1. Docker installieren

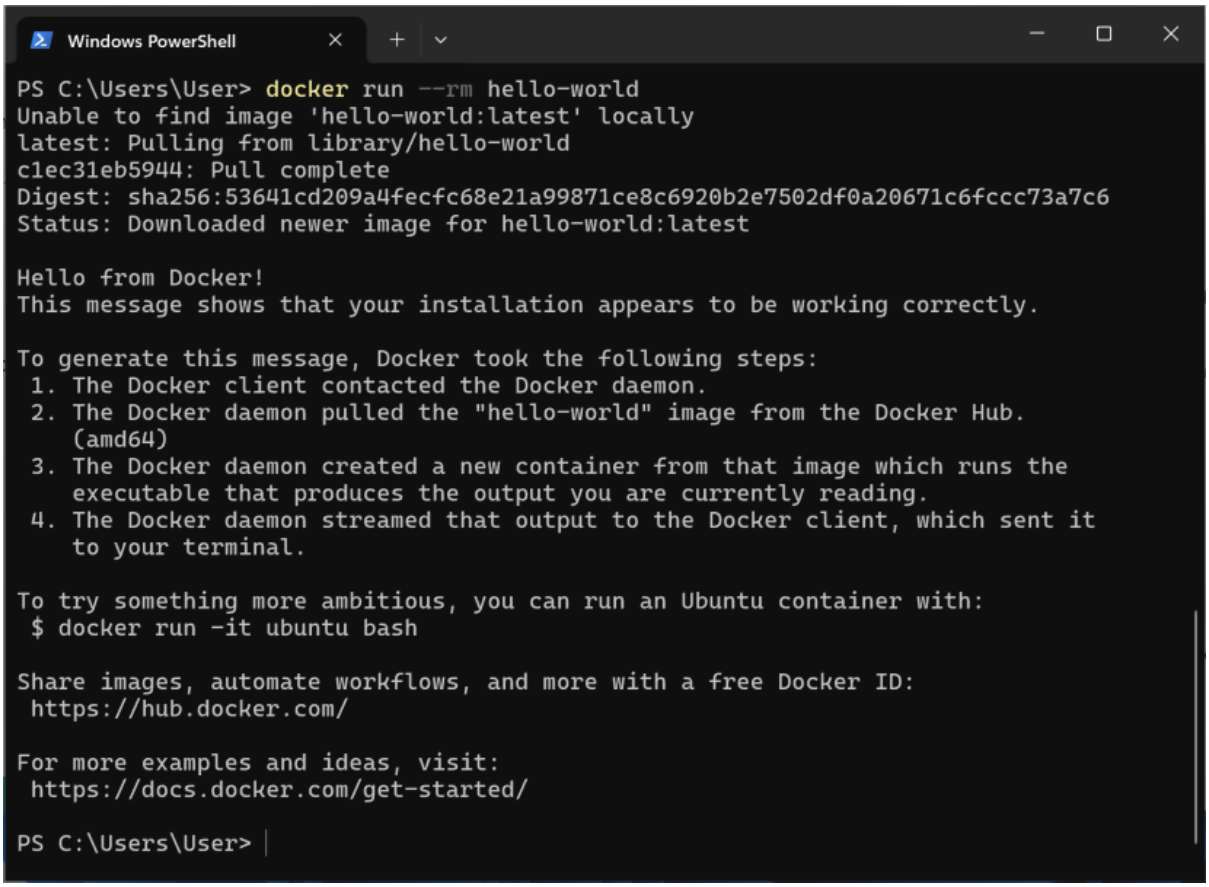
a) Laden Sie von der Seite https://www.docker.com/ das Installationspaket von *Docker Desktop* für Ihr Betriebssystem herunter, spielen Sie die Software ein und starten Sie den Docker Desktop. (Windows-Anwender:innen übernehmen die Vorgabe „Use WSL 2“ und müssen sich nach der Einrichtung ab- und wieder anmelden.)

Falls Sie sich bei Docker registriert (= ein Benutzerkonto angelegt) haben, können Sie sich anmelden; ansonsten wählen Sie die Option „Continue without signing in“.

Öffnen Sie ein Shell-Fenster (Windows: Cmd oder Powershell; MacOS / Linux: beliebiges Terminalprogramm) und testen Sie mit

```
docker run --rm hello-world
```

ob Sie Docker-Container starten können. Docker sollte Dateien herunterladen und dann einige Zeilen Text ausgeben:



b) Für dieses Modul steht ein Ubuntu-Image für Docker zur Verfügung, das Sie wie folgt herunterladen und als Container starten können.

