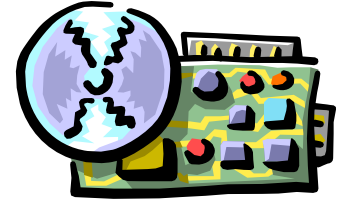




TI III: Operating and Communication Systems



WS 2006/07
Übungsblatt Nr. 2

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

Ausgabe am 2.11.2007 — Abgabe spätestens 16.11.2007, 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: Trace, Paging, Translation Lookaside Buffer, virtueller Speicher.

2. Aufgabe: Process Control Block (6 Punkte)

Erläutern sie in wenigen Sätzen die drei Bereiche in die man den Inhalt des Process Control Blocks einteilen kann.

3. Aufgabe: Auslagerung (6 Punkte)

Diskutieren Sie stichpunktartig die Vor- und Nachteile dreier Ersetzungsstrategien von Pages beim Swapping.

4. Aufgabe: Fifty/Fifty (8 Punkte)

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

- Prozesswechsel sind nach Möglichkeit zu vermeiden, da sie interaktive Benutzeranfragen verlangsamen.
- Die Anzahl von Kindern, die ein Prozess haben kann, ist vom Betriebssystem vorgegeben.
- Gemeinsamer Zugriff von mehreren Prozessen auf denselben Speicherbereich ist aus Sicherheitsgründen zu vermeiden.
- Ausgelagerte Speichersegmente müssen nur bei Schreibzugriffen in den Primärspeicher geladen werden.

5. Aufgabe: C Syntax (8 Punkte)

Kommentieren Sie das folgende Programm, ein Kommentar pro Zeile. Kommentieren Sie ebenfalls vor jeder Funktion deren Funktion.

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

typedef struct ns {
    int data;
    struct ns *next;
} node;

node *list_add(node **head, int i) {
    node *n = malloc(sizeof(node));
    n->data = i;
    n->next = *head;
    *head = n;
    return n;
}

void list_remove(node **head) {
    if(*head != NULL) {
        node *tmp = *head;
        *head = (*head)->next;
        free(tmp);
    }
}

void list_print(node *n) {
    while (n != NULL) {
        printf("[data: %d, next: %p] @ %p, ", n->data, n->next, n);
        n = n->next;
    }
    printf("[NULL]\n");
}

int main(int argc, char *argv[]) {
    node *n = NULL;
    list_print(n);
    list_add(&n, 1);
    list_add(&n, 2);
    list_add(&n, 3);
    list_print(n);
    list_remove(&n);
    list_remove(&n);
    list_remove(&n);
    list_print(n);
    return EXIT_SUCCESS;
}
```

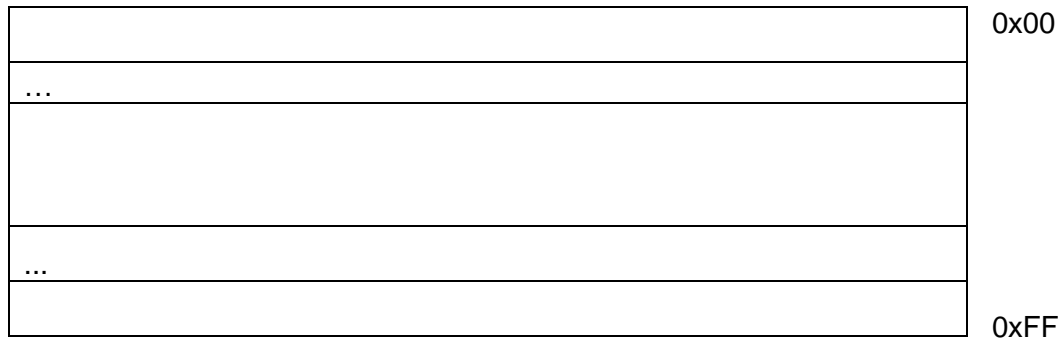
6. Aufgabe: Pointer (8 Punkte)

Implementieren Sie in C folgende Operationen auf einfach verketteten Listen:

- **node* list_get(node** head, int i)** – gibt das i-te Element der Liste zurück oder **NULL**, falls in der Liste zu wenig Elemente vorhanden sind.
- **void list_delete(node** head)** – löscht die gesamte Liste.
- **void list_add_at(node** head, node* n, int i)** – fügt einen neues Element an der gegebenen Stelle hinzu oder am Ende der Liste, falls die Liste nicht genug Elemente enthält. Das Element, das vorher an dieser Stelle stand (und alle folgenden) werden ein Platz nach hinten geschoben.
- **node* list_remove_at(node** head, int i)** – löscht das Element an der gegebenen Stelle aus der Liste und gibt es zurück. Falls die Liste zu kurz ist, bleibt sie unverändert und die Funktion gibt **NULL** zurück.

7. Aufgabe: Prozessabbild (4 Punkte)

Betrachten Sie bei dem von Ihnen in Aufgabe 6 erstellten Programm die Speicheradressen der globalen, lokalen und dynamisch allozierten Variablen, sowie der Funktionen. Tragen Sie Die Adressen in ein Diagramm wie das folgende ein, und weisen Sie den den Bereichen Bezeichnungen wie Code-, Heap- und Stackspeicher zu.



8. Aufgabe: Prozesserzeugung (6 Punkte)

Implementieren Sie in C ein Programm, dass mit dem System Aufruf **fork()** ein Kind-Prozesse erstellt. Dann soll der Kind-Prozess wiederum einen neuen Prozess erstellen und so weiter, bis eine Verschachtelungstiefe von zehn Prozessen erreicht ist. Danach sollen sich alle Prozesse wiederum in umgekehrter Reihenfolge beenden. Dabei soll sich jeder Prozess mit einer Nachricht „Prozess der Tiefe x wird beendet“ (x von 0 bis 9) verabschieden. Um dies zu erreichen, informieren Sie sich in den Manpages über die wait-Funktion.