipe lesen
               /* Schreibende dicht machen ... */
            close(pfd[1]);
```

```
            /* ... und FP fuer Leseende besorgen */
         fp = fdopen(pfd[0], "r");
      }
      else {            // Vater will in die Pipe schreiben
            /* Leseende dicht machen ... */
         close(pfd[0]);
            /* ... und FP fuer Schreibende holen */
         fp = fdopen(pfd[1], "w");
      }
         /*
          * popen/pclose sind nicht reentrant, weil
          * ein zweiter popen Aufruf die folgenden
          * globalen Variablen ueberschreiben wuerde!
          */
      popen_fp = fp;     // FP und Sohn-PID global ...
      popen_pid = pid;   // ... machen fuer pclose
      return fp;
   }
}

int pclose(FILE *fp) {
   int status;      // Status des Sohn-Prozesses von waitpid

   if (popen_pid < 0 || fp != popen_fp)     // kein zugeh. popen-Aufruf?
      return -1;

   if (fclose(popen_fp) < 0) {              // Pipe-Ende schliessen
      fprintf(stderr, "could not close pipe\n");
      return -1;
   }

   if (waitpid(popen_pid, &status, 0) < 0){ // auf den Sohn-Prozess warten
      perror("waitpid()");
      return -1;
   }

   popen_pid = -1;                 // es gibt jetzt kein zugeh. popen mehr

   return status;
}

   /*
    * kleines Testprogramm:
    * Liest von der Standardeingabe Zeilen bis EOF und
    * schreibt diese in die Pipe, welche mit popen erzeugt
    * wurde. Von sort wird die Eingabe dann sortiert auf
```

```
 * die Standardausgabe geschrieben.
 */
int main() {
   char buf[128];
   FILE *fp;

   fp = popen("sort", "w");                   // Pipe zu "sort" zu Schreiben

   while (fgets(buf, sizeof(buf), stdin))     // Zeilen von STDIN => fp
      fprintf(fp, "%s", buf);

