Fakultät für Informatik

MNM TEAM
Munich Network Management Team

Institut für Informatik

LMU

Systempraktikum im Wintersemester 2009/2010 (LMU):
Vorlesung vom 10.12. – Foliensatz 6

Das Signalkonzept (T)
Signale und Signalbehandlung (P)

Thomas Schaaf, Nils gentschen Felde

Termine (vorbehaltlich)

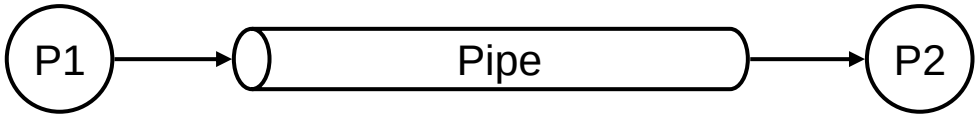
MNM TEAM

- LRZ-Führung:
  - 17.12.2009, 14:00 Uhr
  - Treffpunkt: LRZ-Foyer (Haupteingang, beim Nutzersekretariat)
- Theorieveranstaltungen:
  - 07.01.2010: Rechnernetze / Verteilte Systeme / Sockets
  - 14.01.2010: Wiederholung / Fragen
- Multiple-Choice Tests:
  - 5. Test: 07.01.2010
  - 6. Test: 14.01.2010
- Klausur:
  - Voraussichtlich 28.01.2010
  - Dauer: 120 Minuten
- Projektaufgaben:
  - Ausgabe:
    - Blatt 6: 17.12.2009
    - Blatt 7: hoffentlich vor Weihnachten ;-)
  - Abgaben: 15.01. bzw. 22.01.2010

Interprozesskommunikation

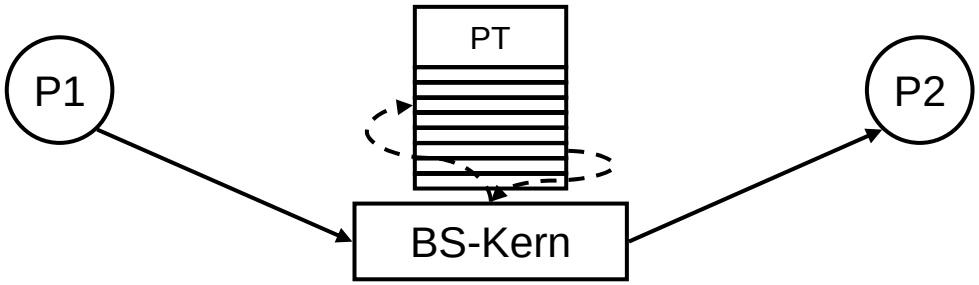
MNM TEAM

- Überblick: Interprozesskommunikation (stark abstrahiert)
  - Pipes und FIFOs:



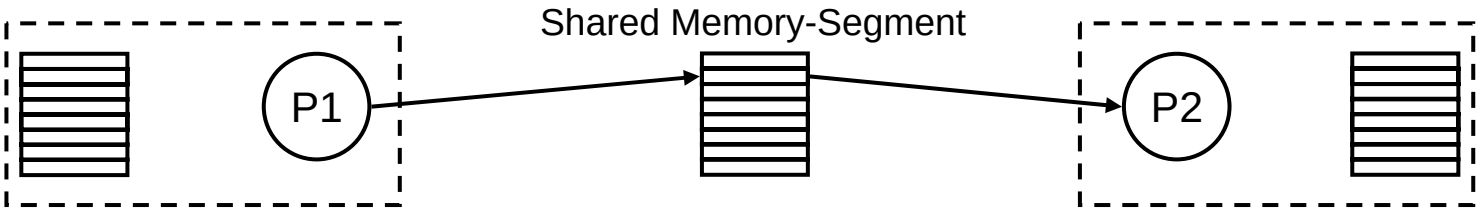
✓

  - Signale:



Heute

  - Speicherbasierte Kommunikation (Shared Memory-Konzept):



✓

  - Netzbasierte Kommunikation (TCP/IP):

