
Technische Informatik III:

Betriebssysteme und Rechnernetze WS 2007/08

Musterlösung zum Übungsblatt Nr. 3

Aufgabe 1: Begriffe

4 Punkte

Beschreiben Sie jeden der folgenden Begriffe durch maximal zwei Sätze: Prioritäteninversion, Real-Time Scheduling, DMA, Pipes.

- **Prioritäteninversion:** Der Umstand, dass ein Prozess a mit höherer Priorität auf die Beendigung eines Prozesses b mit niedriger Priorität wartet, wenn z. B. b eine Sperre auf eine Ressource hält, die a benötigt.
- **Real-Time Scheduling:** Hier muss das Betriebssystem dafür sorgen, dass bestimmte Ereignisse durch einen Prozess in einer vorgegebener Zeit abgearbeitet werden.
- **DMA:** Direct Memory Access, Technik zur Prozessorentlastung mit der E/A-Geräte direkt mit dem Hauptspeicher kommunizieren können, ohne Umweg über den Prozessor.
- **Pipes:** Ein Puffer fester Größe, der zur Interprozesskommunikation benutzt wird.

Aufgabe 2: Scheduling

8 Punkte

In welcher Reihenfolge werden die folgende Prozesse bei den Algorithmen First-Come-First-Served, Round-Robin, Shortest Remaining Time und Highest Response Ratio Next abgearbeitet?

- **First-Come-First-Served:**

	0	5	10
A	X X X X X		
B		X X X	
C			X
D			X
E			X X

- **Round-Robin, $q = 1$:**

	0	5	10
A	X X X	X X X	X
B	X X X	X	
C	X		
D		X	
E		X	X

Queue:

→	A B A B C	A B D E A	E A
	A B C A	B D E A E	A
	A B	D E A	
		A	

- **Round-Robin, $q = 2$:**

	0	5	10
A	X X X	X	X
B	X X	X	
C		X	
D		X	
E		X	X

Queue:

→	A A B B A	A C B D E	E A
	B A A C	C B D E A	A
	C B	B D E A	
		D E A	
		A	

- Shortest Remaining Time:

	0	5	10
A	X		X X
B	X X X		
C		X	
D		X	
E		X X	

- Highest Response Ratio Next:

	0	5	10
A	X X X X X		
B		X X X	
C		X 2	
D		1	X 4
E			3 X X

1:

$$B: \frac{4+3}{3} = 2, \bar{3}$$

$$C: \frac{2+1}{1} = 3 \leftarrow$$

$$D: \frac{0+1}{1} = 1$$

2:

$$B: \frac{5+3}{3} = 2, \bar{6} \leftarrow$$

$$D: \frac{1+1}{1} = 2$$

$$E: \frac{0+2}{2} = 1$$

3:

$$D: \frac{4+1}{1} = 5 \leftarrow$$

$$E: \frac{3+2}{2} = 2, 5$$

4:

$$E: \frac{4+2}{2} = 3 \leftarrow$$

Aufgabe 3: Multiprozessor Scheduling

6 Punkte Punkte

Nennen Sie die drei mögliche Arten des Multiprozessor Scheduling. Beschreiben Sie in wenigen Sätzen, wie die Prozesse bei diesen unterschiedlichen Ansätzen den Prozessoren zugeteilt werden.

- *Global*: Hier existiert *eine* Prozesswarteschlange, die die Prozesse auf alle Prozessoren verteilt.
- *Master/Slave*: Hauptkernel läuft auf einem ausgezeichneten Prozessor (Master), der die Prozesse verwaltet und teilt den anderen (Slaves) und sich selbst die Prozesse zu.
- *Peer*: Jeder Prozessor hat seine eigene Prozesswarteschlange. Die Prozesse werden nach einem Verfahren, z. B. reihum, oder geringste Auslastung, den Prozessoren zugeteilt und werden dort abgearbeitet.

Aufgabe 4: I/O Abstraktionen

4 Punkte

Zeichenbasierter und blockbasierter Zugriff sind die zwei Grundabstraktionen, die ein Betriebssystem beim Zugriff auf Peripheriegeräte zur Verfügung stellt. Nennen Sie jeweils drei typische Geräte für jede der Zugriffsabstraktionen. Erläutern Sie anhand dieser Beispiele den grundsätzlichen Unterschied zwischen den beiden Zugriffsarten.

- Zeichenbasiert:
Die Daten kommen in einem Strom an. Diese kann man nicht adressieren.
Beispiele:
 - Terminal (Maus, Tastatur und Konsole)
 - Modem
 - Drucker
 - Plotter

Bei der Tastatur wird jedes Zeichen direkt beim Tippen an den Benutzerprozess übertragen.

