Virtualisierung

Teil 1: Prozesse und Prozess API

Prof. Dr.-Ing. Andreas Heil

v1.0.1

Lernziele und Kompetenzen

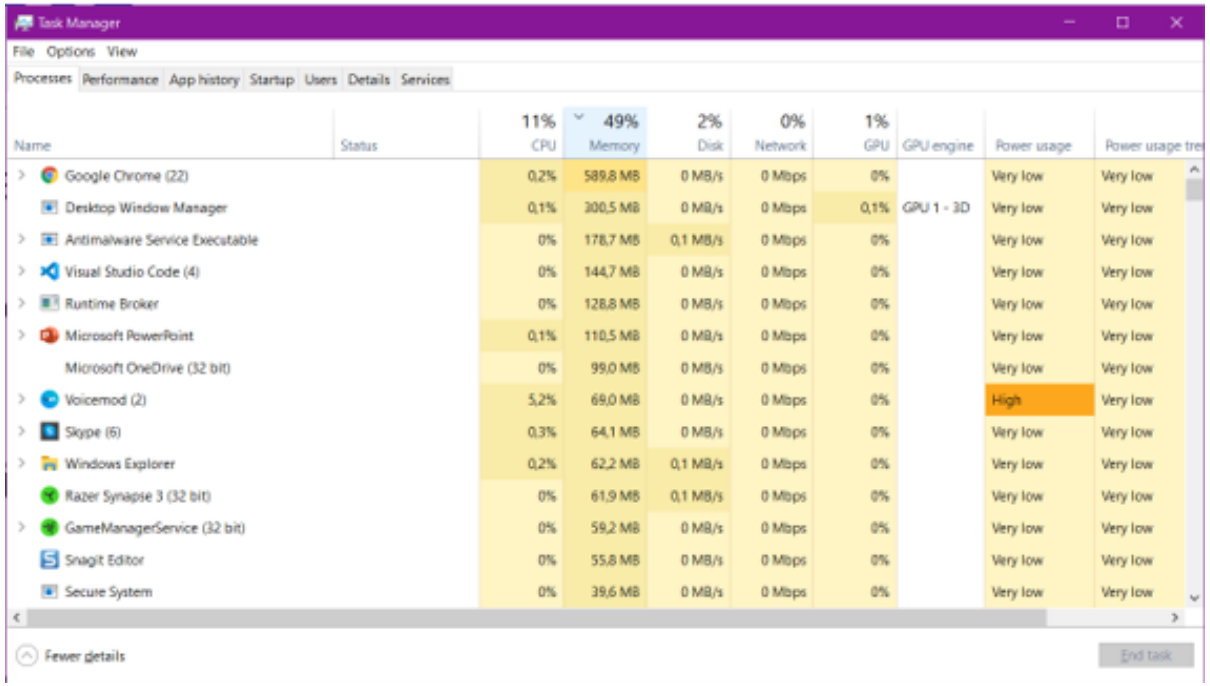
- **Verstehen** wie sich Prozesse zusammensetzen und Prozesse vom Betriebssystem verwaltet werden.
- **Verstehen** wie Prozesse im Betriebssystem gesteuert werden