   pclose(fp);                                // Pipe zumachen

   return 0;
}
```

10 Wer wird Millionär? (10 Punkte)

Sie haben doch sicherlich in letzter Zeit 'mal in die „Glotze" geschaut? Dann haben Sie bestimmt auch auf die Sendung „Wer wir Millionär?" gezappt? Na das wäre doch die Idee, dieses Spiel zu implementieren. Damit Sie nicht ganz im Regen stehen, können sie das *ausführbare Programm* der Beispiellösung auch schon vorab testen. Noch nicht zufrieden? Klar, eine *Datenbank* mit Fragen bekommen Sie natürlich auch. Jetzt aber ans Werk!

Spielregeln: Der Kandidat bekommt vom Showmaster nacheinander Fragen gestellt bis er eine Frage falsch beantwortet oder Millionär ist. (Mit Jokern befassen wir uns erst gar nicht; das ist doch Kinderkram!) Zu jeder Frage werden dem Kandidaten vier Antworten zur Wahl gestellt, von denen er eine auswählen kann. Wählt er keine aus und beendet das Spiel, so hat er das bisher erspielte Geld gewonnen. Wählt er die falsche Antwort aus, so hat er verloren, das Spiel ist beendet und er erhält 32000 DM bzw. 1000 DM, falls er mindestens so viel zuvor erspielt hatte, und andernfalls geht er leer aus, wenn er zuvor weniger als 1000 DM erspielt hatte. Tippt er die richtige Antwort, so kommt er in die nächste Gewinnstufe – außer er hat schon eine Million erspielt; dann ist das Spiel zu Ende und er hat die Million gewonnen – und bekommt die nächste Frage gestellt. Der Kandidat beginnt zunächst mit 0 DM. Danach kann er in den folgenden Schritten bis zu einer Million „klettern": 100 DM, 200 DM, 300 DM, 500 DM, 1000 DM, 2000 DM, 4000 DM, 8000 DM, 16000 DM, 32000 DM, 64000 DM, 125000 DM, 250000 DM, 500000 DM und 1000000 DM.

Aufbau der Datenbank: Die Datenbank ist ein Datei, die für jede Frage genau sechs Zeilen enthält. Davon enthält jeweils die erste die Frage, die nächsten vier enthalten die Antworten und in der letzten steht die Nummer der richtigen Antwort (zwischen 1 und 4).

Aufgabenstellung: Schreiben Sie eine Client-Server-Anwendung, wobei der Client aus dem Server durch ein `fork()` entsteht. Beide kommunizieren miteinander über zwei Pipes miteinander (in jede Richtung eine Pipe).

Der Server (also der Vater-Prozess) liest zunächst die Datenbank mit den Fragen komplett ein. Danach wählt er zufällig eine dieser Fragen aus und vertauscht die Antworten dieser Frage auch zufällig (und vergisst dabei nicht, die Nummer der Lösung anzupassen). Dann schickt er

die Frage mit den Antworten (ohne die Lösung) an den Clienten zusammen mit dem Geldwert (d. h. auf wieviel Geld er durch die richtige Beantwortung dieser Frage kommen kann) dieser Frage. Dann bekommt er vom Clienten den Tipp und antwortet ihm nach der Auswertung des Tipps. In der Antwort teilt er dem Clienten mit, ob das Spiel zu Ende ist oder weiter geht. Ist es zu Ende, dann erfährt der Client durch diese Antwort auch den gewonnenen Geldbetrag und im Falle einer falschen Antwort die richtige Lösung. Andernfalls erfährt er den aktuell erspielten Geldbetrag. Geht das Spiel weiter, so stellt der Server nun die nächste Frage. Der Server achtet natürlich darauf, dass bei einem Spiel keine Frage doppelt gestellt wird!
Der Client liest zunächst vom Server die Frage mit Antworten und den Geldwert der Frage. Nachdem er diese ausgegeben hat, fordert er den Spieler auf, eine Auswahl zu treffen. Diese Auswahl ist entweder eine Antwort oder Spielende. Diese Wahl teil er dem Server mit und erhählt von diesem die Antwort (siehe oben). Daraufhin beendet er sich oder liest vom Server die nächste Frage, je nach dem was in der Antwort des Servers stand. Natürlich macht er für den Spieler entsprechende Ausgaben, damit dieser mitkriegt wie ihm gerade geschieht.

Tipps: Die *Fragen-Datenbank* muss sich im selben Katalog befinden wie das *Demo-Programm*. Werfen Sie doch 'mal einen Blick auf die Beispiel-Programme. Weitere Fragen gibt es unter *http://www.rtl.de/*. Sie können die Fragen-Datenbank nach Lust und Laune erweitern! :-))

Lösung:

```
/*
 * AI4 SS01 Blatt 6 (c) jm sai uni ulm 2001
 * Implementierung des Spiels "Wer wird Millionaer?" als
 * Client-Server-Anwendung. Client und Server kommunizieren
 * miteinander ueber je eine Pipe in beide Richtungen.
 * Der Client ist der Ratende. Der Server stellt die Fragen und
 * kennt die Loesung zu den Fragen.
 */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <wait.h>
#include <string.h>

#define MAX_FRAGE    200   /* max. Laenge einer Frage */
#define MAX_ANTWORT   50   /* max. Laenge einer Antwort */
#define ANZ_ANTWORT    4   /* Anzahl der Antworten pro Frage */

   /*
    * letztes Zeichen vom String s entfernen; dient zum
    * Entfernen eines Newlines am Ende eines Strings
    */
#define CHOMP(s)   (s)[strlen(s)-1] = 0;
```

```
   /*
    * Struktur zur Uebermittlung einer Aufgabe, d.h. Frage mit
    * den Antworten (aber ohne Loesung!); entspricht dem ersten
    * Teil einer Datensatz-Struktur (siehe bei server)!
    */
typedef struct {
   char frage[MAX_FRAGE];
   char antwort[ANZ_ANTWORT][MAX_ANTWORT];
} Aufgabe;

   /*
    * Struktur zur Uebermittlung einer Antwort auf eine Loesung.
    * antwort:  OK (=> weiterraten) oder ENDE (=> Schluss)
    * geld:     aktueller Stand (bei OK) bzw. Gewinn (bei ENDE)
    * loesung:  richtige Loesung (im Falle von ENDE durch falsche Antw.)
    */
typedef struct {
   int antwort;
   int geld;
   int loesung;
} Antwort;

