Prozess . . . Programm in Ausführung

Kontrolliert durch Programme, exekuiert auf einen Prozessor

Prozess, kann die Ausführung mehrerer Programme bedeutet (S. III-34)

- ein Anwendungsprogramm ruft ein Betriebssystemprogramm auf
  - Systemaufruf (engl. *system call*)
  - Programmunterbrechung (engl. *trap*, *interrupt*)
- ein Prozess ist Aktivitätsträger von ggf. mehreren Programmen
  - Adressraumüberlagerung mit einem anderem Programm (`exec(2)`)

Programm, kann von mehreren Prozessen ausgeführt werden

- nicht-sequentielles Programm, im Falle von Uniprozessorsystemen
  - präemptive (d.h. verdrängende) Programmverarbeitung
  - Aufgabe (engl. *task*), Faden (engl. *thread*)
- paralleles Programm im Falle von Multiprozessorsystemen

Prozess $\neq$ Programm

Programm ist statisch, Prozess ist dynamisch

Wissen über das gegenwärtig ausgeführte Programm sagt nicht viel aus über die zu dem Zeitpunkt im System stattfindende Aktivität.

- Welches Zugriffsrecht besitzt das Programm zur Zeit?
  - auf ein Adressraumsegment, auf eine Datei, auf ein Gerät, . . .
  - allgemein: auf ein Betriebsmittel
- Welcher Kontrollfluss ist im mehrfädigen Programm zur Zeit aktiv?
  - Uni- vs. Multiprozessorsystem (SMP)
- Wieviel Programmunterbrechungen sind zur Zeit gestapelt?

⋮

Im Betriebssystemkontext ist das Konzept „Prozess“ daher nützlicher als das Konzept „Programm“, um Abläufe zu beschreiben und zu verwalten.

Prozess $\neq$ Prozessinstanz

Analogie zu Typ oder Klasse einerseits und Instanz bzw. Objekt andererseits

Prozess, ein abstraktes Gebilde

- ein „Programm in Ausführung“, sequentieller Kontrollfluss
- ein „Ablauf“, der eine Verwaltungseinheit ist

Prozessinstanz (auch: Prozessinkarnation), ein konkretes Gebilde

- die „physische Instanz“ des abstrakten Gebildes „Prozess“
  - an Betriebsmittel (Ressource; engl. *resource*) gebunden
  - die Identität (engl. *identity*) einer Programmausführung
- die Verwaltungseinheit, die einen Prozess beschreibt und repräsentiert
  - „dynamische Datenstruktur“ verschiedenartiger Strukturelemente

☞ synonyme Verwendung der Begriffe kann zu Missverständnissen führen

Prozessmodelle

Gewichtsklassen

schwergewichtiger Prozess (engl. *heavyweight process*)

- Prozessinstanz und Benutzeradressraum bilden eine Einheit
- Prozesswechsel $\leadsto$ zwei Adressraumwechsel: $AR_x \Rightarrow BS \Rightarrow AR_y$
  - „klassischer“ UNIX Prozess

leichtgewichtiger Prozess (engl. *lightweight process*)

- Prozessinstanz und Adressraum sind voneinander entkoppelt
- Prozesswechsel $\leadsto$ einen Adressraumwechsel: $AR_x \Rightarrow BS \Rightarrow AR_x$
  - Kernfaden (engl. *kernel thread*): Faden auf Kernebene

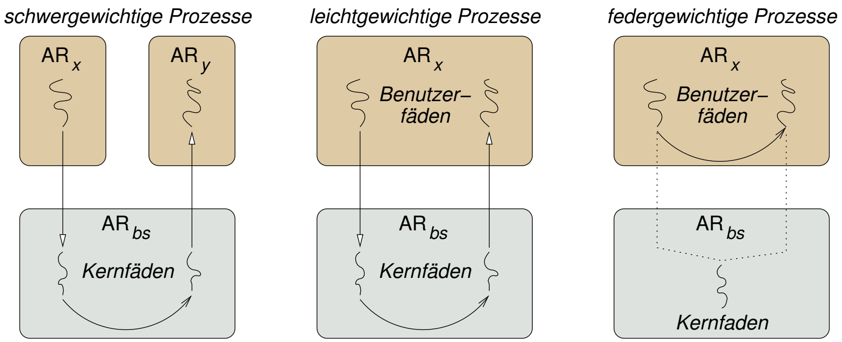
federgewichtiger Prozess (engl. *featherweight process*)

- Prozessinstanz<u>en</u> und Adressraum bilden eine Einheit
- Prozesswechsel $\leadsto$ kein Adressraumwechsel: $AR_x \Rightarrow AR_x$
  - Benutzerfaden (engl. *user thread*): Faden auf Benutzerebene

Kern-/Benutzerfaden $\Rightarrow$ Betriebssystem-/Benutzerprogramm (S. III-34)

Prozessmodelle (Forts.)

