Prof. Dr. Michael Jäger FB MNI

Lösungsvorschlag zur Klausur “Betriebssysteme” vom 26.3.2019

Aufgabe 1 (4+4+2 Punkte)

Punkte ____ *von* **10**

a) Die Ausführung einer Maschineninstruktion im Von-Neumann-Rechner lässt sich in verschiedene Phasen unterteilen. Beschreiben Sie kurz, was in der FETCH-Phase passiert.

Der Prozessor kopiert die Instruktion, deren Adresse im Programmzähler („PC“) steht, aus dem Hauptspeicher in das Befehlsregister.

Anmerkungen:

- Eine simple Antwort, die sich in fast trivialer Weise aus dem Begriff „FETCH-Phase“ ableiten lässt, lautet: „Der nächste Befehl wird geholt.“
  Dieser Antwort fehlen mehrere wichtige Aspekte: Welche Instruktion wird geladen? Wo steht die Instruktion? Wohin wird die Instruktion kopiert?
- Caching-Aspekt: Wie bei jedem Hauptspeicherzugriff, kann es auch beim FETCH dazu kommen, dass der Befehl nicht aus dem Hauptspeicher geladen werden muss, sondern aus einem Cache (Level 1/2/3) gelesen wird.
  Für volle Punktzahl nicht erforderlich, bei Nennung Sonderpunkt.

b) Wenn man bei einem herkömmlichen Rechner mit Multiprozessor-Architektur die Anzahl der Prozessorkerne verdoppelt, führt dies nicht unbedingt zu einer Verdoppelung der Rechenleistung. Warum nicht?

Typische Programme greifen häufig über den Systembus auf den Hauptspeicher zu. Die herkömmliche Bus-Technologie erlaubt keine gleichzeitige Bus-Nutzung durch mehrere Prozessorkerne, sondern setzt exklusiven Bus-Zugriff voraus. Die Bus-Reservierung und die Datenübertragung über den Bus sind vergleichsweise zeitaufwendige Operationen.

Je mehr Prozessorkerne ein Rechner hat, desto häufiger kommt es daher zu Wartesituationen bei Hauptspeicherzugriffen (Von-Neumann-Flaschenhals) und damit zu einer suboptimalen Prozessorauslastung.

Anmerkung: Es gibt natürlich weitere Aspekte, die aber eine geringere Rolle spielen. Auch bei vernünftig begründeten anderen Antworten gibt es Punkte.

c) Angenommen, die Maschineninstruktion `j 80` führt einen unbedingten Sprung zur logischen Adresse 80 aus. Was passiert bei der Ausführung dieser Maschineninstruktion innerhalb des Prozessors?

Kurze Antwort: Der Programmzähler wird auf 80 gesetzt.

Ausführlicher:

Die „jump“-Instruktion steht bei ihrer Ausführung im Befehlsregister des Prozessors. Sie besteht aus dem Opcode und dem Argumentwert 80. Bei der Ausführung wird das Sprungziel (80) aus dem Befehlsregister in das Programmzähler-Register (PC) kopiert.

Dies führt dazu, dass die Instruktion, die im Hauptspeicher an der Adresse 80 steht, als nächste in das Befehlsregister geladen und ausgeführt wird (sofern es sich nicht um einen unzulässigen Speicherzugriff handelt).

Aufgabe 2 (5+5 Punkte)