Das Signalkonzept

MNM TEAM

- Was sind Signale?
  - "Asynchrone Ereignisse"
  - Tatsächliche Verarbeitung logisch synchron
  - Mögliche Auslöser eines Signals:
    - Anderer Prozess
    - Betriebssystem-Kern (Kernel)
    - Benutzer
  - Informationswert eines Signals:
    - Nur numerischer Wert (systemabhängig!)
    - Besser: Verwendung von Signal-Namen (systemunabhängig)
    - Kein Austausch größerer Informationsmengen über Signale
    - Aber: Form der Interprozesskommunikation

Das Signalkonzept

MNM TEAM

- Möglichkeiten zur Reaktion auf ein Signal (= Prozessseitige Signalbehandlung)
  - Ignorieren
  - Default-Aktionen
  - Bewusstes Reagieren → Signal-Handler

- Systemseitige Signalbehandlung/Signalverarbeitung
  1. Signal(ereignis) tritt ein
  2. Hinterlegung des Signals im Prozesstabelleneintrag des empfangenden Prozesses
     → pending signal
  3. Signal wird im Kontext des Prozesses verarbeitet, wenn der Prozess wieder die CPU-Kontrolle erhält (Scheduling)

Das Signalkonzept

MNM TEAM

- Terminologie zu Signalen
  - Generieren eines Signals
  - Zustellen eines Signals: Starten der für ein Signal eingerichteten Aktion
  - Pending Signal (hängendes Signal):
    Zeit zwischen Generierung und Zustellung
  - Blockieren/Sperren/Maskieren eines Signals:
    temporäres Verhindern der (Default-)Signalbehandlung
  - Signalmaske: enthält die Signale, die von einem Prozess temporär blockiert/gesperrt/maskiert werden
  - Ignorieren eines Signals: generelle Nichtbehandlung eines Signals

- Bitfelder (Vektoren) zur Signalverwaltung im Prozesstabelleneintrag
  - signal-Bitmap: alle pending Signale
  - blocked-Bitmap: alle blockierten Signale
  - sigignore-Bitmap: alle ignorierten Signale
  - sigcatch-Bitmap: alle behandelten Signale

Das Signalkonzept

MNM TEAM

- Beispiel: Signal-Kontrollstruktur im PT-Eintrag
  - Signal 3 hängt (pending), Signale 1, 2, 4, 5 und 6 werden blockiert, Signal 2 wird ignoriert, Signal 1, 3 und 5 werden behandelt

#	signal	blocked	sigignore	sigcatch
1	0	1	0	1
2	0	1	1	0
3	1	0	0	1
4	0	1	0	0
5	0	1	0	1
6	0	1	0	0
7	0	0	0	0
...				
30	0	0	0	0

Signale und Signalbehandlung

MNM TEAM

- Ausgewählte Signale

SIGCHLD	Wird an den Elternprozess geschickt, wenn einer seiner Kindprozesse terminiert
SIGCONT	Wird an einen angehaltenen Prozess gesendet, wenn dieser seine Ausführung fortsetzen soll
SIGFPE	Wird bei einem arithmetischen Fehler (z.B. Division durch 0) geschickt
SIGINT	**Wird allen aktiven "Vordergrundprozessen" geschickt, wenn die Unterbrechungstaste/-tastenkombination (i.d.R. STRG + C) gedrückt wird**
SIGKILL	Beendet einen Prozess
SIGPIPE	Wird einem Prozess geschickt, der versucht, in eine Pipe zu schreiben, zu der es keinen offenen Lese-Filedeskriptor gibt
SIGSTOP	Hält einen laufenden Prozess an

Signale und Signalbehandlung

MNM TEAM

- Zusammenhang zwischen Signalen und Kindprozessen
  - Kindprozess erbt Signal-Handler des Elternprozesses
  - Zombie-Prozesse ohne Verwendung von `wait()`/`waitpid()` verhindern: Signal SIGCHLD im Elternprozess ignorieren

