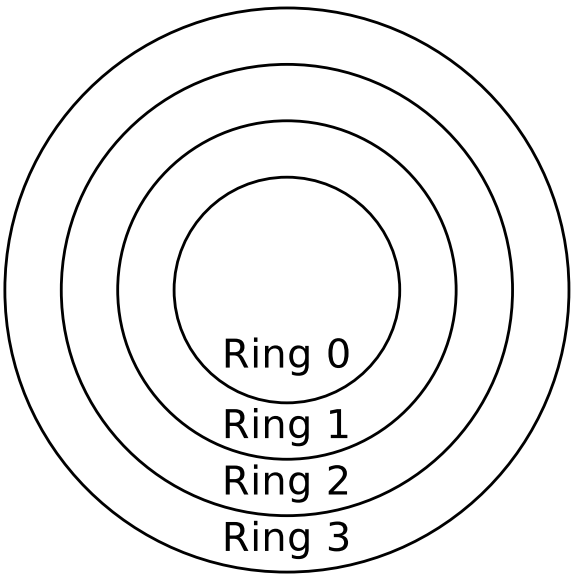
Übungsblatt 7

Aufgabe 1 (Systemaufrufe)

1. x86-kompatible CPUs enthalten 4 Privilegienstufen („Ringe“) für Prozesse. Markieren Sie in der Abbildung (*deutlich erkennbar!*) den Kernelmodus und den Benutzermodus.



2. Nennen Sie den Ring, dem der Betriebssystemkern zugeordnet ist.
3. Nennen Sie den Ring, dem die Anwendungen der Benutzer zugeordnet sind.
4. Nennen Sie den Ring, bei dem Prozesse vollen Zugriff auf die Hardware haben.
5. Nennen Sie einen Grund für die Unterscheidung von Benutzermodus und Kernelmodus.
6. Beschreiben Sie, was ein Systemaufruf ist.
7. Beschreiben Sie, was ein Moduswechsel ist.
8. Nennen Sie zwei Gründe, warum Prozesse im Benutzermodus Systemaufrufe nicht direkt aufrufen sollten.
9. Damit Prozesse im Benutzermodus nicht immer Systemaufrufe aufrufen müssen, gibt es eine alternative Vorgehensweise. Beschreiben Sie diese.

Aufgabe 2 (Prozesse)

1. Nennen Sie die drei Arten von Prozesskontextinformationen, die das Betriebssystem speichert.

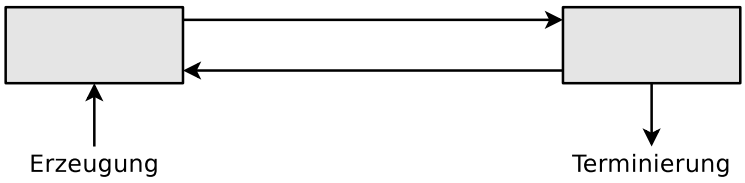
2. Geben Sie an, welche Prozesskontextinformationen nicht im Prozesskontrollblock gespeichert sind.

3. Beschreiben Sie, warum nicht alle Prozesskontextinformationen im Prozesskontrollblock gespeichert sind.

4. Beschreiben Sie die Aufgabe des Dispatchers.

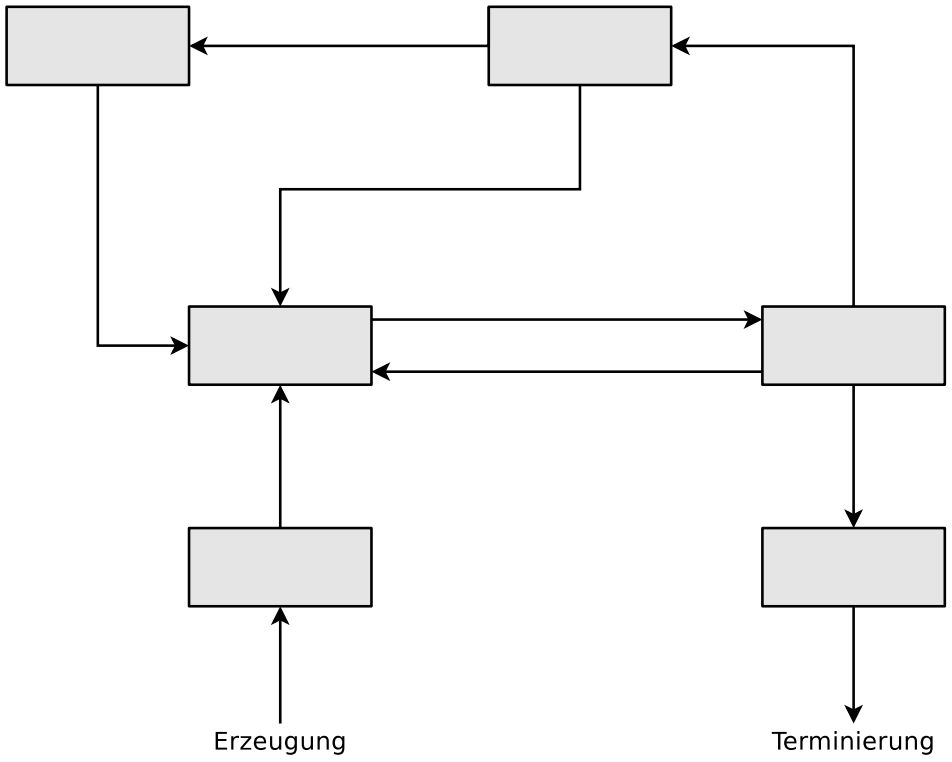
5. Beschreiben Sie die Aufgabe des Schedulers.