#define OK   1   /* s.o.; richtige Antwort => weiter machen */
#define ENDE 2   /* s.o.; Abbruch wg. falscher Antwort oder Spielende */

int pid;         /* PID des Client (= Sohn-Prozess beim Forken) */

   /*
    * Ausgabe des Strings s mit einem abschliessenden Newline.
    * Der String wir (falls moeglich) so umgebrochen, dass
    * in jeder Zeile max. maxcol Zeichen stehen. Der Umbruch
    * erfolgt bei Leerzeichen. Woerter (d.h. Folgen von Nicht-
    * Leerzeichen), die laenger als maxcol sind, werden in
    * jeweils einer eigenen Zeile mit ihrer vollen Laenge ausgegeben.
    */
void printcol(char *s, int maxcol) {
   char *sdup = strdup(s);      // Duplikat erzeugen (wg. strtok)
   char *wort;                  // Zeiger auf ein Wort im String
   int len = 0;                 // Anzahl der Zeichen in akt. Zeile

   wort = strtok(sdup, " ");
   while (wort) {
      if (len == 0) {           // Zeile ist noch leer
         printf("%s", wort);    // => Wort auf jeden Fall
         len += strlen(wort);   //    ... ausgeben
      }
```

```
      else if (len + 1 + strlen(wort) < maxcol) { // Wort passt hin
         printf(" %s", wort);                     // (mit Leerzeichen)
         len += strlen(wort) + 1;                 // => ausgeben
      }
      else {              // Wort passt nicht in akt. Zeile
         puts("");        // => neue Zeile beginnen und
         len = 0;         //    ... das akt. Wort in diese
         continue;        //    ... Zeile schreiben
      }
      wort = strtok(NULL, " ");
   }

   puts("");         // Ausgabe mit Newline abschliessen

   free(sdup);         // Speicher von Duplikat freigeben
}

   /*
    * CLIENT
    * Bekommt eine Frage vom Server, schickt diesem dann die
    * vermutete Loesung und erhaelt dann vom Server eine
    * Anwort. Entsprechend der Antwort terminiert er oder
    * macht mit der naechsten Frage weiter.
    */
void client(int readfd, int writefd) {
   int tipp;         // eingegebener Tipp 0, 1 .. ANZ_ANTWORT
   int wert;         // "wert DM - Frage" momentan
   Aufgabe auf;      // auf = vom Server bekommene Aufgabe
   Antwort ant;      // ant = Antwort vom Server
   char buf[80];
   int i;

   ant.geld = 0;      // ant.geld = akt. Stand (= bish. Gewinn)
   while (1) {
      read(readfd, &auf, sizeof(auf));     // Aufgabe lesen
      read(readfd, &wert, sizeof(wert));   // wert der Frage lesen

      system("clear");         // Bildschirm loeschen

      printf("\nIhr aktueller Stand: %d DM\n\n", ant.geld);

         /*
          * Den Wert der Frage, die Frage selbst und
          * die Antworten formatiert ausgeben.
          */
      for (i = 0; i < 70; i++) putchar('-');
```

```
printf("\n                           %d DM - Frage\n", wert);
for (i = 0; i < 70; i++) putchar('-');
puts(""); printcol(auf.frage, 70); puts("");
   // zwei Antworten pro Zeile ausgeben
for (i = 0; i < ANZ_ANTWORT; i += 2) {
   printf("%d) %-*.*s  ", i+1, 30, 30, auf.antwort[i]);
   if (i+1 < ANZ_ANTWORT)
      printf("%d) %-*.*s\n", i+2, 30, 30, auf.antwort[i+1]);
}
for (i = 0; i < 70; i++) putchar('-');

printf("\n\n0) Spiel beenden mit dem bisherigen Gewinn\n\n\n");

tipp = -1;
while (tipp < 0 || tipp > ANZ_ANTWORT) {   // Tipp erfragen
   printf("Ihr Tipp: ");
   fgets(buf, sizeof(buf), stdin);
   if (sscanf(buf, " %d ", &tipp) != 1)
      continue;
}

write(writefd, &tipp, sizeof(tipp));   // Tipp an Server

read(readfd, &ant, sizeof(ant));       // Antwort vom Server

if (tipp == 0) {                  // Spielende durch Benutzer
   system("clear");
   if (ant.geld > 0)
      printf("\nGratulation!\n\nSie haben %d DM gewonnen!\n\n", ant.geld);
   else
      printf("\nSchade!\n\nSie haben leider nichts gewonnen!\n\n");
   break;
}
else if (ant.antwort == OK)       // richtige Antwort => weiter
   printf("\n\nRichtig!");
else if (ant.geld == 1000000) {   // Millionaer => Spiel beenden
   system("clear");
   printf("\nHerzlichen Glueckwunsch!\n\nSie sind Millionaer!\n\n");
   break;
}
else {                            // falsche Antwort => Spiel beenden
   system("clear");
   printf("\nFalsch!\n\nDie richtige Loesung waere gewesen:\n\n");
   printf("    %d) %s\n\n", ant.loesung, auf.antwort[ant.loesung-1]);
   if (ant.geld > 0)
      printf("Ihnen bleiben noch %d DM.\n\n", ant.geld);
```