- Das "alte" Signalkonzept – Systemfunktionen
  - `signal()`: Verarbeiten von/Reagieren auf Signale
  - `kill()`: Senden von Signalen an andere Prozesse
  - `raise()`: Senden von Signalen an den eigenen Prozess
  - Bemerkung:

```
raise(signr);
```

ist identisch mit

```
kill(getpid(), signr);
```

Signale und Signalbehandlung

MNM TEAM

- Beispiel: SIGINT – Varianten der Default-Behandlung
  - signal1.c:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
   while(1) {
      printf("+");
   }
   //wird nie erreicht
   return EXIT_SUCCESS;
}
```

  - signal2.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>

void mysighandler() {
   exit(-1);
}

int main(int argc, char *argv[]) {
   signal(SIGINT, mysighandler);
   while(1) {
      printf("+");
   }
   //wird nie erreicht
   return EXIT_SUCCESS;
}
```

  - signal3.c:

```
#include <...>

int main(int argc, char *argv[]) {
   signal(SIGINT, SIG_DFL);
   while(1) {
      printf("+");
   }
   //wird nie erreicht
   return EXIT_SUCCESS;
}
```

Signale und Signalbehandlung

MNM TEAM

- Beispiel: Ignorieren von SIGINT
  - signal4.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>

int main(int argc, char *argv[]) {
   signal(SIGINT, SIG_IGN);
   while(1) {
      printf("+");
   }
   //wird nie erreicht
   return EXIT_SUCCESS;
}
```

Signale und Signalbehandlung

MNM TEAM

- Beispiel: Eigener Signal-Handler für SIGINT
  - signal5.c:

```
#include <...>

int versuche = 5;

void mysighandler() {
   if(versuche > 1) {
      printf("Nochmal!\n");
      versuche--;
   } else {
      exit(-1);
   }
}

int main(int argc, char *argv[]) {
   signal(SIGINT, mysighandler);
   while(1) {
      //beliebige Berechnungen
   }
   //wird nie erreicht
   return EXIT_SUCCESS;
}
```

Signale und Signalbehandlung

MNM TEAM

- Das "neue" Signalkonzept
  - Bisher betrachtet: das "alte" Signalkonzept
  - Probleme des alten Signalkonzepts:
    - Erfragen des aktuellen Signal-Handlers ohne Änderung nicht möglich
    - Zeitspanne zwischen Auftreten eines Signals und Aufruf der `signal()`-Funktion
    - Endlosschleifen beim Warten auf das Eintreten von Signalen möglich
  - Das neue Signalkonzept (POSIX):
    - Signalmengen
    - Einrichten und Erfragen von Signal-Handlern mit `sigaction()`

Signale und Signalbehandlung

MNM TEAM

- Signalmengen
  - Eigener Datentyp `sigset_t`
  - Funktionen zur Manipulation von Signalmengen:
    - Initialisierung:
      - `int sigemptyset(sigset_t *set)`: entfernt alle Signale aus der Signalmenge, auf die `set` zeigt
      - `int sigfillset(sigset_t *set)`: fügt alle vorhandenen Signale zur Signalmenge hinzu
    - Hinzufügen und Löchen einzelner Signale:
      - `int sigaddset(sigset_t *set, int signr)`: (einzelnes) Signal mit der Nummer `signr` zur Signalmenge hinzufügen
      - `int sigdelset(sigset_t *set, int signr)`: Signal entfernen
    - Prüfen der Mitgliedschaft eines Signals in der Signalmenge:
      - `int sigismember(const sigset_t *set, int signr)`
    - Rückgabewerte dieser Funktionen:
      - Die ersten vier: 0 bei Erfolg, -1 im Fehlerfall
      - Die Funktion `sigismember()`: 1 für TRUE, 0 für FALSE

Signale und Signalbehandlung

MNM TEAM

- Beispiel: Umgang mit Signalmengen
  - signal6.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

int main( int argc, char* argv[]) {
   int result;
   sigset_t signalSet1;

   result = sigfillset(&signalSet1);
   printf("result nach sigfillset(): %i\n", result);

   result = sigismember(&signalSet1, SIGINT);
   printf("result nach 1. sigismember(): %i\n", result);

   result = sigemptyset(&signalSet1);
   printf("result nach sigemptyset(): %i\n", result);

   result = sigismember(&signalSet1, SIGINT);
   printf("result nach 2. sigismember(): %i\n", result);
```