- Unter Windows (PowerShell), Linux oder macOS:

  ```
  docker run -it --rm -v ${PWD}:/realworld hgesser/ubuntu-dev
  ```

- Unter Windows (klassische CMD-Shell):

  ```
  docker run -it --rm -v %cd%:/realworld hgesser/ubuntu-dev
  ```

(Die beiden Kommandos unterscheiden sich nur darin, wie der Zugriff auf das aktuelle Arbeitsverzeichnis ermöglicht wird: Bei der klassischen Windows-Shell läuft das über `%cd%`, bei modernen Shells über `${PWD}`.) Über `-it` erhalten Sie eine Shell, `--rm` löscht den Container nach dem Verlassen.

Beim ersten Aufruf werden mehrere hundert MByte vom Docker Hub, einem Repository für Docker-Images, heruntergeladen; ab dem zweiten Aufruf sollte ohne Verzögerung ein Container starten.

Nach dem Start des Containers erscheint ein Linux-Shell-Prompt (`bash`). Geben Sie hier

```
cd host
```

ein, um in das Arbeitsverzeichnis zu wechseln, aus dem heraus Sie den Docker-Container gestartet haben.

2. Build-Prozess, Erstkontakt mit Themen der Vorlesung

a) Prozesse: Laden Sie mit

```
wget swf.hgesser.de/bs-b1/prakt/bs1-ue01.zip
```

ein Paket mit Quelldateien herunter und entpacken Sie es mit

```
unzip bs1-ue01.zip
```

Wechseln Sie dann mit

```
cd ue01
```

in den Ordner `ue01` und geben Sie dort

```
make
```

ein. Das Tool `make` liest Anweisungen in der Datei `Makefile` (siehe Aufgabe **b**) und übersetzt das C-Programm `baum.c`. Starten Sie die so erzeugte ausführbare Datei `baum` durch Eingabe von

```
make start
```

und schauen Sie sich dann mit `ps` und `pstree` die **Prozessliste** bzw. den **Prozessbaum** an. (Ein Prozess ist ein gestartetes Programm.) Über die Kombination der Funktionsaufrufe von `fork()` und `execl(...)` in den Zeilen 15/16 bzw. 18/19 wird ein sog. **Kindprozess** erzeugt und das als erster Parameter genannte Programm (hier: `eins` bzw. `zwei`) gestartet. (Ein Kindprozess entsteht auch jedesmal, wenn Sie in der Shell ein Kommando eingeben und damit ein Programm auf der Platte starten.)

```
 1  // baum.c
 2  // Betriebssysteme 1, v0.1, 2024-04-08
 3
 4  #include <unistd.h>
 5  #include <stdlib.h>
 6  #include <stdio.h>
 7
 8  int main (int argc, char **argv) {
 9    if (argc != 1) {
10      sleep (60); exit (0);
11    } else {
12      printf ("baum startet; 60 Sekunden bis Programmende.\n");
13    }
14
15    if (fork() == 0)
16      execl ("./eins", "eins", "--one", "--un", (char  *) NULL);
17
18    if (fork() == 0)
19      execl ("./zwei", "zwei", "--two", "--deux", (char  *) NULL);
20
21    sleep (60); exit (0);
22  }
```

Geben Sie in der Shell

```
man 3 sleep
```

und

```
man 3 exit
```

ein, um nachzulesen, was die Funktionen `sleep` und `exit` tun. Den dadurch gestarteten Dateibetrachter verlassen Sie mit [Q] (also mit Eingabe eines kleinen „q“).

b) In der Datei `Makefile` sind verschiedene Aufgaben (Ziele, engl. **targets**) definiert:

```
1     all: baum
2
3     baum: baum.c
4             gcc -o baum baum.c
5             cp baum eins
6             cp baum zwei
7
8     start:
9             @./baum &
10            @echo
11
12    stop:
13            killall -q -9 baum eins zwei; true
```

Die Target-Namen (hier: `all`, `baum`, `start`, `stop`) starten in der ersten Spalte und enden mit einem Doppelpunkt.

