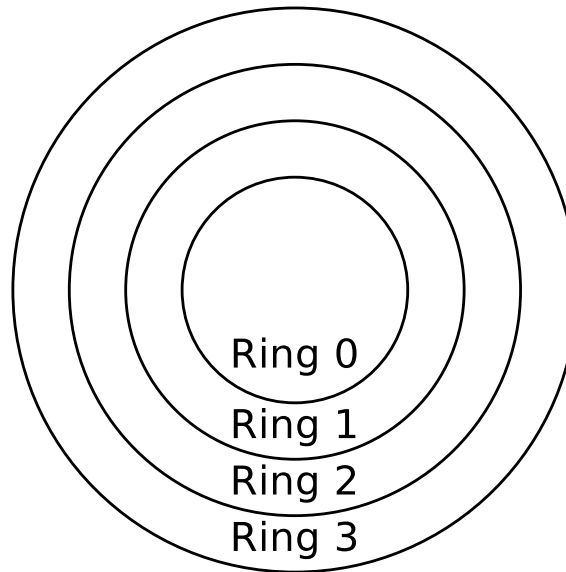


## Übungsblatt 4

### Aufgabe 1 (Systemaufrufe)

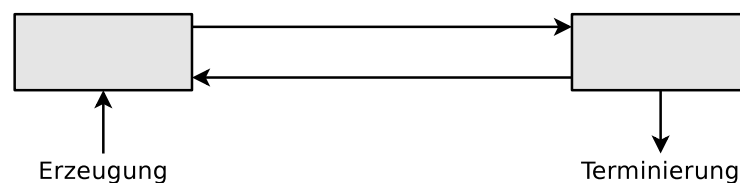
1. x86-kompatible CPUs enthalten 4 Privilegienstufen („Ringe“) für Prozesse. Markieren Sie in der Abbildung (*deutlich erkennbar!*) den Kernelmodus und den Benutzermodus.



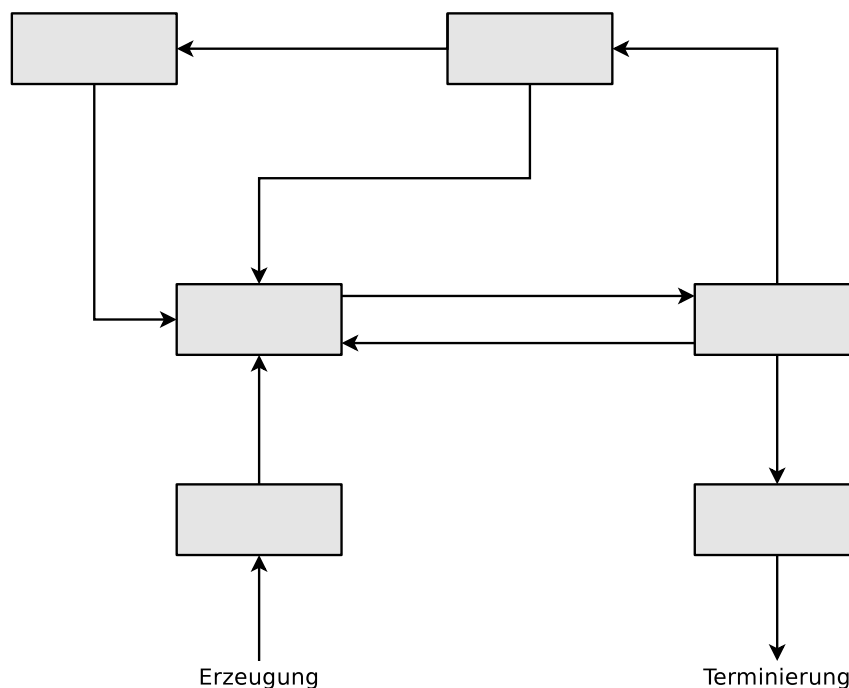
2. Nennen Sie den Ring, dem der Betriebssystemkern zugeordnet ist.
3. Nennen Sie den Ring, dem die Anwendungen der Benutzer zugeordnet sind.
4. Nennen Sie den Ring, in dem Prozesse vollen Zugriff auf die Hardware haben.
5. Nennen Sie einen Grund für die Unterscheidung von Benutzermodus und Kernelmodus.
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. Beschreiben Sie die Alternative, wenn Prozesse im Benutzermodus nicht direkt Systemaufrufe aufrufen sollen.

## Aufgabe 2 (Prozesse)

1. Nennen Sie die drei Arten von Prozesskontextinformationen, die das Betriebssystem speichert.
2. Nennen Sie die Prozesskontextinformation(en), die nicht im Prozesskontrollblock gespeichert sind.
3. Warum sind nicht alle Prozesskontextinformationen im Prozesskontrollblock gespeichert?
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.

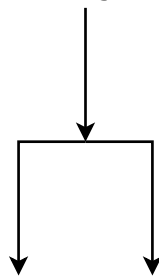


7. Ist das 2-Zustands-Prozessmodell sinnvoll? Begründen Sie kurz ihre Antwort.
8. Tragen Sie die Namen der Zustände in die Abbildung des 6-Zustands-Prozessmodells ein.



9. Beschreiben Sie was ein Zombie-Prozess ist.
10. Beschreiben Sie die Aufgabe der Prozesstabelle.
11. Geben sie an, wie viele Zustandslisten für Prozesse im Zustand „blockiert“ das Betriebssystem verwaltet.
12. Beschreiben Sie was passiert, wenn ein neuer Prozess erstellt werden soll, es aber im Betriebssystem keine freie Prozessidentifikation (PID) 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 ist/sind, um die gezeigte Prozesserzeugung zu realisieren.

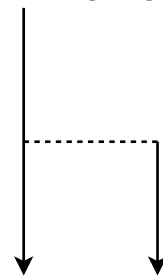
Prozessvergabelung



Prozessverkettung



Prozesserzeugung



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 <code>fork()</code>		

17. Beschreiben Sie was `init` ist und was seine Aufgabe ist.
18. Beschreiben Sie was einen Kindprozess kurz nach der Erzeugung vom Elternprozess unterscheidet.
19. Beschreiben Sie was passiert, wenn ein Elternprozess vor dem Kindprozess beendet wird.
20. Nennen sie die Daten, die das Textsegment enthält.
21. Nennen sie die Daten, die der Heap enthält.
22. Nennen sie die Daten, die der Stack enthält.

## 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 festgestellt werden, ob diese ausgeführt 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.