F Prozesse

F.1 Überblick

- UNIX-Prozesse
  - ◆ Prozeßbild, Speicherorganisation
  - ◆ Verwaltungsstrukturen, Identifikatoren
  - ◆ Prozeßzustände
  - ◆ Erzeugen von Prozessen
  - ◆ Prozeßumschaltungen
  - ◆ Ausführen von Programmen
  - ◆ Beenden von Prozessen

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.1 Überblick (2)

- MACH — Tasks und Threads
  - ◆ Motivation
  - ◆ Tasks
    - ➤ Konzept
    - ➤ Anwendungsschnittstelle
  - ◆ Threads
    - ➤ User-level-Threads / Kernel-Threads
    - ➤ P-Threads
    - ➤ Anwendungsschnittstelle

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.1 Überblick (3)

- Koordinierung
  - ◆ UNIX-Koordinierungsmechanismen
    - ➤ Semaphore
    - ➤ Kern-Koordinierung (sleep/wakeup)
  - ◆ Pthreads-Koordinierung
    - ➤ Mutexes
    - ➤ Condition Variables
- Scheduling
  - ◆ UNIX
    - ➤ SystemV
    - ➤ BSD
  - ◆ MACH

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.2 UNIX — Prozeßbild und Speicherorganisation

1 Speicherorganisation eines Programms

- Format einer ausführbaren Datei (***a.out***- oder ***executable*-Format)**

symbol table
initialized data
text
header
magic-number

magic number	zur Identifizierung des Dateiformats (z. B. verschiedene *executable* Formate möglich)
header	Verwaltungsinformationen (z. B. Größen der einzelnen Segmente)
text	Programmkode
initialized data	initialisierte globale und *static* Variablen
symbol table	Zuordnung der im Programm verwendeten symbolischen Namen von Funktionen und globalen Variablen zu Adressen (z. B. für Debugger)

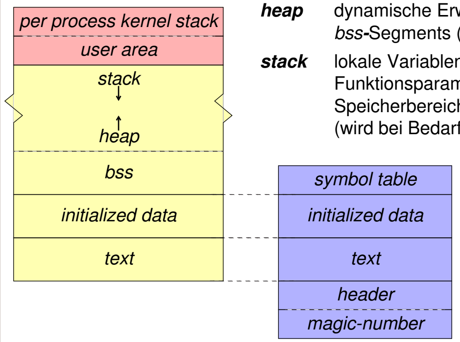
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Speicherorganisation eines Prozesses

bss nicht initialisierte globale und *static* Variablen (wird vor der Vergabe an den Prozess mit 0 vorbelegt)

heap dynamische Erweiterungen des *bss*-Segments (***sbrk(2)***, ***malloc(3)***)

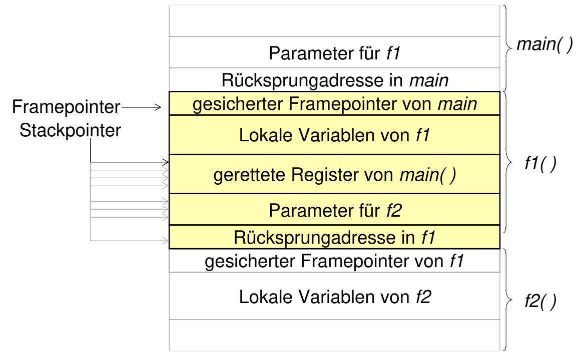
stack lokale Variablen, Funktionsparameter, Speicherbereiche für Registerinhalte, (wird bei Bedarf dynamisch erweitert)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stackaufbau eines Prozesses (2)

- Aufbau eines **Stack-Frames** (Funktionen *main( ), f1( ), f2( )*)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stackaufbau eines Prozesses

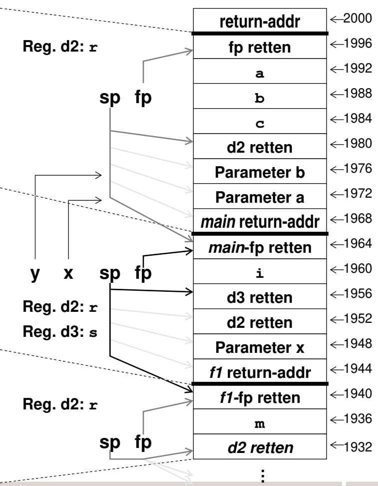
- für jede Funktion wird ein ***Stack-Frame*** angelegt, in dem
  - lokale Variablen der Funktion
  - Aufrufparameter an weitere Funktionen
  - Registerbelegung der Funktion während des Aufrufs weiterer Funktionen

  gespeichert werden
- Stackorganisation ist abhängig von
  - Prozessor
  - Compiler und
  - Betriebssystem
- Beispiele aus einem UNIX auf Motorola 68k-Prozessor
  - typisch für CISC-Prozessoren
  - RISC-Prozessoren mit Registerfiles gehen anders vor!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stack mehrerer Funktionsaufrufe

```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
f1(x,y) {
        int i;
        register r;
        register s;

        x++;

        i = f2(x);

        return(i);
}
f2(z) {
        int m;
        register r;

        r = z;
        m = 100;

        return(z+1);
}
```



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stackaufbau (4)