- Blockbasiert:
Hier kann man die Daten adressieren.
Beispiele:
 - Framebuffer (3D-Grafik)
 - CD-ROM
 - Magnetband
 - Festplatte

Wenn man etwas von der Festplatte lesen will, gibt man einen Dateinamen mit Pfad an. Das Dateisystem weiß dann, welche Datenblöcke zu dieser Datei gehören und liefert die dann von der Festplatte.

Aufgabe 5: Fifty/Fifty

4 Punkte

Entscheiden Sie, ob folgenden Aussagen zutreffend oder nicht zutreffend sind, und begründen Sie Ihre Entscheidung:

- **Die Laufzeit des Scheduling-Algorithmus ist für die Echtzeitfähigkeit des Gesamtsystems von Bedeutung.**
Richtig, die Laufzeit ist auch wichtig, aber eigentlich ist die Art des Scheduling-Algorithmus wichtiger. FCFS zum Beispiel hat eine schnelle Laufzeit, aber erfüllt keine garantierten Echtzeitanforderungen.
- **In den Meta-Informationen zu einer Datei wird die Prozess ID des Prozesses, der zuletzt auf die Datei zugegriffen hat, hinterlegt.**
Falsch, unter UNIX wird nur die letzte Zugriffszeit gespeichert.

Aufgabe 6: IO

8 Punkte

Lösen Sie folgende Aufgaben mit den Mitteln, die Ihnen die `stdio.h` zur Verfügung stellt. Geben Sie hierbei die Ausgabe des Programms auf `stdout` aus und die Fehlermeldungen auf `stderr`.

- `cat`

Testlauf:

```
$ ls -l | ./cat
insgesamt 176
-rwxrwx--- 1 chris chris 12633 2007-11-25 21:22 03_Uebung_Loesung.tex
-rwxrwx--- 1 chris chris  2982 2007-11-24 13:11 aufgabe2.5.c
-rwxr-xr-x 1 chris chris 13982 2007-11-25 21:13 aufgabe3.7
-rw-r--r-- 1 chris chris  1341 2007-11-25 20:58 aufgabe3.7.tex
-rwxr-xr-x 1 chris chris  9537 2007-11-25 21:20 cat
-rwxrwx--- 1 chris chris  1432 2007-11-25 21:20 cat.c
-rwxrwx--- 1 chris chris  1509 2007-11-25 21:21 cat.tex
-rwxrwx--- 1 chris chris 50103 2007-10-22 14:03 Doxyfile
-rwxr-xr-x 1 chris chris  9665 2007-11-25 21:12 grep
-rwxrwx--- 1 chris chris  1563 2007-11-24 14:16 grep.c
-rwxrwx--- 1 chris chris  1623 2007-11-24 21:31 grep.tex
-rwxrwx--- 1 chris chris   638 2007-11-24 14:10 makefile
```

```

-rwxrwx--- 1 chris chris 4170 2007-11-25 21:13 mm.c
-rw-r--r-- 1 chris chris 1089 2007-11-24 14:00 mm.h
-rwx----- 1 chris chris 4074 2007-11-25 21:14 mm.tex
-rwxr-xr-x 1 chris chris 9443 2007-11-25 21:12 wc
-rwxrwx--- 1 chris chris 1733 2007-11-24 14:05 wc.c
-rwxrwx--- 1 chris chris 1790 2007-11-24 21:31 wc.tex

```

cat.c

```

1  /**
2  * @file cat.c
3  * TI III: Betriebssysteme und Rechnernetze
4  * WS 2007/08
5  * Übungsblatt Nr. 3
6  * Übungsaufgabe 6: IO
7  *
8  * cat
9  *
10 * @author Christian Grümme
11 * @see http://cst.mi.fu-berlin.de/teaching/WS0708/19513-V-TI-III/index.html
12 * @date 2007-11-18
13 */
14 #include <stdio.h>
15 #include <stdlib.h>
16 #define MAX 255 /* Blocklänge */
17
18 /** @fn int main(int argc, char *argv[])
19 * Gibt eine Datei oder stdin blockweise nach stdout aus.
20 *
21 * @param argc Anzahl der Argumente
22 * @param argv Erstes Argument wird als Datei eingelesen
23 * @return Status der Terminierung
24 */
25 int main(int argc, char *argv[]) {
26     FILE *f; /* file-descriptor für den Input */
27     char puffer[MAX]; /* Lesepuffer */
28
29     /* Überprüfe, ob ein ein Argument übergeben wurde */
30     if (argc >= 2)
31     {
32         /* Versuche 1. Argument als Datei zu öffnen */
33         f = fopen(argv[1], "r");
34
35         /* Überprüfe, ob das Öffnen der Datei erfolgreich war */
36         if (f == NULL)
37         {
38             /* Fehlerausgabe auf 'stderr' */
39             fprintf(stderr, "Cannot open file %s\n", argv[1]);
40
41             return EXIT_FAILURE;
42         }
43     }
44     else
45     {
46         /* Kein Argument, setze file-descriptor auf 'stdin' */
47         f = stdin;
48     }
49
50     /* Lese alle Zeichen vom file-descriptor bis EOF */
51     while (fgets(puffer, MAX, f) != NULL)
52     {
53         /* Ausgabe des gelesenen Zeichens nach 'stdout' */
54         fprintf(stdout, "%s", puffer);
55     }
56
57     fclose(f); /* Schließe file-descriptor */
58
59     return EXIT_SUCCESS;
60 }