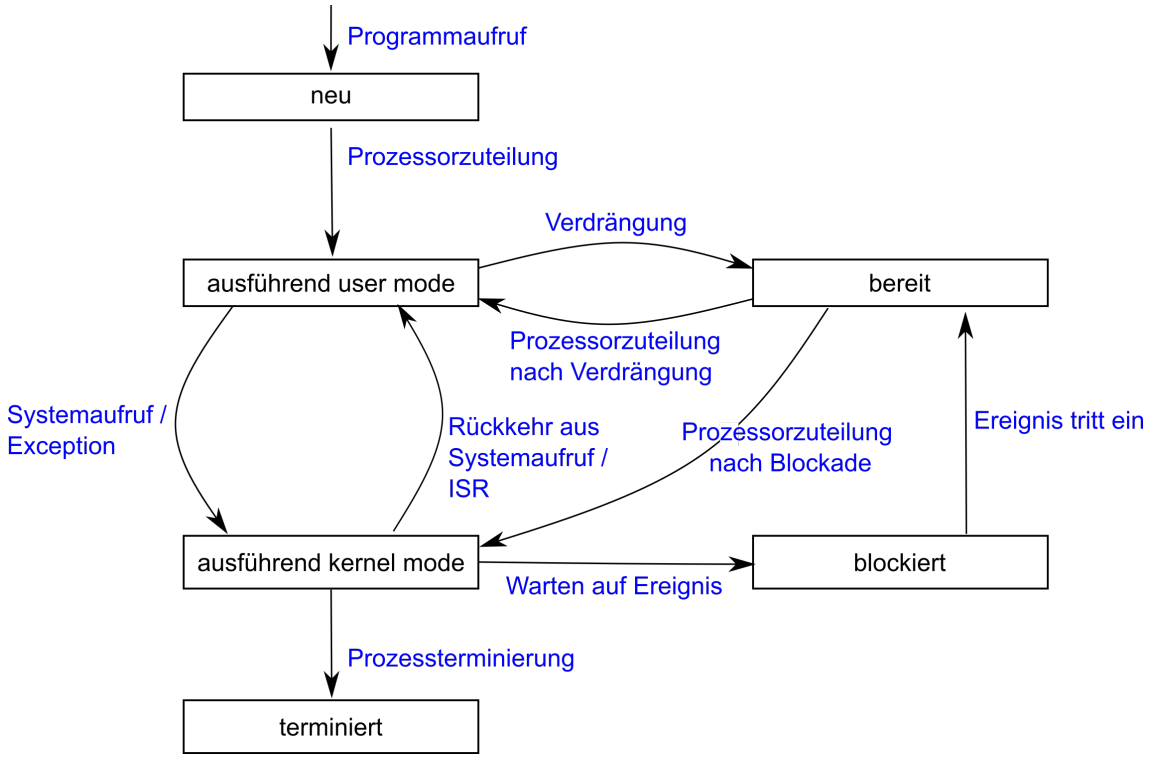
Punkte ____ *von* **10**

In einem Zustandmodell für Threads werden folgende Zustände unterschieden:

neu	
bereit	
ausführend im Anwendermodus	CPU führt Anwendungscode aus
ausführend im Kernelmodus	CPU führt Systemkern-Code aus
blockiert	Thread hat sich durch Systemaufruf selbst blockiert
terminiert	

Ein Thread wird im Kernelmodus nicht verdrängt.

a) Zeichnen Sie ein geeignetes Zustandsübergangsdiagramm. Aus dem Diagramm muss klar ersichtlich sein, welche Ereignisse zu den Zustandsänderungen führen.