Motivation

Prozesse: Definitoirisches

»Vereinfachte Definition«: Prozess

Ein ausgeführtes bzw. laufendes Programm



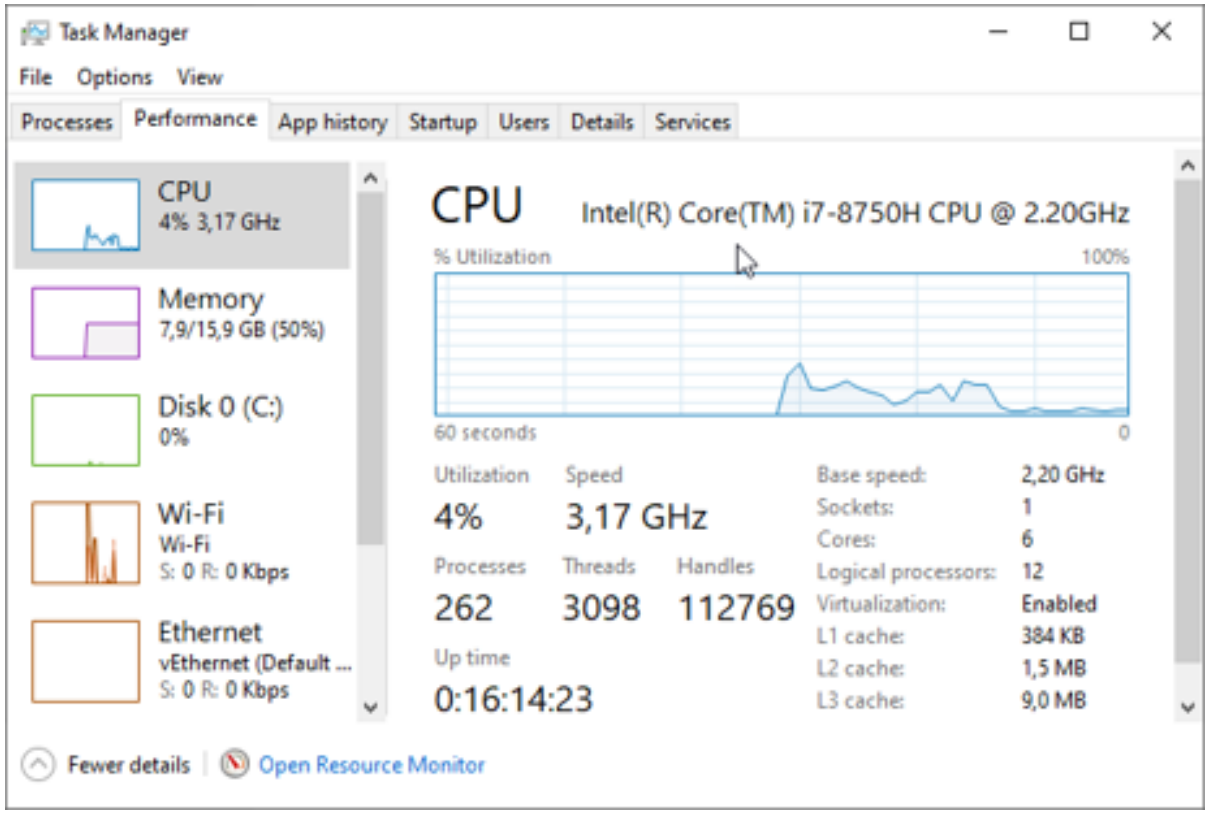
Programme

- Was ist überhaupt ein Programm?
  - Besteht aus Code (Bits) und ggf. statischen Daten
  - Wartet auf der Festplatte und tut nichts
  - Erst durch die Ausführung wird ein Programm zum Prozess
- Was benötigt ein Programm?
  - Benötigt zur Ausführung eine CPU
  - Benötigt für den auszuführenden Code und die Daten Speicher