```

• WC

Testlauf:

```

$ ./wc wc.c
85 266 1716

```

```

1  /**
2  * @file cat.c
3  *
4  * TI III: Betriebssysteme und Rechnernetze
5  * WS 2007/08
6  * Übungsblatt Nr. 3,
7  * Aufgabe 6: IO
8  *
9  * wc
10 *
11 * @author Christian Grümme
12 * @see http://cst.mi.fu-berlin.de/teaching/WS0708/19513-V-TI-III/index.html
13 * @date 2007-11-18
14 */
15 #include <stdio.h>
16 #include <stdlib.h>
17
18 /** @fn int main(int argc, char *argv[])
19 * wc
20 *
21 * @param argc Anzahl der Argumente
22 * @param argv Erstes Argument wird als Datei eingelesen
23 * @return Status der Terminierung
24 */
25 int main(int argc, char *argv[]) {
26     FILE *f;      /* file-descriptor für den Input */
27     char c;        /* Gelesenens Zeichen */
28     long count_bytes = 0; /* Gezählte Bytes */
29     long count_words = 0; /* Gezählte Wörter */
30     long count_lines = 0; /* Gezählte Zeilen */
31     short last_white_space = 1;
32
33     /* Überprüfe, ob ein ein Argument übergeben wurde */
34     if (argc >= 2)
35     {
36         /* Versuche 1. Argument als Datei zu öffnen */
37         f = fopen(argv[1], "r");
38
39         /* Überprüfe, ob das Öffnen der Datei erfolgreich war */
40         if(f == NULL)
41         {
42             /* Fehlerausgabe auf 'stderr' */
43             fprintf(stderr, "Cannot open file %s\n", argv[1]);
44
45             return EXIT_FAILURE;
46         }
47     }
48     else
49     {
50         /* Kein Argument, setze file-descriptor auf 'stdin' */
51         f = stdin;
52     }
53
54     /* Lese alle Zeichen vom file-descriptor bis EOF */
55     while( (c=fgetc(f)) !=EOF)
56     {
57         /* Zähle Byte */
58         ++count_bytes;
59
60         if(c == ' ' || c == '\t' || c == '\n' || c == '\r')
61         {
62             if(last_white_space == 0)
63             {
64                 ++count_words;
65                 last_white_space = 1;
66             }
67         }
68         else
69         {
70             last_white_space = 0;
71         }
72
73         if(c == '\n')
74         {
75             /* Zähle Zeilenende */
76             ++count_lines;
77         }
78     }
79
80     fclose(f);      /* Schließe file-descriptor */
81

```

```

82     printf("%ld %ld %ld\n", count_lines, count_words, count_bytes);
83
84     return EXIT_SUCCESS;
85 }

```

- grep

Testlauf:

```

$ ls -l | grep tex
-rwxrwx--- 1 chris chris 12660 2007-11-25 21:38 03_Uebung_Loesung.tex
-rw-r--r-- 1 chris chris 1341 2007-11-25 20:58 aufgabe3.7.tex
-rwxrwx--- 1 chris chris 1509 2007-11-25 21:21 cat.tex
-rwxrwx--- 1 chris chris 1622 2007-11-25 21:38 grep.tex
-rwx----- 1 chris chris 4074 2007-11-25 21:14 mm.tex
-rwxrwx--- 1 chris chris 1790 2007-11-25 21:35 wc.tex