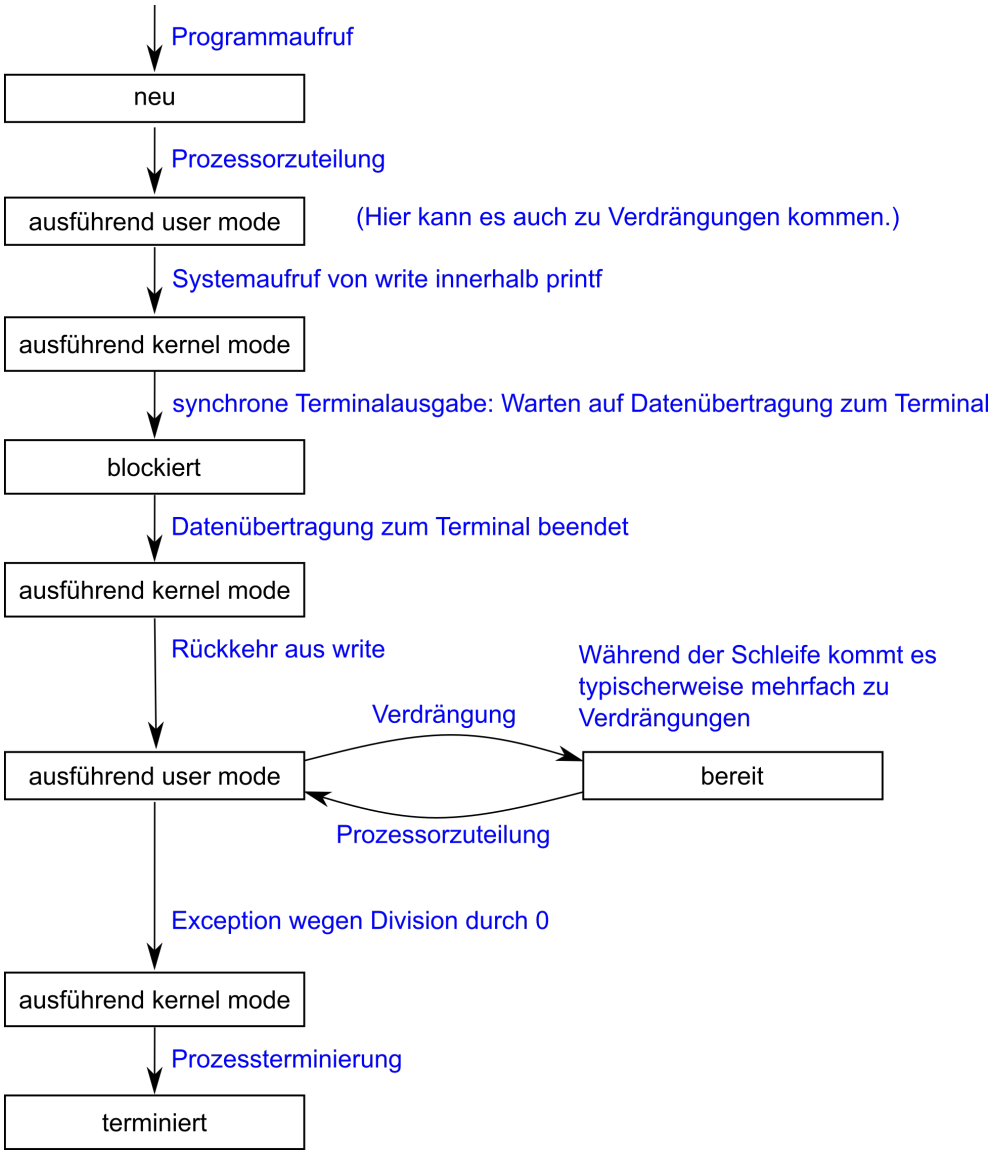
Erläuterungen:

- Bei der Ausführung im Anwendermodus können sowohl Systemaufrufe (Traps) als auch Prozessorausnahmen (Exception, z.B. Division durch 0, Seitenfehler) zur Aktivierung des Kernels führen. Wenn die Trap-/Exception-Behandlung nicht blockiert oder den Thread terminiert, gibt der Kernel die Kontrolle wieder an die Anwendung zurück.
- Im Modell wird nicht zwischen „bereit im Kernelmodus“ und „bereit im Anwendermodus“ unterschieden. Sowohl im Anwendungscode als auch im Systemkern kann es dazu kommen, dass ein Thread auf CPU-Zuteilung wartet.
- Eine Blockade erfolgt innerhalb eines Systemaufrufs. Nach dem Aufwecken muss der Thread zunächst wieder auf CPU-Zuteilung warten. Danach wird der Systemaufruf fortgesetzt.
- Im „user mode“ können Verdrängungen durch unterschiedliche Ereignisse ausgelöst werden, z.B. nach Ablauf des zugeteilten Quantums oder dadurch, dass ein anderer Thread mit höherer Priorität gestartet oder aufgeweckt wird.

b) Betrachten Sie folgendes Programm:

```
#include <stdio.h>
int main(){
  int i=0, j;
  printf("Berechnung startet\n");
  for (i=500000; i>=0; i--)
     j=10/i;
}
```

Bei der Ausführung wird das Programm eine Zeit lang rechnen und dann beim letzten Schleifendurchlauf wegen der Division durch 0 abbrechen. Beschreiben Sie anhand des obigen Zustandsmodells, welche Zustände der zugehörige Prozess bzw. Thread bei einem typischen Ablauf annehmen wird und wodurch die Zustandswechsel ausgelöst werden.