Schwer- vs. leicht- vs. federgewichtige Prozesse



Adressraumwechsel sind (je nach MMU) mehr oder weniger „teuer“

- die zur Adressumsetzung benötigten Deskriptoren werden mit jedem Wechselvorgang aus dem Zwischenspeicher (engl. *cache*) verdrängt
- erneute Adressraumaktivierung hat zur Folge, dass die MMU die Adressraumdeskriptoren erst wieder zwischenspeichern muss

Prozessbenutzthierarchie

Implementierung von Prozessen

schwergewichtiger Prozess

⇓

leichtgewichtiger Prozess

⇓

federgewichtiger Prozess

Basis ist der federgewichtige Prozess

- der eigentliche Kontrollfluss
- Steuerbefehle sind Prozeduren des auszuführenden Programms
  - erzeugen, wechseln, zerstören

Erweiterungen zum Mehrprogrammbetrieb bedeuten „Gewichtszunahme“

- leichtgewichtiger Prozess: vertikale Isolation vom Betriebssystem
  - Steuerbefehle sind Systemaufrufe an den Betriebssystemkern
- schwergewichtiger Prozess: horizontale Isolation von anderen Fäden
  - jeder Faden besitzt seinen eigenen (logischen/virtuellen) Adressraum

Implementierungskonzept von Prozess(instanz)en ist die Koroutine [48]

- in mehr oder weniger stark funktional angereicherter Form

Einplanung von Prozessen

Planung ihres zeitlichen Ablaufs (engl. *scheduling*)

Prozesseinplaung (engl. *process scheduling*) stellt sich allgemein zwei grundsätzlichen Fragestellungen:

1. Zu welchem Zeitpunkt sollen Prozesse ins System eingespeist werden?
2. In welcher Reihenfolge sollen Prozesse ablaufen?

Zuteilung von Betriebsmitteln an konkurrierende Prozesse kontrollieren

Einplanungsalgorithmus (engl. *scheduling algorithm*)

Implementiert die Strategie, nach der ein von einem Rechnersystem zu leistender Ablaufplan zur Erfüllung der jew. Anwendungsanforderungen entsprechend aufzustellen und zu aktualisieren ist.

Einplanung von Prozessen (Forts.)

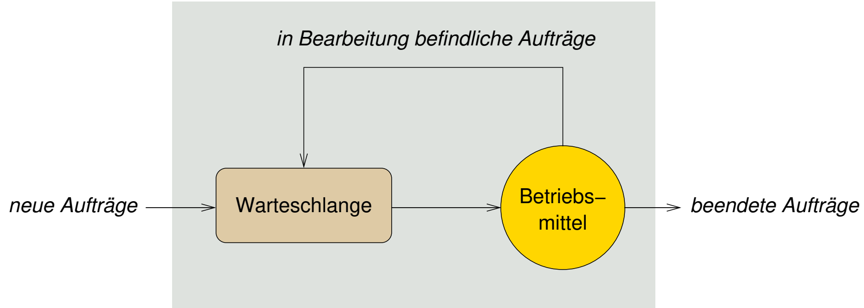
Reihenfolge festlegen, Aufträge sortieren

Ablaufplan (engl. *schedule*) zur Betriebsmittelzuteilung erstellen

- geordnet nach Ankunft, Zeit, Termin, Dringlichkeit, Gewicht, ...
- entsprechend der jeweiligen Einplanungsstrategie
- zur Unterstützung einer bestimmten Rechnerbetriebsart

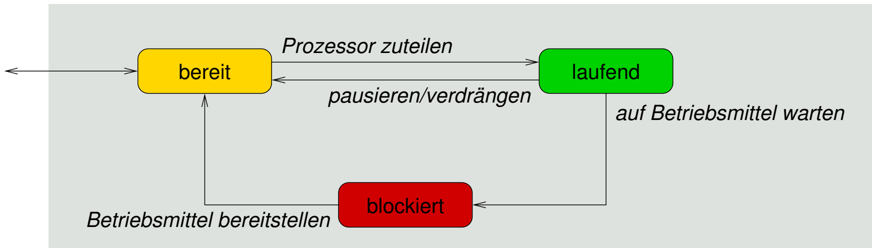
Prozessen das Betriebsmittel { Prozessor, Speicher, Gerät, ⋮, Datei, Nachricht, Puffer } koordiniert zuteilen

Prinzipielle Funktionsweise von Einplanungsalgorithmen
Verwaltung von (betriebsmittelgebundenen) Warteschlangen