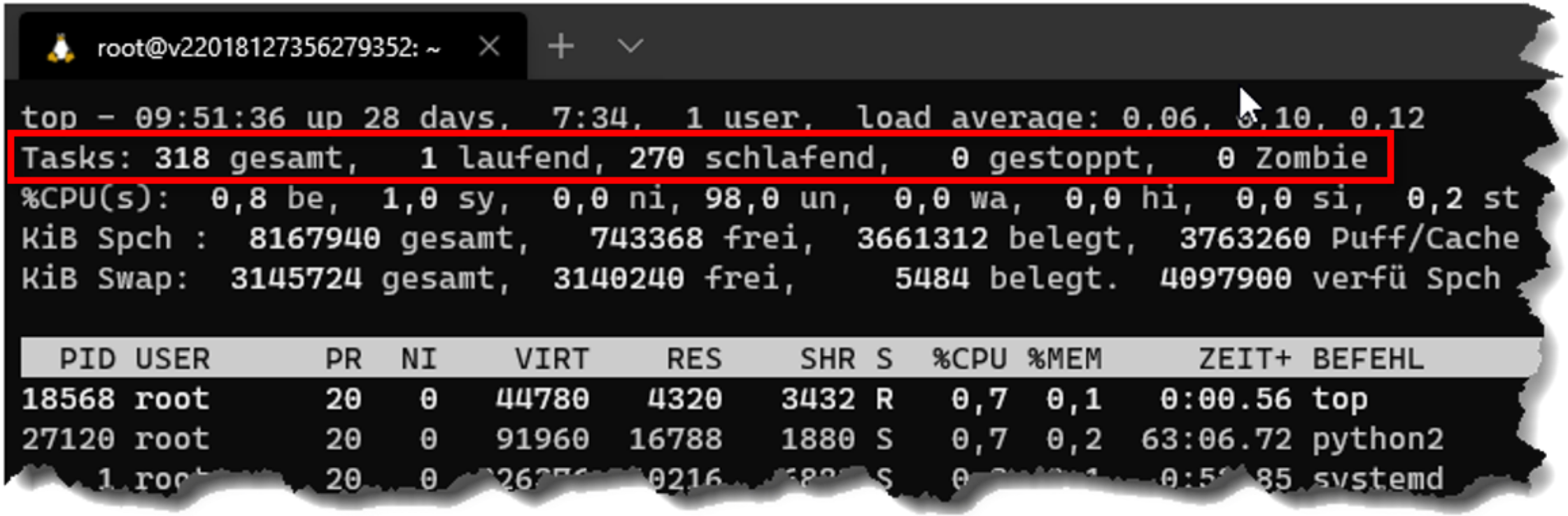
Illusion

Frage: Wie kann die Illusion vieler CPUs geschaffen werden, wenn es nur eine (oder wenige) physikalische CPUs gibt?

Beispiel rechts: Windows Task Manager mit 262 Prozesse



Beispiel: Linux *top*



Was ist Virtualisierung?

- Wir geben jedem Prozess die CPU für eine kurze Zeitspanne
- Dieses sog. »Timesharing« erzeugt eine Illusion mehrerer CPUs
- Konsequenz: Programm läuft langsamer, da die CPU »geteilt« wird

Das ist »sehr vereinfacht« Virtualisierung

Was wird für Virtualisierung benötigt?

- »Low Level Machinery«
  - Methoden und Protokolle für die grundlegende Funktionalität
- »High Level Intelligence«
  - Irgendetwas Geschicktes zum Stoppen und Starten von Programmen
  - Zusätzliches Regelwerk (engl. policies)
  - Regeln wie viele Prozesse auf einer CPU ausgeführt werden dürfen
  - Jemand oder etwas, der bzw. das steuert, welcher Prozess als nächstes ausgeführt wird

Abstraktion von Prozessen

Prozesse bestehen grundlegend aus

- Speicher, in dem die Programmanweisungen bzw. Instruktionen (engl. instructions) liegen
- Speicher, in dem die Daten geschrieben werden
- Vom Prozess adressierbarer Speicher (engl. address space)
- Registern - Instruktionen lesen und schreiben in Register, dies ist notwendig für die Ausführung d. Prozesses

  Diese Informationen können jederzeit »weggespeichert« und wiederhergestellt werden

Spezielle Register, die benötigt werden

- Program Counter (Abk. PC) oder auch Instruction Counter (Abk. IC)
  - Hier steht die nächste Anweisung, die ausgeführt werden soll
- Stack Pointer, Frame Pointer, Funktionsparameter, lokale Variablen und Rücksprungadressen (engl. return address) - mehr dazu später
- Register für I/O-Informationen
  - Liste der Dateien, die der Prozess aktuell geöffnet hat