```
         else
            printf("Sie haben alles verloren!\n\n");
         break;
      }

      sleep(1);                          // kleine Pause zwischen den Fragen
   }
}

#define DB      "fragen.db"   /* Name der Fragen-Datenbank (= Datei) */
#define MAXDB   200           /* max. Anzahl von zu lesenden Datensaetzen */

   /*
    * Datensatz, der einen Eintrag der Fragen-Datenbank beherbergt
    */
typedef struct {
   char frage[MAX_FRAGE];
   char antwort[ANZ_ANTWORT][MAX_ANTWORT];
   int loesung;                             // antwort[loesung-1] ist Lsg.
} Datensatz;

   /*
    * Vertauscht die Antworten im uebergebenen Datensatz zufaellig.
    * (Entsprechend muss natuerlich auch die Loesung angepasst
    * werden.)
    */
void vertausche(Datensatz *d) {
   Datensatz tmp;              // Kopie des Datensatzes *d
   int quelle[ANZ_ANTWORT];    // Marken: kopierte Antworten von tmp
   int ziel[ANZ_ANTWORT];      // Marken: belegte Antworten von *d
   int n;
   int i, j;

   for (i = 0; i < ANZ_ANTWORT; i++)   // alle Antworten als nicht
      quelle[i] = ziel[i] = 0;         // kopiert bzw. belegt mark.

   memcpy(&tmp, d, sizeof(tmp));       // Duplikat *d wird tmp

   for (n = ANZ_ANTWORT; n > 0; n--) { // alle Antworten kopieren
      do {
         i = rand() % ANZ_ANTWORT;     // noch nicht kopierte
      } while (quelle[i]);             // Antwort suchen ...
      quelle[i] = 1;                   // und markieren!

      do {
         j = rand() % ANZ_ANTWORT;     // noch nicht belegte
```

```
      } while (ziel[j]);              // Antwort suchen ...
      ziel[j] = 1;                    // und markieren!

                                      // Antwort kopieren
      memcpy(d->antwort[j], tmp.antwort[i], MAX_ANTWORT);

      if (tmp.loesung == i+1)         // ist dies Antwort die Loesung?
         d->loesung = j+1;            // dann Loesung in *d aendern
   }
}

#define GEWINNE   16      /* Anzahl der moeglichen Gewinne */

   /*
    * SERVER
    * Liest zunaechst die komplette Datenbank (aber max. MAXDB
    * Datensaetze) ein. Schickt dann dem Client eine Frage, liest
    * dessen Tipp und gibt ihm dann die Antwort (=> weiter oder Ende).
    * War der Tipp korrekt und ist der Client noch nicht Millionaer,
    * so stellt er die naechste Frage.
    */
void server(int readfd, int writefd) {
   const int gewinn[GEWINNE] = {0, 100, 200, 300, 500,   // alle mgl. ..
      1000, 2000, 4000, 8000, 16000, 32000,              // ... Gewinne
      64000, 125000, 250000, 500000, 1000000};
   const int restgewinn[GEWINNE] = {0, 0, 0, 0, 0,   // noch verbleibender ..
      1000, 1000, 1000, 1000, 1000, 32000,           // ... Gewinn bei ...
      32000, 32000, 32000, 32000, 32000};            // ... einem Fehler
   Datensatz daten[MAXDB];         // alle eingelesenen Datensaetze
   int verwendet[MAXDB];           // wurde die Frage verwendet?
   int n;                          // Anzahl der Datensaetze
   int m;                          // akt. Index in gewinn
   int tipp;                       // Tipp des Clienten (0=Ende)
   Antwort ant;                    // Antwort auf den Tipp
   char buf[80];
   int i;
   FILE *fp;