Wichtige Aspekte: Der *printf*-Aufruf führt zu einem blockierenden *write*-Systemaufruf. Während der for-Schleife muss mit Verdrängungen gerechnet werden. Die Division durch 0 führt zu einer CPU-Exception, zur Behandlung wird die Kontrolle an den Kernel übergeben.

Anmerkung: Berücksichtigt man im Modell, dass die Terminierung über einen Signalmechanismus erfolgt, ist auch die Anwendung selbst an der Ausnahmebehandlung beteiligt: Kernel erzeugt Signal, Signalbehandlung im Zustand „ausführend user mode“.

Aufgabe 3 (2+2+6 Punkte)

Punkte ____ von **10**

Betrachten Sie den UNIX-Shell-Befehl `cat < cat.input | wc -l`

(`wc -l` zählt die Zeilen in der Standardeingabe und gibt die Anzahl aus)

a) Welche Prozesse sind an der Ausführung beteiligt und wie ist deren Verwandschaftsverhältnis?

Beteiligt sind 3 Prozesse: Die Shell, `cat` und `wc`. `cat` und `wc` sind Subprozesse der Shell.

b) Unter welchen Umständen kann es dazu kommen, dass *wc* blockiert? Welche Systemaufrufe können dabei blockieren?

1) `wc` blockiert beim Lesen der Standardeingabe im read-Aufruf, wenn in der Pipe weniger Daten vorhanden sind, als gemäß read-Parameter gelesen werden sollen. Dies kann während der Ausführung mehrfach passieren, wenn `cat` nicht genügend schnell Daten in die Pipe schreibt. Der read-Aufruf kehrt zurück,
   - wenn genügen viele Daten verfügbar sind oder
   - sowohl `cat` als auch die Shell den Schreibdeskriptor der Pipe schließen.
2) Wenn `wc` beim Schreiben auf die Standardausgabe mit *write* Daten synchron zu einem Gerät (Terminal) überträgt, blockiert der write-Aufruf, bis die Datenübertragung abgeschlossen ist.

c) Geben Sie ein C-Programm an, das diesem Shell-Befehl entspricht. Das Programm soll zum Aufruf von `cat` und `wc` jeweils den Systemaufruf *execlp* verwenden. Vervollständigen Sie dazu das nachfolgende Programmskelett. Lassen Sie dabei jegliche Fehlerbehandlung und die `#include` -Anweisungen weg.

```
int main(){
    pid_t cat_pid, wc_pid;
    int pipefd[2];

    pipe(pipefd);
    switch (cat_pid=fork()){
    case 0:
       /* cat: Eingabe aus cat.input, Ausgabe in die Pipe */
       /* Eingabeumlenkung */
       {
         int fd=open("cat.input", O_RDONLY,0);
         dup2(fd,0);
         close(fd);
       }
       /* Ausgabeumlenkung */
       dup2(pipefd[1],1);
       close(pipefd[1]);
       /* Verklemmung vermeiden */
       close(pipefd[0]);
       execlp("cat", "cat", NULL);
    }

    switch (wc_pid=fork()){
    case 0:
       /* wc: Eingabe aus Pipe */
       /* Eingabeumlenkung */
       dup2(pipefd[0],0);
       close(pipefd[0]);
       /* Verklemmung vermeiden */
       close(pipefd[1]);
       execlp("wc", "wc", "-l", NULL);
    }
```

```
    /* Verklemmung mit den Kindern vermeiden */
    close(pipefd[0]);
    close(pipefd[1]);

    // Warten auf Ende der beiden Pipeline-Prozesse
    waitpid(cat_pid,NULL,0);
    waitpid(wc_pid,NULL,0);
}
```

Aufgabe 4 (4+3+3 Punkte)

