TI III WS 2013, Fr. 12-14

Lösung Übungsblatt 4

Christoph van Heteren-Frese (Matr.-Nr.: 4465677), Julien Stengel
Tutor: Ruhland, eingereicht am 15. November 2013

Aufgabe 3: Speicherzuteilung

Aus der manpage von `malloc`:

> *void *malloc(size_t size);*
> *...*
> *Malloc() allocates size bytes and returns a pointer to the allocated memory. The memory is not cleared.*

Somit versucht der Aufruf von `malloc(1000000000000)` genau 10^{12} Bytes, also 1 Terrabyte bereitzustellen. Dies gelingt nicht ein einziges Mal!

Der größtmögliche Speicherbereich, der mit `malloc()` angefordert werden kann, hängt sowohl vom physikalischen Speicher als auch dem verwendeten Betriebssystem ab. Theoretisch wir er vom verwendetet typ `size_t` limitiert, der selbst durch 4 Byte repräsentiert wird und somit also maximal den Wert $2^{32} - 1 = 4.294.967.295$ darstellen kann (ohne Vorzeichen). malloc(1000000000000) verursacht beim compilieren dementsprechend die Warnung

> *mal.c:5:3: Warnung: Große Ganzzahl implizit auf vorzeichenlosen Typen abgeschnitten [-Woverflow]*

Aufgabe 4

a)

```c
/* Erfolgreich compiliert mit:
 * gcc (Ubuntu/Linaro 4.7.3-1ubuntu1) 4.7.3
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

int main(int argc, char **argv) {
    /* der Typ FILE wird in stdio.h definiert
     * Er enthaelt die noetigen Informationen, um Ein- und
     * Ausgabeoperationen durchzufuehren.
     */
```

```
FILE    *f1;
FILE    *f2;
char    buf[100];
int     i;

/* Wenn die Anzahl der Argumente nicht genau drei ist,
 * gib eine Fehlermeldung aus.
 */
if(argc != 3)
{
    printf("usage: u1_4a <inputfiles> <outputfile>!\n");
    exit(-1);
}

/* Oeffne die erste Textdatei (als erstes Argument uebergeben),
 * um aus ihr zu Lesen (modus rt)
 */
f1 = fopen(argv[1], "rt");

/* Falls das nicht geklappt hat, gib eine Fehlermeldung aus.
 */
if(f1 == NULL)
{
    printf("Error: Can't open '%s' for read!\n", argv[1]);
    exit(-1);
}

/* Oeffne die zweite Textdatei (als zweites Argument uebergeben),
 * um in ihr zu schreiben.
 */
f2 = fopen(argv[2], "at");

/* Falls das nicht geklappt hat, gib eine Fehlermeldung aus.
 */
if(f2 == NULL)
{
    printf("Error: can't open '%s' for write!\n", argv[2]);
    exit(-1);
}

/* Solange nicht das Ende der Datei erreicht wurde...
 */
while(!feof(f1))
{
  /* ...lese ein Zeile von byte/char_t aus dem filestream
   * in den Puffer, aber hoechstens 99 Zeichen.
   */
    fgets(buf, 100, f1);
```

```
        /* Fuer jedes Zeichen im Puffer...
         */
        for(i=0; i<strlen(buf); i++)
        {
          /* ...pruefe, ob es ein kleiner Buchstabe ist.
           */
            if(islower(buf[i]))
            {
              /* falls ja, wandle ihn in einen Grossbuchstben um.
               */
                buf[i] += 'A'-'a';
            }
         }
        /* Puffer in Ausgabedatei schreiben...
         */
        fputs(buf, f2);
    }

    /* ... und beide Dateien schliessen.
     */
    fclose(f1);
    fclose(f2);
    printf("Programm Ende!\n");

    return 0;
}
```

b)

Es gibt drei Modi eine Datei zu öffnen: `r`, `w`, `a`. Abhängig vom verwendeten Compiler und c-Standart (c89/c90, c99, c11) ist es notwendig ein `b` für binary bzw. ein `t` für text zu ergänzen.