Ein einzelner Einplanungsalgorithmus charakterisiert sich durch die Reihenfolge von Prozessen in der Warteschlange und die Bedingungen, unter denen die Prozesse in die Warteschlange eingereiht werden. [32]

Verarbeitungszustände von Prozessen
Zustandsübergänge implementiert ein Planer (engl. *scheduler*)



Prozessverarbeitung impliziert die Verwaltung mehrerer Warteschlangen:
- häufig ist jedem Betriebsmittel eine eigene Warteschlange zugeordnet
  - in der die Prozesse dann auf die Zuteilung dieses Betriebsmittels warten
- im Regelfall sind in Warteschlangen stehende Prozesse blockiert. . .
  - mit Ausnahme der Bereitliste (engl. *ready list*) der CPU
  - die auf Zuteilung der CPU wartenden Prozesse sind laufbereit

Warteschlangentheorie
Theoretische Grundlagen des Scheduling

Betriebssysteme durch die „theoretische/mathematische Brille“ gesehen:
- R. W. Conway, L. W. Maxwell, L. W. Millner. *Theory of Scheduling*.
- E. G. Coffman, P. J. Denning. *Operating System Theory*.
- L. Kleinrock. *Queuing Systems, Volume I: Theory*.

Einplanungsverfahren stehen und fallen mit den Vorgaben der Zieldomäne
- die „Eier-legende Wollmilchsau“ kann es nicht geben
- Kompromisslösungen sind geläufig, aber nicht in allen Fällen tragfähig

Scheduling ist ein Querschnittsbelang (engl. *cross-cutting concern*)

UNIX Scheduling
Charakteristische Eigenschaften — Ausnahmen bestätigen die Regel

Linux, MacOS, SunOS
- die Verfahren wirken verdrängend (engl. *preemptive*)
  - Prozesse können das Betriebsmittel „CPU“ nicht monopolisieren
  - dem laufenden Prozess kann die CPU entzogen werden (CPU-Schutz)
- der fortgeschriebene Ablaufplan ist nicht-deterministisch
  - nicht zu jedem Zeitpunkt ist bestimmt, wie weitergefahren wird
  - die exakte Vorhersage der Prozessorauslastung ist nicht möglich
- Prozessausführung und -einplanung sind gekoppelt (engl. *online*)
  - dynamische Prozesseinplanung während der Programmausführung
  - Planungsziel: Antwortzeiten minimieren, Interaktivität fördern
- das System arbeitet im Zeitmultiplexbetrieb (engl. *time sharing*)

UNIX Prozess

Schwergewicht: Prozess und Adressraum bilden eine Einheit

```c
int foo;
int hal = 42;

int main () {
    for (;;)
        printf("Die Antwort auf alle Fragen lautet %d\n",
            hal + foo);
}
```

Wie ist der Adressraum bzw. Speicher des Prozesses organisiert, der die Ausführung dieses Programms bewirkt?

UNIX Prozess (Forts.)

Adressraumabbildung unter SunOS

```
wosch@faui40 40$ gcc -O6 -static -o hal hal.c; ./hal
Die Antwort auf alle Fragen lautet 42 ...^Z
wosch@faui40 41$ ps
   PID TTY       TIME CMD
 28426 pts/4     0:00 hal
   205 pts/4     0:00 ps
 25965 pts/4     0:00 tcsh-6.0
wosch@faui40 42$ pmap -x 28426
28426:   ./hal
 Address   Kbytes     RSS     Anon  Locked Mode   Mapped File
00010000      216     216        -       - r-x--  hal
00054000       16      16        8       - rwx--  hal
00058000        8       8        8       - rwx--    [ heap ]
FFBFE000        8       8        8       - rw---    [ stack ]
-------- ------- ------- ------- -------
total Kb      248     248       24       -
wosch@faui40 43$
```


UNIX Prozess (Forts.)

Pseudobefehle stecken Text-/Datenbereiche ab (gcc -S -O

```
    .file    "hal.c"
    .global  hal
    .section ".data"
    .align   4
    .type    hal,#object
    .size    hal,4
hal:
    .uaword  42
    .common  foo,4,4
    .section ".rodata"
    .align   8
.LLC0:
    .asciz   "Die Antwort auf alle Fragen lautet %d\n"
    .section ".text"
    .align   4
    .global  main
    .type    main,#function
    .proc    04
```