Die Zeilen, die eingerückt unter einem Target stehen, enthalten Befehle, die ausgeführt werden, um das Target zu erzeugen. So wird beispielsweise das Target `baum` generiert, indem die Quelldatei `baum.c` mit `gcc -o baum baum.c` übersetzt wird. Danach werden außerdem zwei Kopien `eins` und `zwei` von `baum` erzeugt.

Wichtig ist, dass die eingerückten Zeilen mit einem Tabulatorzeichen (und nicht etwa mehreren Leerzeichen) beginnen.

c) Geben Sie

```
top
```

ein. Sie sehen alle im Linux-Container laufenden Programme (Prozesse), sortiert nach CPU-Auslastung. Sie beenden das `top`-Programm, indem Sie [Q] drücken.

d) Hauptspeicherverwaltung: Geben Sie den Befehl `free` ein. Sie erhalten eine Ausgabe, die wie die folgende aussieht:

```
swf@e65dd9f16e4d:~/host$ free
               total        used        free      shared  buff/cache   available
Mem:         8131072      901984     6145660       17476     1083428     6905996
Swap:        1048572           0     1048572
```

Die letzte Zeile zeigt, dass das System Auslagerungsspeicher auf der Platte („Swap“) besitzt.

Starten Sie den Text-Editor `mcedit` durch Eingabe von

```
mcedit speicher.c
```

geben Sie das folgende kurze C-Programm ein und speichern Sie mit [F2].

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
int main () {
  #define SIZE 1024*1024*64
  void *p = malloc (SIZE); memset (p, 'x', SIZE);
  sleep (20);
}
```

Das Programm fordert mit `malloc()` 64 MByte Speicher dynamisch an (→ `man 3 malloc`) und füllt alle 64 M Bytes mit „x“.

Verlassen Sie den Editor mit [F10] oder [Esc], [0] (Null) und kompilieren Sie das Programm mit

```
gcc -o speicher speicher.c
```

Öffnen Sie ein zweites Terminalfenster (im Hostsystem) und starten Sie darin im laufenden Container eine weitere Shell-Sitzung. Dazu lassen Sie sich erst anzeigen, welche ID der laufende Container besitzt:

```
PS C:\Users\...> docker ps
CONTAINER ID   IMAGE                COMMAND   CREATED          STATUS          ...
2da65aee8fa3   hgesser/ubuntu-dev   "bash"    33 minutes ago   Up 33 minutes   ...
PS C:\Users\...> docker exec -it 2da65aee8fa3 bash
```

(Optionen `-it`: interaktiv, Terminal.) Geben Sie dort

```
free -m -s1
```

ein: Dadurch erhalten Sie jede Sekunde eine aktualisierte Ausgabe, und es werden MByte statt KByte angezeigt. Starten Sie im ersten Terminalfenster durch Eingabe von

```
./speicher
```

das Programm und beobachten Sie, wie sich die Zahlen in der Speicheranzeige ändern. (Wichtig: Vor `speicher` muss `./` stehen; keine Leerzeichen zwischen `.` und `/` und `speicher`.)

Drücken Sie [Strg] + [C], um das Programm `speicher` zu beenden, und beobachten Sie erneut die Änderung im Speicherverbrauch.

3. Container: Zugriff auf Host-Verzeichnis

In Aufgabe **1 b)** haben Sie den Container mit

```
docker run -it --rm -v ${PWD}:/realworld hgesser/ubuntu-dev
```

bzw.

```
docker run -it --rm -v %cd%:/realworld hgesser/ubuntu-dev
```

gestartet. Über die Option `-v` haben Sie dabei das aktuelle Arbeitsverzeichnis der Shell im Container unter dem Pfad `/realworld` verfügbar gemacht. Öffnen Sie einen Dateimanager und navigieren Sie darin zum Arbeitsverzeichnis der Shell. Lassen Sie sich dann im Container mit

```
ls -l /realworld/
```

den Inhalt des Ordners `/realworld` anzeigen. Erzeugen Sie über einen Editor im Arbeitsverzeichnis eine neue Textdatei und schreiben Sie ein paar Zeilen Text hinein. Geben Sie dann erneut `ls -l /realworld` ein und zeigen Sie mit

```
cat /realworld/dateiname.txt
```

den Inhalt an. Sie sehen: Es spielt keine Rolle, ob Sie Dateien über Ihr normales System oder aus dem Container heraus erzeugen; `/realworld` und das Arbeitsverzeichnis enthalten dieselben Dateien.