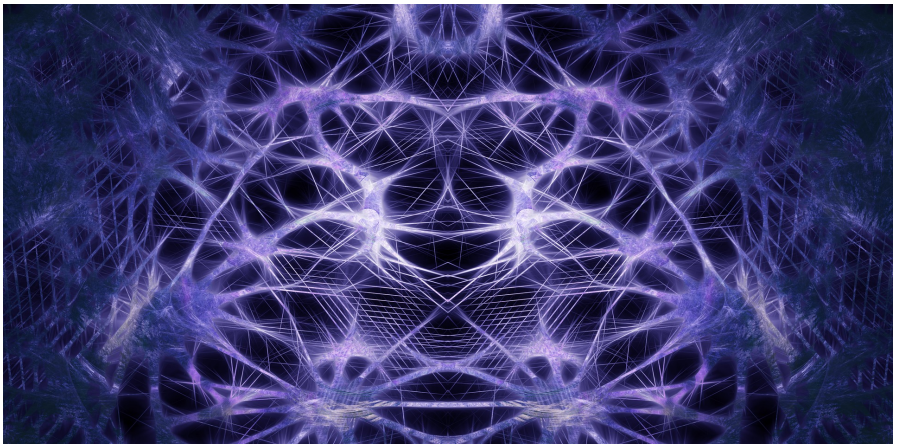

Übung 01

Nichtsequentielle und Verteilte Programmierung

Sebastian Faase, Lutz Schäfer

Tutorium Christopher Filsinger (Mo)

Freie Universität Berlin

17. April 2019

I Korrektheit (10 Punkte)

Aufgabenstellung

Beschreiben Sie kurz (stichpuntartig) was notwendig ist, damit Ihre Programme korrekt ausgeführt werden.

Lösung[1]

- Korrekte Umsetzung der Befehle und Funktionen
  - Compiler/Interpreter, HW
- Korrekte Ausführung der Menge der Befehle
  - Sequentielle Abarbeitung der Programme als separate Prozesse
  - Programmiermodell und Maschinenmodell (Ausführungsmodell) entsprechen einander
- Überprüfen mit
  - Hoare-Kalkül
  - Testen

II Programmierung in C (12 Punkte)

Aufgabenstellung

Richten Sie sich eine Programmierumgebung für die Programmierung in C ein. Beschreiben Sie Ihre Umgebung kurz (stichpunktartig) und testen Sie die Beispiele aus der Vorlesung. Geben Sie zudem an, welche Möglichkeiten der Fehlersuche und -behandlung Ihnen zur Verfügung stehen.

Lösung

Beschreibung einer der Programmierumgebungen von Sebastian Faase

Im Folgenden gehe ich nicht weiter auf die verwendete Hardware ein, als dass es sich um eine x86_64 Architektur handelt.

- OS: Funtoo Linux
- Compiler: die Funtoo eigene Binary von der GCC (gcc (Funtoo 7.4.1-r6) 7.4.1 20181207, Version ändert sich allerdings häufig)
- Editoren: Neovim[2], Atom[3], VSCode[4]

Testläufe der Beispiele aus der Vorlesung

Anmerkung: Da nicht festgelegt ist auf welche Folien sich diese Aufgabe bezieht, gehen wir im Weiteren davon aus, dass es sich um die Folien eins bis drei handelt.

Listing 1: Beispiel 1

```
p7@skor: ~ $ nvim /tmp/test.c

int main (void)
{
  int a = 0;
  int b = 0;
  a = 2;
  b = 3;
  a = a * b;
  return a;
}
:x

p7@skor: ~ $ gcc -std=c11 /tmp/test.c -o /tmp/test
p7@skor: ~ $ cd /tmp
p7@skor: /tmp $ ./test
p7@skor: /tmp $ echo $?
6
p7@skor: /tmp $ -
p7@skor: ~ $
```

Listing 2: Beispiel 2

```
p7@skor: ~ $ nvim /tmp/test.c
```

```
int main (void)
{
  int a = 0;
  int b = 0;
  b = 3;
  while (b > 0)
  {
    a += 2;
    b--;
  }
  return a;
}
:x

p7@skor: ~ $ gcc -std=c11 /tmp/test.c -o /tmp/test
p7@skor: ~ $ cd /tmp
p7@skor: /tmp $ ./test
p7@skor: /tmp $ echo $?
6
p7@skor: /tmp $ -
p7@skor: ~ $
```