Prozess-API

Außerdem benötigen wir eine Programmierschnittstelle (engl. process api), die jedes Betriebssystem beinhalten muss (wird später noch weiter vertieft)

- `create`: Ausgewähltes Programm wird gestartet und ein neuer Prozess erzeugt
- `destroy`: Falls sich ein Programm nicht von selbst beendet, ist dies sehr hilfreich
- `wait`: Durchaus sinnvoll zu warten, bis ein Prozess von selbst aufhört zu laufen
- `status`: Statusinformation von Prozessen abfragen

  Weitere Möglichkeiten sind je nach Betriebssystem unterschiedlich, z.B.: `suspend` und `resume` um Prozesse anzuhalten und weiterlaufen zu lassen

Wie wird ein Prozess erzeugt?

1. Voraussetzung: Ein Programm muss in ausführbarer Form vorliegen (mehr dazu später)
2. Programm und statische Daten werden in den Adressraum des Prozesses geladen
    - »Früher« wurde das gesamte Programm in den Speicher geladen (engl. eagerly)
    - »Heute« wird nur der benötigte Programm-Code und die erforderlichen Daten geladen (engl. lazy)

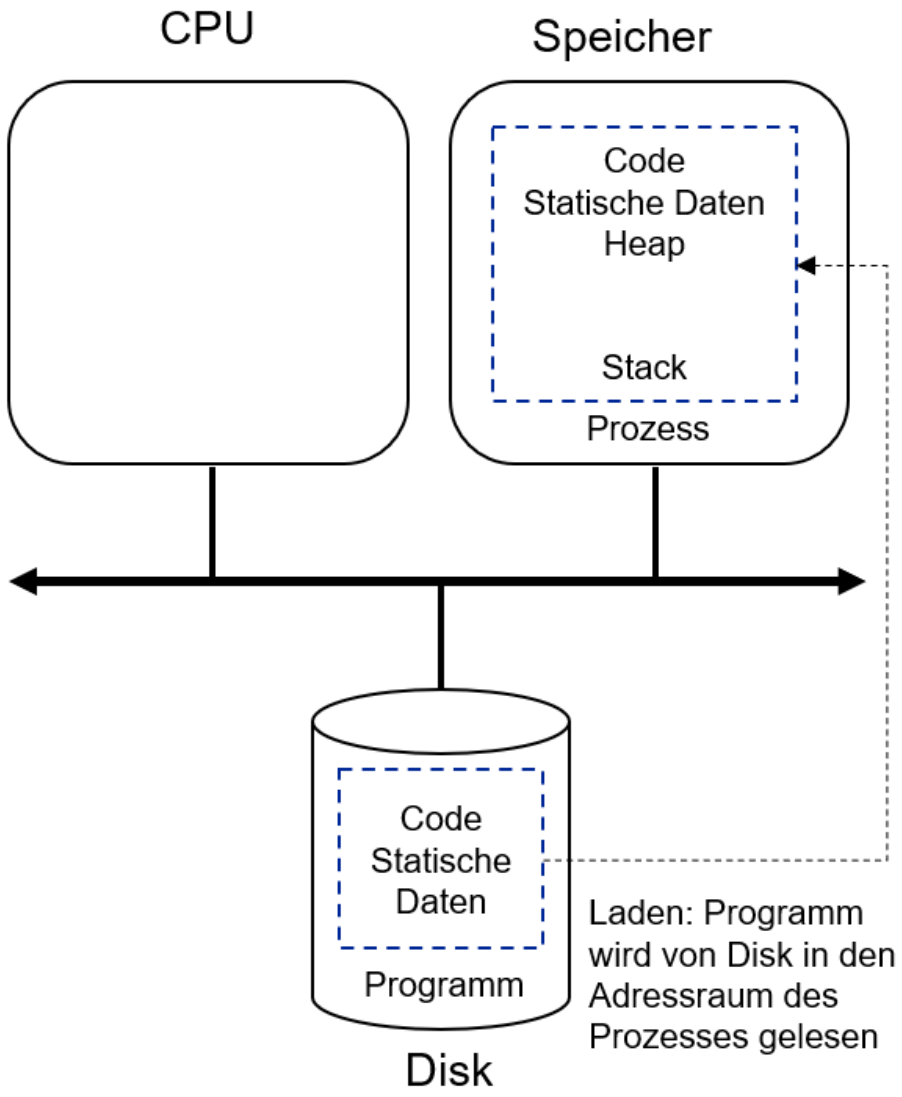
      Um dieses sog. »Lazy Loading« zu verstehen, werden wir uns später noch mit »Paging« und »Swapping« befassen müssen