```

grep.c

```

1  /**
2  * @file cat.c
3  *
4  * TI III: Betriebssysteme und Rechnernetze
5  * WS 2007/08
6  * Übungsblatt Nr. 3,
7  * Aufgabe 6: IO
8  *
9  * grep
10 *
11 * @author Christian Grümme
12 * @see http://cst.mi.fu-berlin.de/teaching/WS0708/19513-V-TI-III/index.html
13 * @date 2007-11-18
14 */
15 #include <stdio.h>
16 #include <stdlib.h>
17 #include <string.h>
18 #define MAX 255 /* Maximale Zeilenlänge */
19
20 /** @fn int main(int argc, char *argv[])
21 * grep
22 *
23 * @param argc Anzahl der Argumente
24 * @param argv Erstes Argument wird als Datei eingelesen
25 * @return Status der Terminierung
26 */
27 int main(int argc, char *argv[]) {
28     FILE *f; /* file-descriptor für den Input */
29     char line[MAX]; /* Gelesenes Zeichen */
30
31     /* Überprüfe, ob ein Argument übergeben wurde */
32     if(argc >= 3)
33     {
34         /* Versuche 1. Argument als Datei zu öffnen */
35         f = fopen(argv[2], "r");
36
37         /* Überprüfe, ob das Öffnen der Datei erfolgreich war */
38         if(f == NULL)
39         {
40             /* Fehlerausgabe auf 'stderr' */
41             fprintf(stderr, "Cannot open file %s\n", argv[1]);
42
43             return EXIT_FAILURE;
44         }
45     }
46     else if(argc == 2)
47     {
48         /* Kein Argument für eine Datei, setze file-descriptor auf 'stdin' */
49         f = stdin;
50     }
51     else
52     {
53         /* Gar kein Argument übergeben, beende */
54         printf("Usage: grep term file\n");
55
56         return EXIT_SUCCESS;
57     }
58
59     /* Lese Zeilenweise vom file-descriptor bis NULL */

```

```

60     while(fgets(line,MAX,f) != NULL)
61     {
62         /* Überprüfe, ob gesuchter Term in der Zeile enthalten ist */
63         if(strstr(line, argv[1]) != NULL)
64         {
65             printf("%s\n",line);
66         }
67     }
68
69     fclose(f);    /* Schließe file-descriptor */
70
71     return EXIT_SUCCESS;
72 }

```

Aufgabe 7: Speicherverwaltung

16 Punkte Punkte

Pointer und Arrays sind in C eng miteinander verbunden. Beschreiben Sie die exakte Funktionsweise der Operatoren `*`, `&` und `[]` aus folgendem Beispiel:

aufgabe3.7.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #define SIZE          10
4  #define ASCII_OFFSET  65
5
6  /* char* 'String'-Typ in C, [] -> Feld von Strings */
7  int main(int argc, char *argv[]) {
8      /* Deklaration von Zeiger auf ein Charakter ('String') */
9      char* ptr;
10     /* Deklaration von Zeiger auf ein Charakter ('String') */
11     char* tmp;
12     /* Deklaration eines Feldes mit 10 Character */
13     char array[SIZE];
14     int i;
15     for(i = 0; i < SIZE; i++)
16         /* Ansprechen des i.-ten Feldes von array*/
17         array[i] = i + ASCII_OFFSET;
18     /* Ansprechen des letzten Feldes von array; StringTermination */
19     array[SIZE - 1] = '\0';
20     printf("array is \"%s\".\n", array);
21     ptr = array;
22     printf("ptr points to \"%s\".\n", ptr);
23     printf("(array + 5) is \"%s\".\n", array + 5);
24     /* Speicheradresse des 5. Elementes im Feld array */
25     printf("&array[5] is \"%s\".\n", &array[5]);
26     /* Expliziter Cast + Multiplikation */
27     ptr = (char*) malloc(SIZE * sizeof(char));
28     if(ptr == NULL) {
29         printf("Can't allocate enough memory.\n");
30         return EXIT_FAILURE;
31     }
32     i = 0;
33     tmp = ptr;
34     while(array[i])
35         /* * ist Dereferenzierung, Schreibe i-ten Werte von array als Wert an die Speicher Adresse
36            tmp + i */
37         *(tmp++) = array[i++];
38     printf("ptr points to \"%s\".\n", ptr);
39     free(ptr);
40     return EXIT_SUCCESS;
41 }

```

Basierend auf sämtlichen vorangegangenen Beispielen implementieren Sie nun ein neues Modul *mm* (also bestehend aus *mm.h* und *mm.c*), das die Funktionen **void* my_malloc(int bytes)** und **void my_free(void* p)** zur Verfügung stellt. Mit diesen beiden Funktionen soll Ihr Modul analog zu den vordefinierten Funktionen **malloc()** und **free()** einen Speicherbereich verwalten. Legen Sie hierzu ein 10 KB großes *char*-Array an und verwenden Sie Listen für die Verwaltung der vergebenen Speicherzellen. Testen Sie Ihr Modul durch geeignete Aufrufe der beiden Funktionen.

Testlauf mit *aufgabe2.7.c*:

```

$ ./aufgabe3.7
[NULL]
[data: 3, next: 0x8049148] @ 0x8049150,

```

```
[data: 2, next: 0x8049140] @ 0x8049148,  
[data: 1, next: (nil)] @ 0x8049140,  
[NULL]  
[NULL]
```

mm.c