      /*
       * alle Datensaetze (Fragen mit Antworten und Loesung)
       * aus der Datei DB in das Array daten[] einlesen
       */
   if (!(fp = fopen(DB, "r")))
      fprintf(stderr, "Server: could not open %s\n", DB),
      kill(pid, 9),                               // Client abschiessen
      exit(1);
```

```
for (n = 0; n < MAXDB; n++) {
      // Frage einlesen (bzw. EOF feststellen)
   if(!fgets(daten[n].frage, sizeof(daten[n].frage), fp))
      break;
   CHOMP(daten[n].frage);                     // Newline entfernen
      // ... alle Antworten einlesen ...
   for (i = 0; i < ANZ_ANTWORT; i++) {
      fgets(daten[n].antwort[i], sizeof(daten[n].antwort[i]), fp);
      CHOMP(daten[n].antwort[i]);             // Newline entfernen
   }
      // ... und zugehoerige Loesung holen
   fgets(buf, sizeof(buf), fp);
   sscanf(buf, " %d ", &daten[n].loesung);
   verwendet[n] = 0;             // gelesene Frage wurde nicht gestellt
}
fclose(fp);

   /*
    * Wenn nicht genuegend Fragen fuer ein Spiel
    * bis zu einer Million zur Verfuegung stehen,
    * soll ein Abbruch erfolgen.
    */
if (n < GEWINNE - 1)
   fprintf(stderr, "Server: database has not enough questions\n"),
   kill(pid, 9),
   exit(1);

srand(time(NULL));      // Zufallszahlengenerator initialisieren
m = 0;                  // bei gewinn[0] starten
while (1) {
   do {
      i = rand() % n;        // noch nicht gestellte Frage ...
   } while (verwendet[i]);   // ... auswaehlen ...
   verwendet[i] = 1;         // ... und als gestellt markieren

   vertausche(&daten[i]);    // Antworten zufaellig vertauschen

      /*
       * ersten Teil von daten[i] (ohne die Loesung!)
       * zum Client schicken; dieser Teil entspricht
       * genau einer Aufgabe-Struktur!
       */
   write(writefd, &daten[i], sizeof(Aufgabe));

      // "Wert" der Frage dem Clienten mitteilen,
      // d.h. "... DM - Frage" => ... ist der Wert
```

```
      write(writefd, &gewinn[m+1], sizeof(gewinn[m+1]));

      read(readfd, &tipp, sizeof(tipp));   // Tipp vom Client

      if (tipp == 0) {                     // Spielende gewaehlt
         ant.geld = gewinn[m];
         break;
      }
      else if (tipp == daten[i].loesung) { // richtige Loesung
         ant.geld = gewinn[++m];
         if (gewinn[m] == 1000000)         // Millionaer?
            break;                         // ... dann Ende
         ant.antwort = OK;                 // ... sonst weiter
         write(writefd, &ant, sizeof(ant));
      }
      else {                               // falsche Loesung
         ant.geld = restgewinn[m];         // nur noch ein Rest
         break;                            // ...vom Geld bleibt
      }
   }

   ant.antwort = ENDE;                     // dem Clienten mit der
   ant.loesung = daten[i].loesung;         // ... Antwort Spielende
   write(writefd, &ant, sizeof(ant));      // ... signalisieren
}

int main() {
   int cs_pfd[2];                      // Pipe: Client ----------> Server
   int sc_pfd[2];                      // Pipe: Client <---------- Server

   pipe(cs_pfd);                       // Pipe: Client schreibt, Server liest
   pipe(sc_pfd);                       // Pipe: Server schreibt, Client liest

   switch (pid = fork()) {
      case -1:                         // Fehler beim Forkeln
         perror("fork()");
         exit(1);
      case 0:                          // Sohn-Prozess => Client
         close(sc_pfd[1]);                  // nicht benoetigte
         close(cs_pfd[0]);                  // Enden dicht machen
         client(sc_pfd[0], cs_pfd[1]);
         exit(0);
      default:                        // Vater-Prozess => Server
         close(sc_pfd[0]);                  // nicht notwendige
         close(cs_pfd[1]);                  // Enden zu machen
         server(cs_pfd[0], sc_pfd[1]);
```

```
                                // brav noch auf den Sohn warten
        while (waitpid(pid, NULL, 0) != pid);
        exit(0);
    }

    return 0;
}
```