Wie wird ein Prozess erzeugt? (Forts.)

3. Der sog. »Stack« bzw. »Runtime Stack« wird zugewiesen
    - C nutzt den Stack für lokale Variablen, Funktionsparameter und Rücksprungadressen
4. Das Betriebssystem füllt z.B. die Parameterlisten
    - Bei C sind dies `argc` und `argv`, so dass das Programm (hier die `main`-Funktion) auf die Werte zugreifen kann[^4]
    - Kennen Sie auch aus Java

Wie wird ein Prozess erzeugt? (Forts.)

5. Nun wird noch der Heap reserviert
    - In C für dynamischen Speicherzuordnung via `malloc()` und `free()`
    - Exkurs: Memoryleaks baut man übrigens, indem man in C vergisst `free()` aufzurufen



Wie wird ein Prozess erzeugt? (Forts.)

6. Das Betriebssystem unterstütz nun den Prozess, indem es z.B. dem Prozess mehr Speicher gibt, wenn der Heap vergrößert werden muss
7. Nun werden noch Input/Output-Resourcen erzeugt (sie ahnen es, später mehr dazu)
    - Unter UNIX sind dies die drei sog. »File Descriptors« ^1
        - Standard Input,
        - Standard Output und
        - Standard Error Output

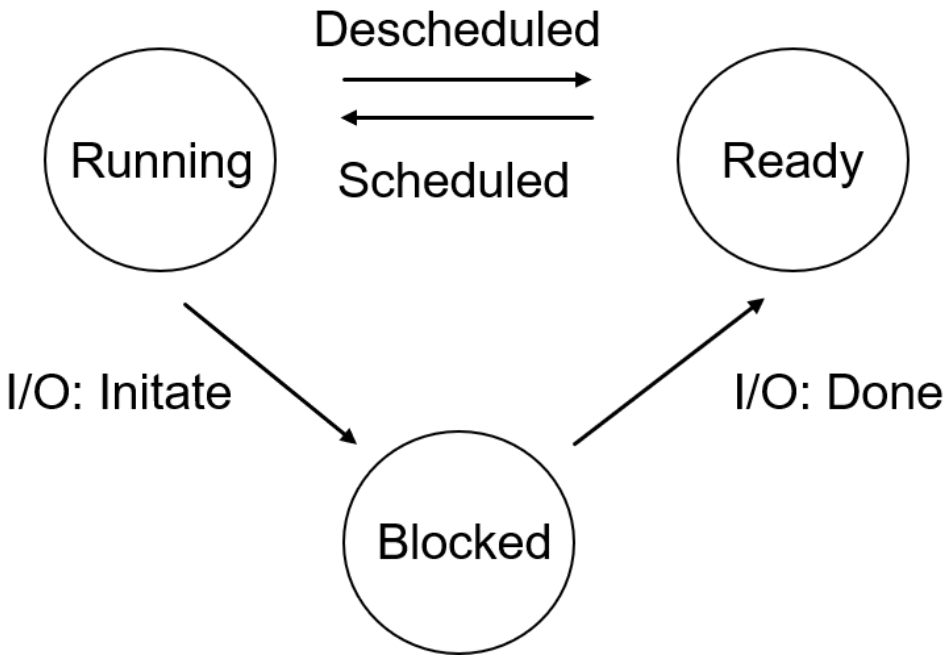
Prozess Status

Was bedeuten eigentlich die Status...?