6. Das 2-Zustands-Prozessmodell ist das kleinste, denkbare Prozessmodell. Tragen Sie die Namen der Zustände in die Abbildung des 2-Zustands-Prozessmodells ein.

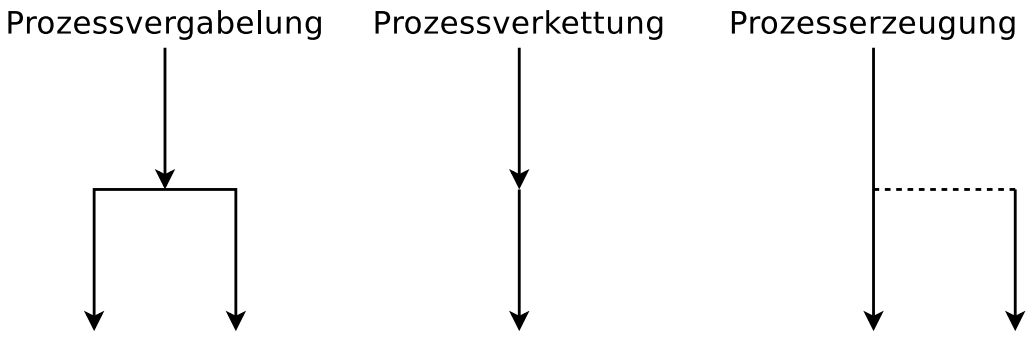


7. Ist das 2-Zustands-Prozessmodell sinnvoll? Begründen Sie kurz ihre Antwort.

8. Tragen Sie die Namen der Zustände in die Abbildung des 6-Zustands-Prozessmodells ein.



9. Beschreiben Sie, was ein Zombie-Prozess ist.
10. Beschreiben Sie die Aufgabe der Prozesstabelle.
11. Wie viele Zustandslisten für Prozesse im Zustand „blockiert“ verwaltet das Betriebssystem?
12. Beschreiben Sie was passiert, wenn ein neuer Prozess erstellt werden soll, es aber im Betriebssystem keine freie Prozessidentifikation (PIDs) mehr gibt.
13. Beschreiben Sie, was der Systemaufruf `fork()` macht.
14. Beschreiben Sie, was der Systemaufruf `exec()` macht.
15. Die drei Abbildungen zeigen alle existierenden Möglichkeiten, einen neuen Prozess zu erzeugen. Schreiben Sie zu jeder Abbildung, welche(r) Systemaufruf(e) nötig sind, um die gezeigte Prozesserzeugung zu realisieren.



16. Ein Elternprozess (PID = 75) mit den in der folgenden Tabelle beschriebenen Eigenschaften erzeugt mit Hilfe des Systemaufrufs `fork()` einen Kindprozess (PID = 198). Tragen Sie die vier fehlenden Werte in die Tabelle ein.