```
1  /**  
2  * @file mm.c  
3  *  
4  * TI III: Betriebssysteme und Rechnernetze  
5  * WS 2007/08  
6  * Übungsblatt Nr. 3,  
7  * Aufgabe 7: Speicherverwaltung  
8  *  
9  * Implementierung von my_malloc und my_free  
10 *  
11 * @author Christian Grümme  
12 * @see http://cst.mi.fu-berlin.de/teaching/WS0708/19513-V-TI-III/index.html  
13 * @date 2007-11-24  
14 */  
15 #include <stdio.h>  
16 #include <stdlib.h>  
17 #include <stddef.h> /* für ptrdiff_t */  
18 #define MAX 10240 /* 10KB */  
19  
20 char memory[MAX] = { 0 }; /** Der tatsächliche Speicher */  
21 long manage[MAX] = { 0 }; /** Verwaltung des Speichers, 0 unbelegt,  
22     -1 belegt, i>0 Anfang eines Blockes der Größe i */  
23 long cursor = 0; /** Aktueller Cursor für Next-Fit */  
24  
25 /** @fn int my_malloc_is_free(size_t size)  
26 * Diese Funktion überprüft, ob ab dem aktuellen Cursor noch  
27 * size Bytes unreserviert sind. Gibt 0 zurück, wenn das nicht  
28 * der Fall ist, sonst 1.  
29 *  
30 * @param size Größe des zu reservierenden Speichers in Byte  
31 * @return -1, wenn Block frei sonst index des ersten belegten Feldes  
32 */  
33 int my_malloc_is_free(int bytes)  
34 {  
35     long i = 0; /* Laufindex */  
36     long index; /* Index der Verwaltung */  
37  
38     /* Überprüfe, ob das Ende des Speichers erreicht würde */  
39     if((cursor + bytes) > MAX)  
40     {  
41         /* Passt nicht hinten in den Speicher rein */  
42         return (MAX - 1);  
43     }  
44  
45     /* Gehe aktuellen Block durch */  
46     while(i < bytes)  
47     {  
48         index = (cursor+i) % MAX; /* Aktueller Index */  
49         if(manage[index] != 0)  
50         {  
51             /* Block ist nicht frei */  
52             return index;  
53         }  
54  
55         ++i;  
56     }  
57  
58     /* Vom aktuellen Cursor sind size Bytes frei */  
59     return -1;  
60 }  
61  
62 /** @fn void* my_malloc(size_t size)  
63 * Diese Funktion reserviert size Byte im Array memory.  
64 * Diese Funktion überprüft mit my_malloc_is_free, ob noch genug  
65 * Platz da ist, wenn nicht wird solange der Cursor um eins verschoben  
66 * bis Platz da oder bis der Cursor wieder an der Stelle ist,  
67 * wo er war -> Fehler. Ansonsten schreibt sie in manage rein, welche  
68 * Speicherzellen von memory reserviert wurden.  
69 *  
70 * @param size Größe des zu reservierenden Speichers in Byte  
71 * @return pointer auf den reservierten Platz.  
72 */  
73 void* my_malloc(int bytes)  
74 {
```



```

75     long i = 1;          /* Laufindex */
76     long shift = 0;      /* Fortschritt beim suchen */
77     long old_cursor = cursor; /* Speichere aktuellen Cursor */
78
79     /* Überprüfe Maximum */
80     if(bytes > MAX)
81     {
82         /* Nicht genug Speicher*/
83         perror("Not enough memory!\n");
84
85         return NULL;
86     }
87
88     /* Suche nächsten freien Block der Größe bytes */
89     i = my_malloc_is_free(bytes);
90     while(i > 0)
91     {
92         /* Speichere Fortschritt */
93         if(i > cursor)
94         {
95             shift += (i - cursor);
96         }
97         else
98         {
99             shift += (MAX - cursor) + i;
100         }
101
102         if(shift > MAX)
103         {
104             /* Einmal alle Speicheradressen durch gegangen */
105             perror("Not enough memory!\n");
106
107             return NULL;
108         }
109         /* Gehe im Speicher weiter */
110         cursor = i;
111
112         i = my_malloc_is_free(bytes);
113     }
114
115     /* Belege Block der Größe bytes in der Verwaltung */
116     manage[cursor] = bytes;
117     for(i = 1; i < bytes; ++i)
118     {
119         manage[((cursor + i) % MAX)] = -1;
120     }
121
122     /* Speichere aktuellen Cursor */
123     old_cursor = cursor;
124
125     /* Verschiebe Cursor ans Ende des reservierten Blockes */
126     cursor = (cursor + bytes) % MAX;
127
128     /* Gebe Zeiger auf den Anfang des reservierten Blockes zurück */
129     return (void*) &memory[old_cursor];
130 }
131
132 /** @fn void my_free(void* pointer)
133  * Diese Funktion überprüft, ob der übergebene Zeiger gültig ist
134  * und gibt den Speicher in memory wieder frei.
135  *
136  * @param pointer Zeiger auf den freizugebenen Speicher
137  */
138 void my_free(void* p)
139 {
140     long i = 0;
141     long size;
142     long index;
143     ptrdiff_t diff; /* Datentyp für die Differenz von Zeigern */
144
145     /* Berechne Index vom übergebenen Zeiger im Array */
146     diff = p - (void*) &memory;
147
148     /* Überprüfe Index */
149     if(diff < 0 || diff >= MAX)
150     {
151         /* Übergebener Zeiger liegt nicht im Array */
152         fprintf(stderr, "my_free: invalid pointer:%p\n ", p);
153
154         return;
155     }
156

```