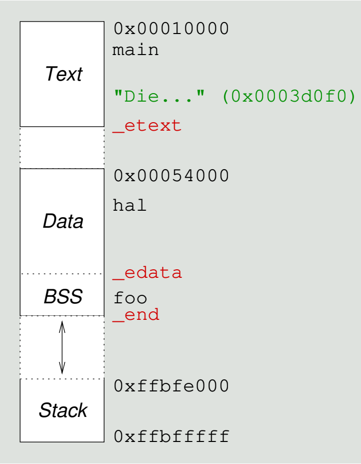
```
main:
    !#PROLOGUE# 0
    save     %sp, -112, %sp
    !#PROLOGUE# 1
    sethi    %hi(hal), %l2
    sethi    %hi(foo), %l1
    sethi    %hi(.LLC0), %l0
    ld       [%l2+%lo(hal)], %g1
.LL5:
    or       %l0, %lo(.LLC0), %o0
    ld       [%l1+%lo(foo)], %o3
    call     printf, 0
    add      %g1, %o3, %o1
    b        .LL5
    ld       [%l2+%lo(hal)], %g1
.LLfe1:
    .size    main,.LLfe1-main
    .ident   "GCC: (GNU) 3.0.4"
```


UNIX Prozess (Forts.)

Adressraumsegmente unter SunOS: Text, Daten, BSS, Stapel

```
wosch@faui40 43> nm -p -g hal
 :
 0000066112 T main      ↦ 0x00010240
 0000352140 D hal       ↦ 0x00055f8c
 0000360336 B foo       ↦ 0x00057f90
 :
 0000286461 D _etext    ↦ 0x00045efd
 0000358433 D _edata    ↦ 0x00057821
 0000361444 D _end      ↦ 0x000583e4
 :
```

Nicht alle Übersetzer/Binder unter UNIX verwenden den Unterstrich ('_'), um die Symbole problemorientierter Programmiersprachen von Symbolen der Assembliersprachen unterscheiden zu können.



UNIX Prozess (Forts.)

Symbolische Adressen unterteilen den statischen Adressraum

Symbole, die vom Binder definiert und mit Werten belegt werden:

`extern etext`

- die erste Adresse nach dem Programmtext

`extern edata`

- die erste Adresse nach dem initialisierten Datenbereich

`extern end`

- die erste Adresse nach dem uninitialisierten Datenbereich
- entspricht anfangs der „Bruchstelle“ des Programms (☞ Aufgabe 4)
  - kann zur Ausführungszeit verschoben werden (`brk(2)`/`sbrk(2)`)
- `sbrk((intptr_t*)0)` liefert den aktuell gültigen Wert

BSS (engl. *block started by symbol*, [52, 53]) initialisiert der Lader mit 0

- der Binder legt nur die Größe fest: [`edata`, `end`[

UNIX Systemfunktionen

Operationen auf Prozesse und Prozessadressräume

Linux, MacOS, SunOS

```
pid = fork()
pid = wait(status)
void _exit(status)
pid = getpid()
pid = getppid()
ok  = nice(incr)
err = execv(path, argv)
err = execve(path, argv, envp)
    ⋮
```


Parthenogenese in UNIX (☞ Aufgabe 3)

Aufspalten, abwarten und beenden

```
#include <sys/types.h>
#include <sys/wait.h>

char parent[] = "Elter: ", child[] = " Kind: ";

main() {
    pid_t pid;
    int zwerg;

    switch ((pid = fork())) {
        case -1:
            perror("fork");
            exit(1);
        case 0:
            printf("%sHier ist der Kindprozess, meine PID ist %d.\n", child, getpid());
            printf("%sDie PID meines Elterprozesses ist %d.\n", child, getppid());
            printf("%sGib mir einen (kleinen Wert als) Exitstatus: ", child);
            scanf("%d", &zwerg);
            printf("%sDanke und Tschüss!\n", child);
            exit(zwerg);
        default:
            printf("%sHier ist ein Elterprozess, meine PID ist %d.\n", parent, getpid());
            printf("%sDie PID meines Kindprozesses ist %d...\n", parent, pid);
            wait(&zwerg);
            printf("%sDer Exitstatus meines Kindprozesses ist %d.\n", parent, WEXITSTATUS(zwerg));
            printf("%sHollaröhdulliöh!\n", parent);
    }
}
```


Parthenogenese in UNIX (Forts.)

Ablaufprotokoll der Interaktion Elter ↔ Kind

```
wosch@gondor 71$ gcc -O6 -o fork fork.c
wosch@gondor 72$ ./fork
Elter: Hier ist ein Elterprozess, meine PID ist 1984.
Elter: Die PID meines Kindprozesses ist 1985...
 Kind: Hier ist der Kindprozess, meine PID ist 1985.
 Kind: Die PID meines Elterprozesses ist 1984.
 Kind: Gib mir einen (kleinen Wert als) Exitstatus: 42
 Kind: Danke und Tschüss!
Elter: Der Exitstatus meines Kindprozesses ist 42.
Elter: Hollaröhdulliöh!
wosch@gondor 73$
```