Hochschule Karlsruhe
University of Applied Sciences

Fakultät für **Informatik**
und Wirtschaftsinformatik

Prof. Dr. Carsten Sinz
Lehrgebiet Software-Entwicklung

Labor Betriebssysteme: Aufgabe 1
Prozesse

Hinweise:

- Bearbeiten Sie die Teilaufgaben in Teams und laden Sie Ihre Antworten in ILIAS hoch. Zusätzlich müssen Sie Ihre Abgaben auch im Labor präsentieren. Dabei müssen alle Teammitglieder anwesend sein.
- In ILIAS abzugeben sind Programme zu Teilaufgaben 2c) und 3c) sowie Antworten zu den Fragen der anderen Teilaufgaben.
- Der Aufgabentext geht davon aus, dass Sie CLion als C/C++-IDE verwenden. Sollten Sie die HKA-lizenzierte Version von CLion nicht herunterladen können, verwenden Sie bitte die 30-Tage-Evaluierungslizenz von CLion.

In diesem Projekt soll ein bestehender, in C++ programmierter Ray-Tracer parallelisiert werden, um die Rechenleistung von heute weitverbreiteten Mehrkernprozessoren auszunutzen.

Teilaufgabe 1 (Vorbereitung):

Wir wollen in diesem Projekt den Raytracer des Buches "Ray Tracing in One Weekend" von Peter Shirley verwenden. Dieser Raytracer, ebenso wie das zugehörige E-Book, ist auf GitHub unter `https://github.com/RayTracing/raytracing.github.io` erhältlich.

Anstatt diesen per `git clone` herunterzuladen, wollen wir ihn direkt in CLion importieren. Wählen Sie dazu in CLion `File` > `New` > `Project From Version Control` und geben dort unter URL die oben genannte URL ein.

Gehen Sie dann zu `File` > `Settings` und fügen Sie unter `Build,Execution,Deployment` > `CMake` > `Profiles` das Profil `Release` hinzu und wählen im Drop-Down-Menü zu den Run-Konfigurationen (rechts oben in der IDE) die Konfiguration `inOneWeekend` | `Release` aus.

Bauen Sie das Projekt und führen es probehalber aus. Es sollten auf der Run-Konsole Zahlenkolonnen bestehend aus drei Zahlen pro Zeile ausgegeben werden.

Öffnen Sie ein Terminal, wechseln in das Verzeichnis `cmake-build-release` des Projekts und führen dort das Kommando

```
./inOneWeekend >image.ppm
```

aus. Dieses erzeugt ein Bild im `ppm`-Format (NetPBM-P3, siehe auch z.B. `https://en.wikipedia.org/wiki/Netpbm`), das Sie mit einem passenden Anzeigeprogramm betrachten können.

Teilaufgabe 2 (Prozesse in Unix und der `fork()` System Call):

Der `fork()` System Call erzeugt einen neuen Prozess (*child process*), der eine fast exakte Kopie des aufrufenden Prozesses (*parent process*) ist. Beide Prozesse setzen ihre Ausführung direkt nach dem `fork()`-Aufruf fort. Unterschiedlich ist z.B. die Prozess-ID (*pid*) der beiden Prozesse.

a) Betrachten Sie das folgende C-Programm:

```c
#include <unistd.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  fork();

  printf("Hello!\n");
  sleep(1);
}
```

Speichern Sie es in einer Datei `simple-fork.c` ab und übersetzen Sie es oder legen Sie ein neues C-Projekt in CLion an.

Was ist die Ausgabe des Programms und wieso?

Modifizieren Sie das Programm so, dass der `fork()`-Aufruf drei Mal direkt hintereinander im Programm steht.

Was ist nun die Ausgabe des Programms? Erklären Sie.

b) Betrachten Sie nun das folgende C-Programm `fork-example.c`:

```c
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  // fork a process
  pid_t pid = fork();

  if (pid == 0) {

    // child process
    pid = getpid();
    pid_t ppid = getppid();
    printf("This is child process (pid=%d) with parent %d.\n", pid, ppid);
    // child sleeps for 5 secods
    sleep(5);
    exit(0);

  } else if (pid > 0) {

    // parent process
    pid = getpid();
    printf("This is the parent process with process id %d.\n", pid);
    // wait for child to finish
    pid = wait(NULL);
    if (pid == -1) {
      printf("Wait failed.\n");
      exit(1);
```

```
32      }
33      printf("Child %d finished.\n", pid);
34      exit(0);
35
36    } else {
37
38      printf("Fork failed.\n");
39      exit(1);
40
41    }
42  }
```

Legen Sie wiederum ein neues C-Projekt in CLion für dieses Programm an und führen Sie es aus.