```

157     index = (long) diff;
158
159     /* Überprüfe Block */
160     if (manage[index] <= 0)
161     {
162         /* Übergebener Zeiger ist nicht Anfang eines Blockes */
163         fprintf(stderr, "my_free: invalid pointer:%p\n ", p);
164
165         return;
166     }
167
168     /* Gebe Speicher frei */
169     size = manage[index];
170     for (i = 0; i < size; ++i)
171     {
172         /* Setze Verwaltung zurück */
173         manage[((index+i)%MAX)] = 0;
174         /* Setze Speicher zurück */
175         memory[((index+i)%MAX)] = 0;
176     }
177 }

```

Speicherverwaltung - Implementierung mit Liste:

mm.h

```

1  /**
2  * @file mm.h
3  *
4  * TI III: Betriebssysteme und Rechnernetze
5  * WS 2007/08
6  * Übungsblatt Nr. 3,
7  * Aufgabe 7: Speicherverwaltung - Implementierung mit Listen
8  *
9  * Leicht abgewandelte Implementierung der Musterlösung aus dem letzten Jahr
10 * mit sehr vielen Kommentaren
11 *
12 * @author Barbara Haupt
13 * @see http://cst.mi.fu-berlin.de/teaching/WS0708/19513-V-TI-III/index.html
14 * @date 2007-12-13
15 */
16
17 //Präprozessor Direktive (dient dazu, dass die Header-Datei mm.h nur einmal
18 //eingebunden wird)
19 #ifndef _INC_MM
20 #define _INC_MM
21
22 #include <stdio.h>
23 #include <stdlib.h>
24 #include <assert.h>
25 #include <string.h>
26
27 //Listenknoten (einfach verkettete Liste)
28 typedef struct ns {
29     int addr;
30     int size;
31     struct ns *next;
32 } node;
33
34 void *my_malloc(int bytes);
35 void my_free(void* p);
36 void my_print();
37
38 #endif

```

mm.c

```

1  /**
2  * @file mm.c
3  *
4  * TI III: Betriebssysteme und Rechnernetze
5  * WS 2007/08
6  * Übungsblatt Nr. 3,
7  * Aufgabe 7: Speicherverwaltung - Implementierung mit Listen
8  *
9  * Leicht abgewandelte Implementierung der Musterlösung aus dem letzten Jahr
10 * mit sehr vielen Kommentaren. Die Speicherverwaltung verwendet einen
11 * First-Fit-Algorithmus. Die Speicherverwaltung mittels Liste lässt sich denken
12 * als Inhaltsverzeichnis (die Liste) zu einem Buch (dem Speicher).

```