- Laufend
- Schlafend
- Gestoppt
- Zombie

> Tasks shown as running should be more properly thought of as 'ready to run' -- their task_struct is simply represented on the Linux run-queue. Even without a true SMP machine, you may see numerous tasks in this state depending on top's delay interval and nice value.^2

Mögliche Statusübergänge



Prozessstatus

- **Running:** Prozess läuft auf einer CPU
- **Ready:** Prozess könnte laufen, aber das OS hat entschieden, den Prozess noch nicht laufen zu lassen
- **Blocked:** Prozess hat eine Aktion ausgeführt, die erst abgeschlossen werden kann, wenn ein anderes Ereignis stattgefunden hat - typischerweise handelt es sich hierbei um eine I/O-Operation

  Ist ein Prozess geblockt, wartet das Betriebssystem auf die I/O-Operation, um dann den Prozess wieder in den Status *Ready* zu verschieben.

Ein kleines Problem

Wer entscheidet eigentlich welcher Prozess als nächster gestartet wird?

Der sog. »Scheduler« trifft diese Entscheidung (später mehr dazu)

Bevor wir uns den Scheduler anschauen, müssen wir uns allerdings noch ein paar weitere Gedanken über Prozesse machen…

Ein paar Gedanken zu Prozessen

Wir benötigen

- Eine Datenstruktur für Prozesse
- Eine Liste aller Prozesse
- Eine Liste aller blockierten Prozesse
- Eine Möglichkeit Register bei Stoppen wegzuspeichern und beim Anlaufen des Prozesses wieder zu laden (engl. context switch)

Und was passiert eigentlich, wenn ein Prozess beendet ist, aber noch nicht alles »aufgeräumt« wurde?

In UNIX-Systemen haben solche Prozesse einen eigenen Status: **Zombie**

Exkurs: Datenstruktur von xv6-Prozessen

Alle Informationen über einen Prozess stehen in einem Prozesskontrollblock (engl. process control block, kurz PCB)

```
26  // but it is on the stack and allocproc() manipulates it.
27  struct context {
28    uint edi;
29    uint esi;
30    uint ebx;
31    uint ebp;
32    uint eip;
33  };
34
35  enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
36
37  // Per-process state
38  struct proc {
39    uint sz;                     // Size of process memory (bytes)
40    pde_t* pgdir;                // Page table
41    char *kstack;                // Bottom of kernel stack for this process
42    enum procstate state;        // Process state
43    int pid;                     // Process ID
44    struct proc *parent;         // Parent process
45    struct trapframe *tf;        // Trap frame for current syscall
46    struct context *context;     // swtch() here to run process
47    void *chan;                  // If non-zero, sleeping on chan
48    int killed;                  // If non-zero, have been killed
49    struct file *ofile[NOFILE];  // Open files
50    struct inode *cwd;           // Current directory
51    char name[16];               // Process name (debugging)
52  };
53
```

Zusammenfassung

- Prozesse sind die grundlegende Abstraktion eines Programmes
- Zu jedem Zeitpunkt kann ein Prozess über seinen Status, den Speicherinhalt, seinen Adressraums, den Inhalt der CPU-Register (einschl. program counter und stack pointer) und den I/O-Informationen (d.h. geöffnete Dateien) beschrieben werden
- Die Prozess-API besteht aus Aufrufen, die in Zusammenhang mit Prozessen ausgeführt werden können, z.B. zum Erzeugen oder Beenden von Prozessen
- Unterschiedliche Ereignisse führen zu Statusänderungen im Prozess (z.B. der Aufruf einer blockierenden I/O-Operation)
- Eine Prozessliste enthält alle Informationen über die Prozesse auf einem System