■ Motorola 68000-Assembler — main( )

```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
```

```
#NO_APP
gcc2_compiled.:
.text
        .even
.globl _main

_main:
        link a6,#-12        # alten Framepointer retten und Stackbereich fuer lokale Var. reservieren (12 Byte)
        movel d2,sp@-       # Register d2 fuer Variable r retten - auf Stack legen und Stackpointer dekr.
        jbsr ___main        # (C++ - Schnittstelle - hier ihne Bedeutung)
        moveq #10,d1        # 10 in Hilfsregister d1 laden
        movel d1,a6@(-4)    # a (Framepointer - 4) = 10 (aus Register d1)
        moveq #20,d1        # 20 in Hilfsregister d1 laden
        movel d1,a6@(-8)    # b (Framepointer - 8) = 20 (aus Register d1)
        moveq #40,d2        # r (Register d2) = 40
        movel a6@(-8),sp@-# b (Framepointer - 8) auf Stack legen, Stackpointer um Wortlaenge (4 Byte) dekr.
        movel a6@(-4),sp@-# a (Framepointer - 4) auf Stack legen, Stackpointer um Wortlaenge (4 Byte) dekr.
        jbsr _f1            # Funkt. f1 aufrufen, dabei Ruecksprungaddr. auf Stack legen und Stackpointer dekr.

                            # ... hier Fortsetzung nach return aus f1
        addqw #8,sp         # Stackpointer um 8 erhoehen - damit entfernen der Aufrufpar. a und b (je 4 Byte)
        movel a6@(-4),d0    # a (Framepointer - 4) in Register d0 laden (Rueckgabeparameter)
        jra L1              # jump auf return-code

L1:                         # return-code fuer main
        movel a6@(-16),d2   # Register d2 restaurieren (war fuer Var. r auf Stack gerettet worden)
        unlk a6             # geretten alten Framepointer restaurieren, Stackpointer = Framepointer
        rts                 # return - Ruecksprungadresse von Stack nehmen - Stackpointer inkrementieren
```


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998 F-Prozesse.doc 1998-01-14 08.33
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stackaufbau (5)

■ Motorola 68000-Assembler — Funktion f1( )

```
f1(x, y)
{
        int i;
        register r, s;
        x++;
        i = f2(x);
        return(i);
}
```

```
.even
.globl _f1

_f1:
        link a6,#-4         # Framepointer der Funktion main retten und
                              Stackbereich fuer lokale Var. reservieren (4 Byte)
        movel d3,sp@-       # Register d3 fuer Var. s retten - auf Stack legen und Stackptr. dekr.
        movel d2,sp@-       # Register d2 fuer Var. r retten - auf Stack legen und Stackptr. dekr.
        addql #1,a6@(8)     # 1 auf Parameter x addieren (x liegt auf Adresse Stackpointer + 8)
        movel a6@(8),sp@-   # x auf Stack legen, Stackpointer um Wortlaenge (4 Byte) dekrementieren
        jbsr _f2            # f2 aufrufen, dabei Ruecksprungaddr. auf Stack legen und Stackpointer dekr.

                            #  ... hier Fortsetzung nach return aus f2
        addqw #4,sp         # Stackpointer um 4 erhoehen - damit entfernen des Aufrufparameters x (4 Byte)
        movel d0,a6@(-4)    # Rueckgabewert von f2 (steht in Reg. d0) in Var. i (Framepointer - 4) speichern
        movel a6@(-4),d0    # Variable i in Register d0 laden (Rueckgabewert an main)
        jra L2              # jump auf return-code

L2:                              # return-code fuer f1
        movel a6@(-12),d2   # Register d2 restaurieren (war fuer Var. r auf Stack gerettet worden)
        movel a6@(-8),d3    # Register d3 restaurieren (war fuer Var. s auf Stack gerettet worden)
        unlk a6             # geretten Framepointer von main restaurieren, Stackpointer = Framepointer
        rts                 # return - Ruecksprungadresse von Stack nehmen - Stackpointer inkrementieren
```


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998 F-Prozesse.doc 1998-01-14 08.33
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stackaufbau (6)

■ Motorola 68000-Assembler — Funktion f2( )

```
f2(z)
{
        int m;
        register r;
        r = z;
        m = 100;
        return(z+1);
}
```

```
.even
.globl _f2

_f2:
         link a6,#-4        # Framepointer der Funktion f1 retten und Stackbereich fuer
                            lokale Var. reservieren (4 Byte)
        movel d2,sp@-       # Register d2 retten - auf Stack legen und Stackptr. dekr.
        movel a6@(8),d1     # Parameter z (liegt auf Addr. Framepointer +8) an Var. r (Register d1) zuweisen
        moveq #100,d2       # 100 in Hilfsregister d2 schreiben
        movel d2,a6@(-4     # m (Framepointer - 4) = 100 (aus Register d2)
        movel a6@(8),d0     # Uebergabeparameter z in Register d0 (Rueckgabewert) kopieren
        addql #1,d0         # 1 auf Rueckgabewert addieren
        jra L3              # jump auf return-code

L3:                         # return-code fuer f1
        movel a6@(-8),d2    # Register d2 restaurieren
        unlk a6             # geretten Framepointer von f1 restaurieren, Stackpointer = Framepointer
        rts                 # return - Ruecksprungadresse von Stack nehmen - Stackpointer inkrementieren
```


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998 F-Prozesse.doc 1998-01-14 08.33
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.3 UNIX — Verwaltungsstrukturen des Systemkerns

1 Verwaltungsdaten pro Prozeß

■ **User area**

◆ Daten zur Verwaltung des Programmablaufs durch den Systemkern, z. B.

- Segmenttabelle bzw. Verweis auf Seiten-Kachel-Tabelle
- Verweis auf Eintrag in die Prozeßtabelle
- Statistik-Daten (Laufzeiten, ...)
- aktuelle Directory *(current working directory)*
- aktuelle Root
- user file descriptor table
- Adressen der *signal-handler*-Funktionen
- Bereich zum Sichern des Prozeßzustands (Register) bei Prozeßwechsel
- Puffer für Systemaufruf-Parameter und -Rückgabewerte

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998 F-Prozesse.doc 1998-01-14 08.33
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.