```

13  * Das Inhaltsverzeichnis (die Liste) soll jeweils den Anfang eines
14  * Speicherblocks sowie die Größe des reservierten Speicherblocks speichern.
15  *
16  * @author Barbara Haupt
17  * @see http://cst.mi.fu-berlin.de/teaching/WS0708/19513-V-TI-III/index.html
18  * @date 2007-12-13
19  */
20
21 //Includes
22 #include "mm.h"
23
24 //Konstanten
25 #define MEM_SIZE 10240
26
27 // "Speicher"
28 char my_mem[MEM_SIZE];
29
30 //Listenkopf
31 node *head = NULL;
32
33 /**@fn void *my_malloc(int bytes)
34  *
35  * Gibt einen Pointer auf den freien Speicher zurück, falls der Speicher reserviert
36  * werden konnte. Falls nicht genügend freier Speicher vorhanden ist, gibt die Funktion
37  * NULL zurück.
38  */
39 void *my_malloc(int bytes) {
40
41     //Deklariere den einzufügenden Knoten
42     node *new_node;
43
44     //Hier wird sichergestellt, dass bytes eine natürliche Zahl ist.
45     //Man hätte auch einen nicht-vorzeichenbehafteten Datentyp verwenden können -
46     //aber das Interface ist nunmal so definiert.
47     assert(bytes>0);
48
49     //Hier wird überprüft, ob mehr Speicher reserviert werden soll, als tatsächlich
50     //handen ist. Falls dem so ist, gib NULL zurück.
51     if(bytes > MEM_SIZE)
52         return NULL;
53
54     //Bytes enthält also eine sinnvolle Zahl. Im folgenden lassen sich 2 Fälle
55     //unterscheiden: 1. Fall: Liste ist leer, 2. Fall: Die Liste ist nicht leer.
56     if(!head) {
57
58         //Reserviere Speicher für den neuen Knoten, der einmal der Listenkopf werden soll.
59         new_node = malloc(sizeof(node));
60
61         //Sicherstellen, dass Speicher reserviert werden konnte
62         assert(new_node);
63
64         //Speichere die Adressen intern nur als Offset zu my_mem, daher bekommt der
65         //Listenkopf die Adresse 0;
66         new_node->addr = 0;
67         new_node->size = bytes;
68         new_node->next = NULL;
69         head = new_node;
70
71         //Gib die tatsächliche Speicheradresse zurück,
72         //(damit der Benutzer etwas in den Speicher schreiben kann)
73         return my_mem + head->addr;
74     }
75     //2. Fall: Liste ist nicht leer
76     else{
77         //Plan: Durchlaufe die Liste. Wird ein ausreichend großer freier Platz gefunden,
78         //      reserviere den Platz, sonst gib NULL zurück.
79         //      1) berechne jeweils den freien Speicher zwischen zwei
80         //          Listenknoten. Falls tmp->next NULL ist, berechne den
81         //          freien Speicher zwischen tmp und dem Speicherende:
82         //          free_bytes = tmp->next->addr - (tmp->addr + tmp->size)
83         //          bzw.
84         //          free_bytes = MEM_SIZE - (tmp->addr + tmp->size)
85         //      2) Überprüfe, ob der freie Speicher ausreicht:
86         //          (free_bytes >= bytes)
87         //      3) Falls ja, füge neuen Listenknoten ein. Fertig.
88         //      4) Falls nicht, gehe zum nächsten Listenknoten und zu 1)
89         //          Falls es keinen nächsten Listenknoten gibt (tmp->next ist NULL),
90         //          gehe zu 5)
91         //      5) Es wurde kein ausreichend großer Speicherblock gefunden. Gib
92         //          NULL zurück.
93
94         node *tmp = head;

```

```

95     int next_addr;
96     int free_bytes;
97
98     while(1) {
99
100         //1)
101         (tmp->next) ? (next_addr = tmp->next->addr) : (next_addr = MEM_SIZE);
102         free_bytes = next_addr - (tmp->addr + tmp->size);
103         //2) + 3)
104         if (free_bytes >= bytes) {
105             node* new_node = malloc(sizeof(node));
106             assert(new_node);
107
108             new_node->addr = tmp->addr + tmp->size;
109             new_node->size = bytes;
110             new_node->next = tmp->next;
111             tmp->next      = new_node;
112
113             return my_mem + new_node->addr;
114         }
115         //4)
116         if(!tmp->next)
117             break;
118         else
119             tmp = tmp->next;
120     }
121 }
122 //5)
123 return NULL;
124 }
125
126
127 /**@fn void my_free(void *addr)
128 *
129 *   Gibt den Speicher an Adresse addr frei.
130 *   Stellt sicher, dass ein Knoten mit Adresse addr existiert.
131 */
132 void my_free(void* addr) {
133
134     //Plan: Durchlaufe die Liste, schaue sämtliche Adressen an
135     //       Entferne den Knoten, der die Adresse addr enthält aus der Liste.
136     node *tmp;
137     node *prev;
138     tmp = head;
139     while(tmp) {
140         if(tmp->addr == (char*)addr - my_mem) {
141             (tmp==head) ? (head = tmp->next) : (prev->next = tmp->next);
142             free(tmp);
143             return;
144         }
145         prev=tmp;
146         tmp=tmp->next;
147     }
148
149     //Hier sollte man nie ankommen
150     assert(0);
151 }
152
153 /**@fn void my_print()
154 *
155 *   "Druckt" ein Speicherabbild aus.
156 */
157 void my_print() {
158     node* tmp = head;
159     puts(" -- Speicherabbild --");
160     while(tmp) {
161
162         printf("%05d -> %05d [%5d bytes]\t[%c %c %c ... %c]\n", tmp->addr, tmp->addr+tmp->size-1,
163             tmp->size, my_mem[tmp->addr], my_mem[tmp->addr+1], my_mem[tmp->addr+2], my_mem[tmp->addr
164             +tmp->size-1]);
165
166         tmp=tmp->next;
167     }
168 }

```

mmtest.c