Punkte ____ *von* **10**

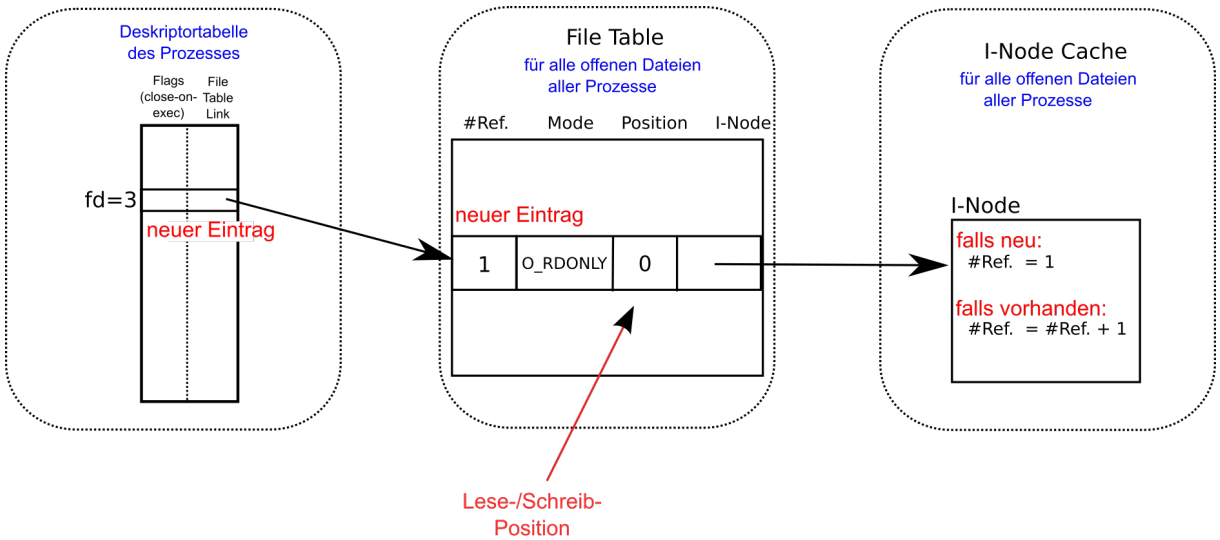
a) In einem C-Programm wird eine Datei wie folgt geöffnet

```
fd = open("/tmp/input.txt", O_RDONLY);
```

Beschreiben Sie in Stichworten, wie der Systemkern dabei die einzelnen Verzeichnisse im Dateipfad Schritt für Schritt bearbeitet und welche Prüfungen dabei erfolgen. Gehen Sie dabei davon aus, dass der Inhalt einer Verzeichnisdatei in einem Datencluster außerhalb des I-Node gespeichert ist.

1) I-Node Wurzelverzeichnis `/` lesen:
   - Dateityp („Directory“) prüfen
   - Lese- und Ausführungsberechtigung prüfen
   - Adresse Datencluster bestimmen
2) Datencluster von `/` :
   - Eintrag von „tmp“ suchen und I-Node-Nummer dazu bestimmen
3) Im I-Node von `/tmp`:
   - Dateityp („Directory“) prüfen
   - Lese- und Ausführungsberechtigung prüfen
   - Adresse Datencluster bestimmen
4) Datencluster von `/tmp`:
   - Eintrag von „input.txt“ suchen und I-Node-Nummer dazu bestimmen
5) I-Node von `/tmp/input.txt`:
   - Dateityp prüfen („reguläre Datei“)
   - Leseberechtigung prüfen

b) Falls im obigen Beispiel der `open`-Aufruf erfolgreich ist, werden im Systemkern durch das *open* einige interne Datenstrukturen (I-Node-Cache, File Table, Deskriptortabelle des Prozesses) modifiziert. Geben Sie mit Hilfe einer Skizze an, in welcher Weise.



