

BETRIEBSSYSTEME - PRAKTIKUM 1

VORBEREITUNG (ZU HAUSE)

- Lesen Sie die *man pages* zu den Funktionsaufrufen `fork(2)`, `wait(2)`, `execv(2)`, `pthread_create()`, `pthread_join()`
- Versuchen Sie die Beispiele *Prozess Starten* und *Thread Starten* aus der Vorlesung zu kompilieren und führen Sie die Programme aus.
- Starten Sie auf dem Terminal den Befehl *gedit* (es sollte sich ein Texteditor öffnen). Führen Sie den Befehl `ps -ef | less` aus und suchen Sie nach dem Editor. Versuchen Sie die Informationen zu verstehen, die Ihnen der Befehl ausgibt. Nutzen Sie die *man page* von `ps`, um Informationen dazu zu bekommen.
- Lesen Sie die Aufgaben durch und verstehen Sie die Aufgabenstellung.

AUFGABE 1: STARTEN ANDERER PROGRAMME

- a) Laden Sie sich die Datei *start_program.c* aus dem Moodle-Kurs herunter.
- b) Schreiben Sie eine Makefile, mit der das Programm kompiliert werden kann
- c) Implementieren Sie das Programm wie folgt:
 - Das Programm soll einen Kind-Prozess erzeugen und dann ein externes Programm in diesem Kind-Prozess starten.
 - Der Name des zu startenden Programms und seine Argumente werden über die Kommandozeile als Argumente an *start_program* übergeben.
 - D.h. der Aufruf

```
./start_program /bin/ls /
```

soll also das Programm */bin/ls* mit dem Argument */* aufrufen.
- d) Testen Sie Ihr Programm mit den folgenden zwei Aufrufen:
 - `./start_program /bin/ls /`
 - `./start_program ls /`

Diskutieren Sie: Warum funktioniert der erste Aufruf aber der zweite Aufruf nicht?

AUFGABE 2: MULTIPROGRAMMIERUNG MIT PROZESSEN

- a) Laden Sie sich das Programm-Template: *process_creation.c* herunter
- b) Erweitern Sie die Makefile, so dass Sie auch dieses Programm kompilieren kann
- c) Implementieren Sie die main-Funktion im Template wie folgt:
 - Das Programm soll so viele Kind-Prozesse erstellen, wie in der Variable *children* hinterlegt. Diese Kind-Prozesse sollen jeweils parallel drei Zufallszahlen generieren. Implementieren Sie dazu:

- Starten Sie *children* neue Prozesse in einer Schleife
 - Jeder Kind-Prozess soll die Funktion `print_random_number(int)` aufrufen. Als Argument soll die Nummer des gestarteten Prozesses mitgegeben werden
 - Die Kind-Prozesse sollen sich nach Ende der Funktion `print_random_number(int)` beenden
- d) In der Funktion `print_random_number(int)` erzeugt jeder Child-Prozess drei Zufallszahlen. Sind diese für jeden Child-Prozess unterschiedlich? Warum ist das so? Welche Änderung muss man durchführen um das Verhalten zu ändern?
- e) Führen Sie das Programm mehrfach aus und prüfen Sie die Reihenfolge der Ausgaben. Ist diese immer die selbe? Warum ist das so?
- f) Am Ende des Programms werden Sie aufgefordert die Taste *Enter* zu drücken um das Programm zu beenden. Führen Sie vor dem Beenden des Programms den Befehl `ps -ef | less` in einem zweiten Terminal aus. Finden Sie mehr als einen Prozess mit dem Namen des Programms? Warum sehen Sie mehr als einen Prozess bzw. warum sehen Sie nur einen Prozess? Falls Sie mehrere Prozesse sehen, dann beheben Sie das Problem.

AUFGABE 3: MULTIPROGRAMMIERUNG MIT THREADS

- a) Laden Sie sich die Datei `thread_creation.c` aus Moodle herunter
- b) Erweitern Sie die Makefile, so dass Sie auch dieses Programm kompilieren kann
- c) Implementieren Sie in der Main-Funktion wie folgt:
- Starten Sie *children* neue Threads in einer Schleife
 - Jeder der neu gestarteten Threads soll die Funktion `print_random_number()` aufrufen. Als Argument soll ein Zeiger auf ein `struct thread_data` übergeben werden. Die Variable `thread_number` soll die Nummer des gestarteten Threads enthalten. Die Variable `thread_id` soll als erstes Argument zu `pthread_create` verwendet werden.
- d) In der Funktion `print_random_number()` erzeugt jeder Thread drei Zufallszahlen. Sind diese für jeden Thread unterschiedlich? Warum ist das so? Worin liegen die Unterschiede im Vergleich zur vorherigen Aufgabe?
- e) Führen Sie das Programm mehrfach aus und prüfen Sie die Reihenfolge der Ausgaben. Ist diese immer die selbe? Warum ist das so?