```

1  /**
2  * @file mmtest.c
3  *
4  * TI III: Betriebssysteme und Rechnernetze
5  * WS 2007/08

```

```

6  * Übungsblatt Nr. 3,
7  * Aufgabe 7: Speicherverwaltung - Implementierung mit Listen
8  *
9  * Leicht abgewandelte Implementierung der Musterlösung aus dem letzten Jahr
10 * mit sehr vielen Kommentaren
11 *
12 * @author Barbara Haupt
13 * @see http://cst.mi.fu-berlin.de/teaching/WS0708/19513-V-TI-III/index.html
14 * @date 2007-12-13
15 */
16
17 #include "mm.h"
18 /**@fn int main(int argc, char *argv[])
19  *
20  * Testfunktion
21  */
22 int main(int argc, char *argv[]) {
23
24     void* a;
25     void* b;
26     void* c;
27     void* d;
28     void* e;
29
30     puts("\nSpeicher wird reserviert...");
31     //Reserviere 1000 Byte
32     a = my_malloc(1000);
33     //Stelle sicher, dass der Speicher reserviert werden konnte
34     assert(a);
35     //Fülle den Speicherbereich mit dem Buchstaben A
36     memset(a, 65, 1000);
37
38     b = my_malloc(1000);
39     assert(b);
40     //Fülle den Speicherbereich mit dem Buchstaben B
41     memset(b, 66, 1000);
42
43     c = my_malloc(1000);
44     assert(c);
45     //Fülle den Speicherbereich mit dem Buchstaben C
46     memset(c, 67, 1000);
47
48     d = my_malloc(1000);
49     assert(d);
50     //Fülle den Speicherbereich mit dem Buchstaben D
51     memset(d, 68, 1000);
52     my_print();
53
54     puts("\nSpeicher wird freigegeben...");
55     my_free(b);
56     my_free(c);
57     my_print();
58
59     puts("\nSpeicher wird reserviert...");
60     b = my_malloc(1500);
61     assert(b);
62     //Fülle den Speicherbereich mit dem Buchstaben E
63     memset(b, 69, 1500);
64     my_print();
65
66     puts("\nVersuche mehr Speicher als vorhanden zu reservieren...");
67     e = my_malloc(10000);
68     assert(!e);
69     my_print();
70
71     puts("\nVersuche falschen Speicherbereich freizugeben...");
72     my_free(c);
73
74     getchar();
75
76     return 0;
77 }

```

Testlauf mit *mmtest.c*

```

nawab:/home/bude/haupt/TUTORIUM/TI3/03/Speicherverwaltung [86]% ls
Makefile mm2.c mm.c mm.h mm.o mmtest mmtest.c
nawab:/home/bude/haupt/TUTORIUM/TI3/03/Speicherverwaltung [87]% make all
rm -f mm.o mmtest
cc -c -o mm.o mm.c

```

```
cc -o mctest mctest.c mm.o
nawab:/home/bude/haupt/TUTORIUM/TI3/03/Speicherverwaltung [88]% ./mctest
```

```
Speicher wird reserviert...
```

```
-- Speicherabbild --
```

```
00000 -> 00999 [ 1000 bytes]    [A A A ... A]
01000 -> 01999 [ 1000 bytes]    [B B B ... B]
02000 -> 02999 [ 1000 bytes]    [C C C ... C]
03000 -> 03999 [ 1000 bytes]    [D D D ... D]
```

```
Speicher wird freigegeben...
```

```
-- Speicherabbild --
```

```
00000 -> 00999 [ 1000 bytes]    [A A A ... A]
03000 -> 03999 [ 1000 bytes]    [D D D ... D]
```

```
Speicher wird reserviert...
```

```
-- Speicherabbild --
```

```
00000 -> 00999 [ 1000 bytes]    [A A A ... A]
01000 -> 02499 [ 1500 bytes]    [E E E ... E]
03000 -> 03999 [ 1000 bytes]    [D D D ... D]
```

```
Versuche mehr Speicher als vorhanden zu reservieren...
```

```
-- Speicherabbild --
```

```
00000 -> 00999 [ 1000 bytes]    [A A A ... A]
01000 -> 02499 [ 1500 bytes]    [E E E ... E]
03000 -> 03999 [ 1000 bytes]    [D D D ... D]
```

```
Versuche falschen Speicherbereich freizugeben...
```

```
mctest: mm.c:114: my_free: Assertion '0' failed.
```

```
Abbruch
```

```
nawab:/home/bude/haupt/TUTORIUM/TI3/03/Speicherverwaltung [89]%
```