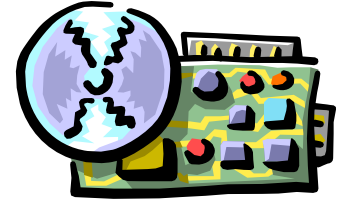




# TI III: Operating and Communication Systems



WS 2006/07  
Übungsblatt Nr. 3

Georg Wittenburg, M.Sc., AG Technische Informatik, Freie Universität Berlin

**Ausgabe am 16.11.2007 — Abgabe spätestens 30.11.2007, 10:00 Uhr**

Bitte bei der Abgabe beide Namen/Matr.Nr. der Mitglieder einer Gruppe, NUMMER DER ÜBUNG/TEILAUFGABE und DATUM auf den Lösungsblättern **nicht vergessen!** Darauf achten, dass die Lösungen beim richtigen Tutor/der richtigen Tutorin abgegeben werden.

Achten Sie bei Programmieraufgaben außerdem darauf, dass diese im Linuxpool kompilierbar sind.

**Zu spät abgegebene Lösungen werden nicht mehr berücksichtigt!**

## 1. Aufgabe: Begriffe (4 Punkte)

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

## 2. Aufgabe: 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?

Prozess	Ankunftszeit	Bearbeitungszeit
A	0	5
B	1	3
C	3	1
D	5	1
E	6	2

Nutzen Sie die in der Vorlesung verwendeten Zeitdiagramme zur Bearbeitung dieser Aufgabe.

## 3. Aufgabe: Multiprozessor Scheduling (6 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.

## 4. Aufgabe: 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.

## 5. Aufgabe: Fifty/Fifty (4 Punkte)

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

- Die Laufzeit des Scheduling-Algorithmuses ist für die Echtzeitfähigkeit des Gesamtsystems von Bedeutung.
- In den Meta-Informationen zu einer Datei wird die Prozess ID des Prozesses, der zuletzt auf die Datei zugegriffen hat, hinterlegt.

## 6. Aufgabe: 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`.

1. Implementieren Sie ein Programm „cat“, das einen Dateinamen als Parameter erhält und den Inhalt dieser Datei ausgibt.
2. Implementieren Sie ein Programm „wc“, das einen Dateinamen als Parameter erhält und die Anzahl der Zeilen, Worten und Bytes dieser Datei ausgibt.
3. Implementieren Sie ein Programm „grep“, das einen String und einen Dateinamen als Parameter erhält und alle Zeilen aus der Datei ausgibt, die diesen String enthalten.

Passen Sie nun Ihre Implementierung so an, dass für den Fall, dass kein Dateiname angegeben wurde, die Eingabe von `stdin` eingelesen wird. Testen Sie Ihre Programme durch geeignete Aufrufe. So soll beispielsweise der Befehl

```
$ ./cat file_name | ./grep test | ./wc
```

in einer UNIX-Shell die Anzahl von Zeilen, Worten und Bytes ausgeben, die in den Zeilen der Datei `file_name` enthalten sind, die auch den String „test“ enthalten.

Eine Referenz der C Standard Bibliothek finden Sie beispielsweise im Internet unter <http://www.utas.edu.au/infosys/info/documentation/C/CStdLib.html>.

## 7. Aufgabe: Speicherverwaltung (16 Punkte)

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

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE          10
#define ASCII_OFFSET  65

int main(int argc, char *argv[]) {
    char* ptr;
    char* tmp;
    char array[SIZE];
    int i;

    for(i = 0; i < SIZE; i++)
        array[i] = i + ASCII_OFFSET;
    array[SIZE - 1] = '\0';
    printf("array is \"%s\".\n", array);

    ptr = array;
    printf("ptr points to \"%s\".\n", ptr);

    printf("(array + 5) is \"%s\".\n", array + 5);
    printf("&array[5] is \"%s\".\n", &array[5]);

    ptr = (char*) malloc(SIZE * sizeof(char));
    if(ptr == NULL) {
        printf("Can't allocate enough memory.\n");
        return EXIT_FAILURE;
    }
    i = 0;
    tmp = ptr;
    while(array[i])
        *(tmp++) = array[i++];
    printf("ptr points to \"%s\".\n", ptr);
    free(ptr);

    return EXIT_SUCCESS;
}
```

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 10KB großes **char**-Array an und verwenden Sie Listen für die Verwaltung der vergebenen Speicherzellen. Testen Sie Ihr Modul durch geeignete Aufrufe der beiden Funktionen.