Wichtige Aspekte:

- Die Einträge in File Table und Deskriptortabelle werden beim *open* neu angelegt.
- Wurde die Datei schon anderweitig geöffnet, wird der Referenzzähler des im Cache vorhandenen I-Node inkrementiert. Ansosnten wird der I-Node vom Datenträger in den I-Node-Cache kopiert und der Referenzzähler auf 1 gesetzt.

c) Bei einem klassischen UNIX-Dateisystem wird für jedes Datencluster einer Datei im Dateideskriptor ein Verweis benötigt. Bei Windows NTFS und modernen UNIX-Dateisystemen benutzt man stattdessen Extents. Was ist ein Extent?

Ein Extent ist eine Folge logisch zusammenhängender Dateicluster, die auch physikalisch zusammenhängend auf dem Datenträger angeordnet sind.

Wo ist der Vorteil von Extent-basierter Verwaltung der Datencluster?

In wenig fragmentierten Dateisystemen sind auch sehr große Dateien (sehr viele Cluster) nur auf wenige Extents verteilt. Die Extent-Deskriptoren einer Datei benötigen insgesamt so wenig Speicherplatz, dass sie direkt im Dateideskriptor gespeichert werden können.

Gegenüber einer Cluster-bezogenen Buchführung benötigt man keine (einfach und mehrfach) indirekten Verweise von Clusternummern auf Sektornummern. Die Bestimmung der zu einem Cluster gehörigen Adresse auf dem Datenträger kann dadurch deutlich schneller erfolgen.

Aufgabe 5 (2+2+2+2+2 Punkte)

Punkte ____ *von* **10**

Ein System verwendet virtuellen Speicher mit Paging. Für die Berechnung der 40 Bit großen realen Adressen wird eine zweistufige Seitentabelle benutzt. Eine virtuelle Adresse ist 32 Bit groß und von der Form | p_1 | p_2 | D | mit folgender Aufteilung:

- p_1: 8 Bit für die Seitentabelle der 1. Stufe
- p_2: 12 Bit für die Seitentabelle der 2. Stufe
- D: 12 Bit für die Distanz

Nehmen Sie an, die virtuelle Adresse `v=0x1001004` entspricht einer realen Adresse im 10. Rahmen des Hauptspeichers (Rahmen 9).

Zur Ermittlung der realen Adresse *r* werden 2 Seitentabellen benötigt.

a) Wo genau steht die Adresse der benötigten Seitentabelle der 2. Stufe?

Aufteilung der logischen Adresse: $p_1 = 0x01, p_2 = 0x001, D = 0x004$

Der Wert von p_1 entspricht dem 2. Eintrag der Seitentabelle der Stufe 1. Dort steht die Anfangsadresse der benötigten Seitentabelle der 2. Stufe.

b) Geben Sie die reale Adresse dieses Bytes hexadezimal an:

0x9004 (Rahmen 9, Distanz 0x004)

c) Geben Sie hexadezimal an, wie der zugehörige TLB-Eintrag aussieht (inklusive führender Nullen):

0x0 1001 $\rightarrow$ 0x000 0009

20 Bit für log. Adresse, 28 Bit für reale Adresse
(Die 12 Bit große Distanz `0x004` steht nicht im TLB).

d) Welche Speicherorte bieten sich für die Seitentabelle der 1. Stufe an? (Begründung)

Normalerweise stehen die Seitentabellen im Hauptspeicher. Da ein Eintrag für die Seitentabelle der 1. Stufe nur 8 Bit groß ist, hat die Seitentabelle nur 256 Einträge. Es wäre denkbar, die Tabelle aus Geschwindigkeitsgründen in der MMU als Registersatz mit 256 Registern zu realisieren. Bei einem TLB-“Miss“ müßte dann nur der Zugriff auf die Seitentabelle der 2. Stufe als (langsamer) Hauptspeicherzugriff erfolgen.

e) Was ist Demand-Paging?

Demand-Paging ist eine Seiten-Einlagerungsstrategie: Für eine Seite des logischen Adressraums eines Prozesses wird erst dann ein Rahmen im physikalischen Hauptspeicher reserviert, wenn der Prozess zum ersten Mal darauf zugreift, d.h. wenn es einen Seitenfehler gibt.