Listing 3: Beispiel 3

```
p7@skor : ~ $ nvim /tmp/test.c

#include <stdlib.h>
#include <time.h>
#include <stdio.h>
int main (void)
{
  int random_nr = 0;
  srand ((unsigned) time (NULL));
  random_nr = rand ();
  if (random_nr < RAND_MAX / 2)
  printf ("Die Zufallszahl liegt in der unteren Haelfte! \n");
  else
  printf ("Die Zufallszahl liegt in der oberen Haelfte! \n");
  return 0;
}
:x

p7@skor : ~ $ gcc -std=c11 /tmp/test.c -o /tmp/test
p7@skor : ~ $ cd /tmp
p7@skor : /tmp $ ./test
p7@skor : /tmp $ echo $?
6
p7@skor : /tmp $ -
p7@skor : ~ $ nvim /tmp/test.c
```

```
p7@skor : ~ $ cd /tmp
p7@skor : /tmp $ gcc -std=c11 /tmp/test.c -o /tmp/test
p7@skor : /tmp $ ./test
Die Zufallszahl liegt in der oberen Haelfte!
p7@skor : /tmp $ ./test
Die Zufallszahl liegt in der oberen Haelfte!
p7@skor : /tmp $ ./test
Die Zufallszahl liegt in der oberen Haelfte!
p7@skor : /tmp $ ./test
Die Zufallszahl liegt in der unteren Haelfte!
p7@skor : /tmp $ ./test
Die Zufallszahl liegt in der unteren Haelfte!
p7@skor : /tmp $ ./test
Die Zufallszahl liegt in der unteren Haelfte!
p7@skor : /tmp $ ./test
Die Zufallszahl liegt in der unteren Haelfte!
p7@skor : /tmp $ ./test
Die Zufallszahl liegt in der unteren Haelfte!
p7@skor : /tmp $ ./test
Die Zufallszahl liegt in der oberen Haelfte!
p7@skor : /tmp $ ./test
Die Zufallszahl liegt in der oberen Haelfte!
p7@skor : /tmp $ -
p7@skor : ~ $
```

Listing 4: Beispiel 4

```
$ nvim test.c
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>     // minimal correction here to avoid warning:
#include <sys/types.h>  // implicit declaration of function 'fork'
#include <sys/wait.h>
int main(void)
{
  int status; pid_t pid;
  pid = fork();
  if(pid == 0)
  {
    printf("Child process running...\n"); // Do something...
    printf("Child process done.\n"); exit(123);
  }
  else if(pid > 0)
  {
    printf("Parent process, waiting for child %d...\n", pid);
    pid = wait(&status);
    printf("Child process %d terminated, status %d.\n", pid, \
      WEXITSTATUS(status));
    exit(EXIT_SUCCESS);
```

```
  }
  else
  {
    printf("fork() failed\n"); exit(EXIT_FAILURE);
  }
}

$ gcc -std=c11 test.c -o test
$ ./test
Parent process, waiting for child 23824...
Child process running...
Child process done.
Child process 23824 terminated, status 123.
$ ./test
Parent process, waiting for child 23832...
Child process running...
Child process done.
Child process 23832 terminated, status 123.
$
```

Listing 5: Beispiel 5

```
$ nvim test.c

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>     // minimal correction here to avoid warning:
#include <sys/types.h>  // implicit declaration of function 'fork'
#include <sys/wait.h>
int main (void)
{
  int status = 0;
  pid_t pid;
  int a, b = 0;
  pid = fork ();
  if (pid == 0)
  {
    // child process is doing
    // something...
    a = 2;
    b = 3;
    a = a * b;
    printf ("result of child process: %d \n", a);
    exit (123);
  }
  else if (pid > 0)
  {
    // parent process is doing
    // something and is waiting for child ...
    a = 2;
```

```
    b = 3;
    a = a * b;
    pid = wait (&status);
    printf ("␣result␣of␣parent␣process:␣%d␣\n", a);
    printf("Child␣process␣%d␣terminated,␣status␣%d.\n", pid, \
      WEXITSTATUS(status));
    exit (EXIT_SUCCESS);
  }
  else
  {
    printf ("fork()␣failed\n");
    exit(EXIT_FAILURE);
  }
  return 0;
}
:x

$ gcc -std=c11 test.c -o test
$ ./test
result of child process: 6
result of parent process: 6
Child process 26514 terminated, status 123.
$ ./test
result of child process: 6
result of parent process: 6
Child process 26521 terminated, status 123.
$ ./test
result of child process: 6
result of parent process: 6
Child process 26529 terminated, status 123.
```

Möglichkeiten der Fehlersuche und -behandlung

- Debugger
- Modelchecker
- Code Reviews
- manuelles Testen
- Google
- Stackoverflow
- Rubber Ducking
- Tutoren
- KomilitonX
- Lernraum

III Performance (8 Punkte)

Aufgabenstellung

Beschreiben Sie mindestens eine sinnvolle Anwendung, die durch eine Ausführung mit mehreren Prozessen einen Geschwindigkeitszugewinn erfahren kann. Geben Sie an, an welcher Stelle der Programmausführung der Geschwindigkeitsgewinn erzielt wird.

Lösung

Anwendung 1 - Mergesort

Anmerkung: der Mergesort Algorithmus ist als bekannt vorausgesetzt. Das Sortieren der Teillisten kann zur Effizienzsteigerung parallelisiert werden.

Anwendung 2 - Compiler

Wann immer Anweisungen keinen logischen Zusammenhang besitzen, kann ein Compiler sie parallel bearbeiten und davon profitieren.

Literatur

[1] B. Linnert, "Konzepte der nichtsequentiellen und verteilten programmierung - nebenlÄufigkeit," *ALP IV: Konzepte der nichtsequentiellen und verteilten Programmierung SoSe 2019*, vol. 1, p. 27, 2019.

[2] *Neovim*. `https://neovim.io/`.

[3] *Atom*. `https://atom.io/`.

[4] *VSCode*. `https://code.visualstudio.com/`.