Signale und Signalbehandlung

MNM TEAM

- Beispiel: Umgang mit Signalmengen (Forts.)
  - signal6.c:

```
   result = sigaddset(&signalSet1, SIGINT);
   printf("result nach sigaddset(): %i\n", result);

   result = sigismember(&signalSet1, SIGINT);
   printf("result nach 3. sigismember(): %i\n", result);

   return EXIT_SUCCESS;
}
```

  - Ausführung und Ausgabe:

```
result nach sigfillset():
result nach 1. sigismember():
result nach sigemptyset():
result nach 2. sigismember():
result nach sigaddset():
result nach 3. sigismember():
```

Signale und Signalbehandlung

MNM TEAM

- Die Struktur `sigaction`
  - Definition:

```
struct sigaction {
   void (*sa_handler)();
   sigset_t sa_mask;
   int sa_flags;
};
```

  - Erläuterungen:
    - `sa_handler`: Adresse des Signal-Handlers **oder** SIG_IGN **oder** SIG_DFL
    - `sa_mask`: (zusätzlich) zu blockierende/sperrende Signale während der Signalbehandlung
    - `sa_flags`: Signaloptionen, z.B.:
      - SA_SIGINFO
      - SA_NOCLDSTOP
      - SA_RESTART
      - …

Signale und Signalbehandlung

MNM TEAM

- Signal-Handler mit der Funktion `sigaction()`
  - Funktionssignatur von `sigaction`:

```
int sigaction(int signr, const struct sigaction
 *neuerHandler, struct sigaction *alterHandler)
```

  - Argumente:
    - 1. Argument: Die Nummer des Signals, zu dem ein Signal-Handler eingerichtet oder abgefragt werden soll
    - 2. Argument: Wenn `neuerHandler` **kein** NULL-Zeiger ist, wird für das Signal ein neuer Signal-Handler eingerichtet
    - 3. Argument: Wenn `alterHandler` **kein** NULL-Zeiger ist, liefert die Funktion den Signal-Handler, der momentan für das Signal eingerichtet ist
  - Rückgabewert: 0 bei Erfolg, -1 im Fehlerfall

Signale und Signalbehandlung

MNM TEAM

- Beispiel: Signal-Handler für SIGINT mit `sigaction()`
  - signal7.c (gleiches Verhalten wie signal5.c):

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

int versuche = 5;

void mySigHandler(int theSignal, siginfo_t *sigInfo, void *ptr) {
   switch(theSignal) {
      case SIGINT:
         if(versuche > 1) {
            printf("Nochmal!\n");
            versuche--;
         } else {
            exit(-1);
         }
         break;
   }
}
```

Signale und Signalbehandlung

MNM TEAM

- Beispiel: Signal-Handler für SIGINT mit `sigaction()` (Forts.)
  - signal7.c (gleiches Verhalten wie signal5.c):

```
int main( int argc, char* argv[]) {
   sigset_t signalSet1;
   struct sigaction sigActions;

   sigfillset(&signalSet1);

   sigActions.sa_sigaction = mySigHandler;
   sigActions.sa_mask = signalSet1;
   sigActions.sa_flags = SA_SIGINFO;

   sigaction(SIGINT, &sigActions, NULL);
   while(1);
}
```

Zusammenfassung

MNM TEAM

- Signalkonzept
- Prozessseitige Signalbehandlung
- Systemseitige Signalverarbeitung
- Terminologie zu Signalen
- Bitfelder zur Signalverwaltung

- Übersicht: Ausgewählte Signale
- Signale und Kindprozesse
- Das "alte" Signalkonzept
- Signalbehandlung
- Das "neue" Signalkonzept
- Signalmengen und `sigaction`