Modus	Bedeutung	Bemerkung
r	read	Öffnet eine Datei um am Anfang beginnend aus ihr zu lesen. Existiert die Datei nicht, bricht die Funktion mit einem Fehler ab.
w	write	Erzeugt eine Datei um in ihr zu schreiben. Existiert die Datei bereits, wird der Inhalt zerstört.
a	append	Öffnet eine Datei und fügt an das Ende des Inhalts an.
r+	read *extended*	Öffnet eine Datei am Anfang beginnend um aus/in ihr zu lesen/schreiben. Existiert die Datei nicht, bricht die Funktion mit einem Fehler ab.
w+	write *extended*	Erzeugt eine Datei zum lesen/schreiben. Existiert die Datei bereits, wird der Inhalt zerstört.
a+	append *extended*	Öffnet eine Datei zum lesen/schreiben. Exisitiert die Datei bereits, wird an das Ende geschrieben. Andernfalls wird die Datei erzeugt.

Des Weiteren kann seit c11 bei den Modi `w` und `w+` ein `x` ergänzt werden. Dies hat zur Folge, das die Funktion abbricht, solle die Datei bereits existieren. Der Inhalt wird somit nicht zerstört.

Damit das Beispielprogramm an das Ende der Ausgabedatei schreibt, ist somit nur Zeile 41 zu ändern:

*f2 = fopen(argv[2], "**a**t");*

c)

Abhänhig vom System macht dies der Regel macht dies keinen Unterschied. In der manpage zu `fopen` heisst es:

> *The mode string can also include the letter 'b' either as a last character or as a character between the characters in any of the two- character strings described above. This is strictly for compatibility with C89 and has no effect; the 'b' is ignored on all POSIX conforming systems, including Linux. (Other systems may treat text files and binary files differently, and adding the 'b' may be a good idea if you do I/O to a binary file and expect that your program may be ported to non-UNIX environments.)*

Auf meinem System ist dementsprechend kein Unterschied feststellbar.

TI3 Übung 4

1.

a)

Prozess	0	1	2	3	4	5	6	7	8
P1	new	ready	running	running	ready	ready	ready	ready	ready
P2		new	ready	ready	Running	Running	ready	Ready	ready
P3			new	ready	ready	ready	Running	waiting	waiting
P4				new	ready	Ready	ready	ready	running

Prozess	9	10	11	12	13	14	15	16	17
P1	ready	running	running	waiting	waiting	waiting	waiting	waiting	ready
P2	Ready	ready	ready	running	Running	ready	ready	ready	ready
P3	Waiting	waiting	waiting	ready	ready	running	running	ready	ready
P4	Running	ready	ready	ready	ready	ready	ready	running	Running

Prozess	18	19	20	21	22	23	24	25
P1	running	running	ready	ready	ready	ready	running	running
P2	Ready	Ready	running	Running	waiting	waiting	waiting	waiting
P3	ready	Ready	Ready	Ready	running	terminated		
P4	ready	Ready	ready	ready	ready	running	terminated	

Prozess	26	27	28	29	30	31
P1	terminated					
P2	waiting	running	Running	running	Running	terminated
P3						
P4						

b)

Der Prozessor befindet sich zu den Zeitpunkten 7 und 26 im Zustand busy waiting

c)

Sollte ein Programm sich in einer Endlosschleife befinden oder auf einen Input warten, der aus irgendwelchen Gründen nicht mehr erfolgen wird, dann ist es sinnvoll einen Prozess abzubrechen.

Unter Unix lässt sich mit dem Befehl **kill -9 (PID)** ein Prozess abbrechen.

2.

Da der Speicher in variable Bereiche unterteilt ist, kann es vorkommen, dass ein eingehender Prozess der einen gewissen Anteil an Speicher benötigt ihn nicht bekommt. Das liegt daran, dass der freie Speicher nicht zusammenhängend ist. Daher kann der Prozess gar nicht erst gestartet werden, selbst wenn theoretisch genug freier Speicher vorhanden ist. Dadurch kann es zu freien Speicherplätzen kommen, die nicht groß genug sind, um einen neuen Prozess aufzunehmen und daher unbrauchbar werden.