Wie wird die Ausführung unterschiedlichen Programmcodes im Kind- und Elternprozess erreicht?

Was bewerkstelligen die System Calls `getpid()`, `getppid()` und `wait()`? Was gibt das Argument und was der Rückgabewert von `wait()` an?

c) Erstellen Sie ein Programm, das genau n Kindprozesse mittels `fork()` anlegt. Jeder Kindprozess soll ausgeben: "This is child process i.", wobei i die Nummer des Kindprozesses zwischen 1 und n ist. Nach der Ausgabe soll jeder Kindprozess noch 5 Sekunden warten (mittels `sleep()`) und sich dann beenden. Die Anzahl n der Kindprozesse soll als Kommandozeilenargument angegeben werden. (Sie können hier die Funktion `atoi()` aus der Standardbibliothek verwenden und diese auf `argv[1]` anwenden.)

Der Elternprozess soll auf die Beendigung sämtlicher Kindprozesse warten und sich dann auch mit der Meldung "Parent process finished." beenden.

Teilaufgabe 3 (Parallelisierung Raytracer):

In dieser Teilaufgabe soll der Raytracer aus Teilaufgabe 1 parallelisiert werden.

a) Machen Sie sich mit den relevanten Teilen des Quellcodes des Raytracers vertraut. Dies sind insbesondere die Dateien `camera.h` und `color.h` im Verzeichnis `src/InOneWeekend`.

b) Überlegen Sie sich, wie die Raytracer-Berechnung auf n Prozesse aufgeteilt werden kann.

c) Modifizieren Sie den Quellcode so, dass ein paralleler Raytracer entsteht. Dabei können Sie wie folgt vorgehen:

1. Extrahieren Sie den Programmcode, der eine Zeile des Bildes berechnet, in eine eigene Funktion, z.B. `renderLine()`. Übergeben Sie relevante Werte als Parameter oder mittels globaler Variablen an diese Funktion.
2. Modifizieren Sie den Programmcode so, dass er Farbwerte nicht direkt auf `std::cout` ausgibt, sondern diese stattdessen in ein Array mit Einträgen vom Typ `color` schreibt. Das Array sollte zwischen allen Prozessen geteilt sein und so groß, dass es das gesamte Bild aufnehmen kann.

   Geteilten Speicher für das Bild können Sie mittels der Funktion `mmap()` anfordern. (Lesen Sie sich die man page für diesen system call durch!) Für ein Bild der Höhe

`image_height` und Breite `image_width` ergibt sich Programmcode wie folgt:

```
int image_size_in_bytes = sizeof(color) * image_width * image_height;
color *rendered_image = (color *) mmap(nullptr, image_size_in_bytes,
    PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
```

Um die Speicheradresse eines Pixels (x, y) zu ermitteln, müssen Sie die Koordinaten in eine eindimensionale Anordnung im Array `rendered_image` transformieren.

3. Fügen Sie Programmcode hinzu, der n Kindprozesse anlegt, so dass jeder einen passenden, nicht-überlappenden Teil des Bildes berechnet. Jeder Prozess schreibt seine berechneten Farbwerte in das geteilte Array `rendered_image`.
4. Fügen Sie Programmcode hinzu, der den Elternprozess auf die n Kindprozesse warten lässt und danach das Bild in eine Datei (bzw. auf `std::cout`) schreibt.

d) Testen Sie Ihr Programm und ermitteln Sie die Beschleunigung, die im Vergleich zur sequentiellen Version erreicht werden kann. Wählen Sie dabei für die parallele Version die Anzahl der Kindprozesse so, dass sie der Anzahl der Kerne Ihres Rechners entspricht.

Beobachten Sie die Auslastung Ihres Rechners während der Raytracing-Berechnung. Wieso fällt diese gegen Ende der Berechnung ab und wie könnte dies vermieden werden?