	Elternprozess	**Kindprozess**
PPID	72	
PID	75	198
UID	18	
Rückgabewert von `fork()`		

17. Erklären Sie, was `init` ist.

18. Nennen Sie den Unterschied eines Kindprozess vom Elternprozess kurz nach der Erzeugung.

19. Beschreiben, Sie was passiert, wenn ein Elternprozess vor dem Kindprozess beendet wird.

20. Nennen Sie den Inhalt des Textsegments.

21. Nennen Sie den Inhalt des Heap.

22. Nennen Sie den Inhalt des Stack.

Aufgabe 3 (Zeitgesteuerte Kommandoausführung, Kontrollstrukturen, Archivierung)

1. Schreiben Sie ein Shell-Skript, das zwei Zahlen als Kommandozeilenargumente einliest. Das Skript soll prüfen, ob die Zahlen identisch sind und das Ergebnis der Überprüfung ausgeben.

2. Erweitern Sie das Shell-Skript dahingehend, dass wenn die Zahlen nicht identisch sind, überprüft wird, welche der beiden Zahlen die Größere ist. Das Ergebnis der Überprüfung soll ausgegeben werden.

3. Schreiben Sie ein Shell-Skript, das ein Verzeichnis Ihrer Wahl sichert. Von dem Verzeichnis soll eine Archivdatei mit der Endung `.tar.bz2` erzeugt werden. Diese Datei soll im Verzeichnis `/tmp` abgelegt werden. Der Name der Archivdatei entsprechen soll folgendem Benennungsschema:

   `Backup_<USERNAME>_<JAHR>_<MONAT>_<TAG>.tar.bz2`

   Die Felder `<USERNAME>`, `<JAHR>`, `<MONAT>` und `<TAG>` sollen durch die aktuellen Werte ersetzt werden.

4. Schreiben Sie ein Shell-Skript, das testet, ob heute schon eine Archivdatei gemäß dem Benennungsschema aus Teilaufgabe 3 angelegt wurde. Das Ergebnis der Überprüfung soll in der Shell ausgegeben werden.

5. Schreiben Sie zwei `cron`-Jobs. Der erste `cron`-Job soll an jedem Tag (außer am Wochenende) um 6:15 Uhr das Shell-Skript aus Teilaufgabe 3 aufrufen, das die Archivdatei mit dem Backup erzeugt.

   Der zweite `cron`-Job soll jeden Tag (außer am Wochenende) um 11:45 Uhr das Shell-Skript aus Teilaufgabe 4 aufrufen, das testet, ob heute schon eine Archivdatei angelegt wurde.

   Die Ausgabe der Shell-Skripte soll in eine Datei `/tmp/Backup-Log.txt` angehängt werden. Wenn die Archivdatei `Backup...tar.bz2` erfolgreich erzeugt wurde, soll dieses in der Log-Datei `/tmp/Backup-Log.txt` vermerkt werden.

   Vor jedem neuen Eintrag in die Datei sollen Zeilen nach folgendem Muster (mit aktuellen Werten) in die Log-Datei `/tmp/Backup-Log.txt` eingefügt werden:

```
**************************
20.11.2013 --- 21:39:51 Uhr
```

Aufgabe 4 (Shell-Skripte)

1. Schreiben Sie ein Shell-Skript, das für eine als Argument angegebene Datei feststellt, ob die Datei existiert und ob es sich um eine ein Verzeichnis, einen symbolischen Link, einen Socket oder eine benannte Pipe (FIFO) handelt.

   - Das Skript soll das Ergebnis der Überprüfung ausgeben.

2. Erweitern Sie das Shell-Skript aus Teilaufgabe 1 dahingehend, dass wenn die als Argument angegebene Datei existiert, soll feststellt werden, ob diese ausführt werden könnte und ob schreibend darauf zugegriffen werden könne.

3. Schreiben Sie ein Shell-Skript, das so lange auf der Kommandozeile Text einliest, bis es durch die Eingabe von `ENDE` beendet wird.

   - Die eingelesenen Daten soll das Skript in Großbuchstaben konvertieren und ausgeben.

4. Schreiben Sie ein Shell-Skript, das für alle eingeloggten Benutzer die Anzahl der laufenden Prozesse ausgibt.

5. Erweitern Sie das Shell-Skript aus Teilaufgabe 4 dahingehend, dass die Ausgabe sortiert ausgegeben wird.

   - Der Benutzer mit den meisten Prozessen soll am Anfang stehen.

6. Schreiben Sie ein Shell-Skript, das nach dem Start alle 10 Sekunden überprüft, ob eine Datei `/tmp/lock.txt` existiert.

   - Jedes Mal, nachdem das Skript das Vorhandensein der Datei überprüft hat, soll es eine entsprechende Meldung auf der Shell ausgeben.
   - Sobald die Datei `/tmp/lock.txt` existiert, soll das Skript sich selbst beenden.