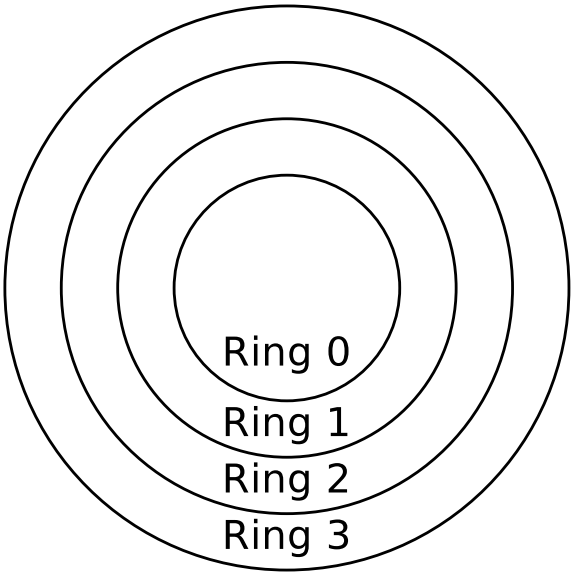
Übungsblatt 4

Aufgabe 1 (Systemaufrufe)

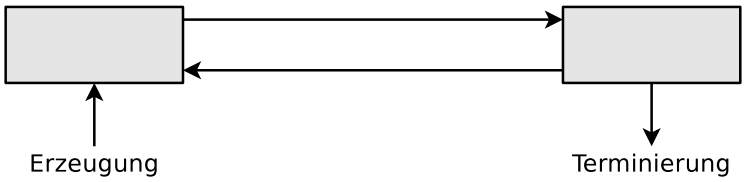
1. x86-kompatible CPUs enthalten 4 Privilegienstufen („Ringe“) für Prozesse. Markieren Sie in der Abbildung (*deutlich erkennbar!*) den Kernelmodus und den Benutzermodus.



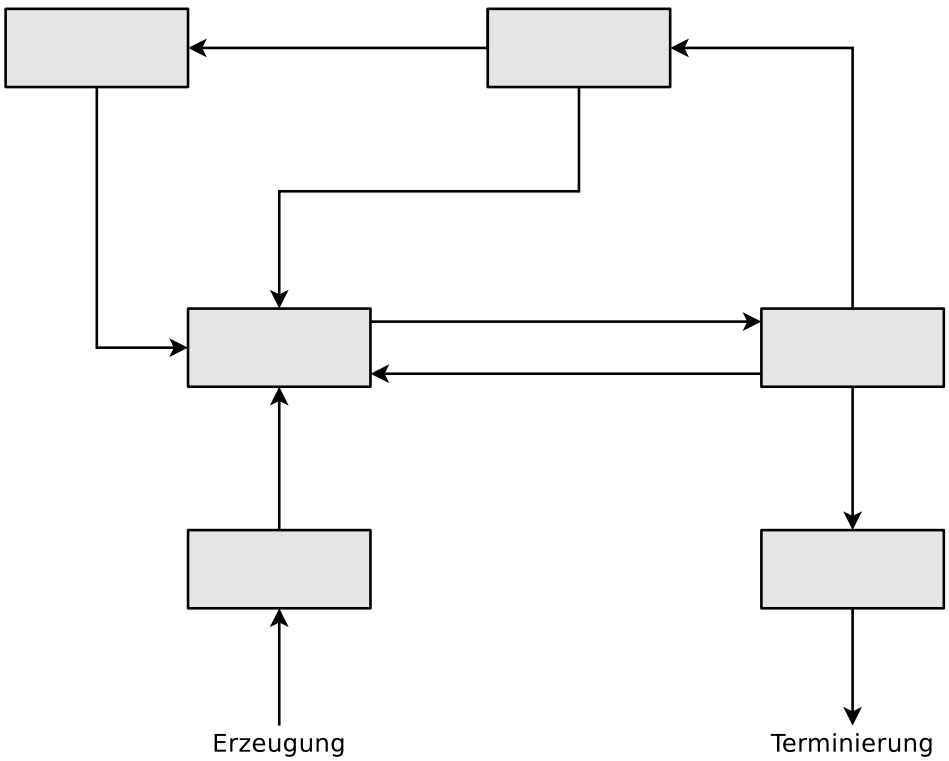
2. In welchem Ring befindet sich der Betriebssystemkern?
3. In welchem Ring befinden sich die Anwendungen der Benutzer?
4. Prozesse in welchem Ring haben vollen Zugriff auf die Hardware?
5. Nennen Sie einen Grund für die Unterscheidung von Benutzermodus und Kernelmodus.
6. Was ist ein Systemaufruf?
7. Was ist ein Moduswechsel?
8. Nennen Sie zwei Gründe, warum Prozesse im Benutzermodus Systemaufrufe nicht direkt aufrufen sollten.
9. Welche Alternative gibt es, wenn Prozesse im Benutzermodus nicht direkt Systemaufrufe aufrufen sollen?

Aufgabe 2 (Prozesse)

1. Welche drei Arten Prozesskontextinformationen speichert das Betriebssystem?
2. Welche Prozesskontextinformationen sind nicht im Prozesskontrollblock gespeichert?
3. Warum sind nicht alle Prozesskontextinformationen im Prozesskontrollblock gespeichert?
4. Was ist die Aufgabe des Dispatchers?
5. Was ist 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. Was ist ein Zombie-Prozess?

10. Welche Aufgabe hat die Prozesstabelle?

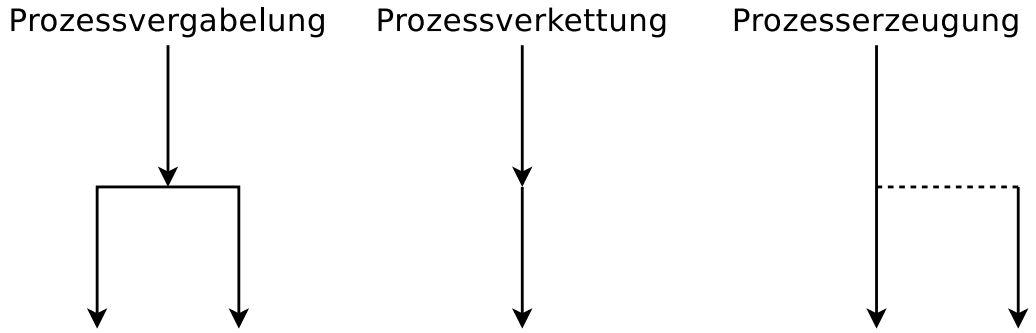
11. Wie viele Zustandslisten für Prozesse im Zustand „blockiert“ verwaltet das Betriebssystem?

12. Was passiert, wenn ein neuer Prozess erstellt werden soll, es aber im Betriebssystem keine freie Prozessidentifikation (PID) mehr gibt?

13. Was macht der Systemaufruf `fork()`?

14. Was macht der Systemaufruf `exec()`?

15. Die drei Abbildungen zeigen alle existierenden Möglichkeiten, einen neuen Prozess zu erzeugen. Schreiben Sie zu jeder Abbildung, welche(r) Systemaufruf(e) nötig ist/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. Was ist `init` und was ist seine Aufgabe?

18. Was unterscheidet einen Kindprozess vom Elternprozess kurz nach der Erzeugung?

19. Was passiert, wenn ein Elternprozess vor dem Kindprozess beendet wird?

20. Welche Daten enthält das Textsegment?

21. Welche Daten enthält der Heap?

22. Welche Daten enthält der Stack?

Aufgabe 3 (Kontrollstrukturen)

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.

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 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.