

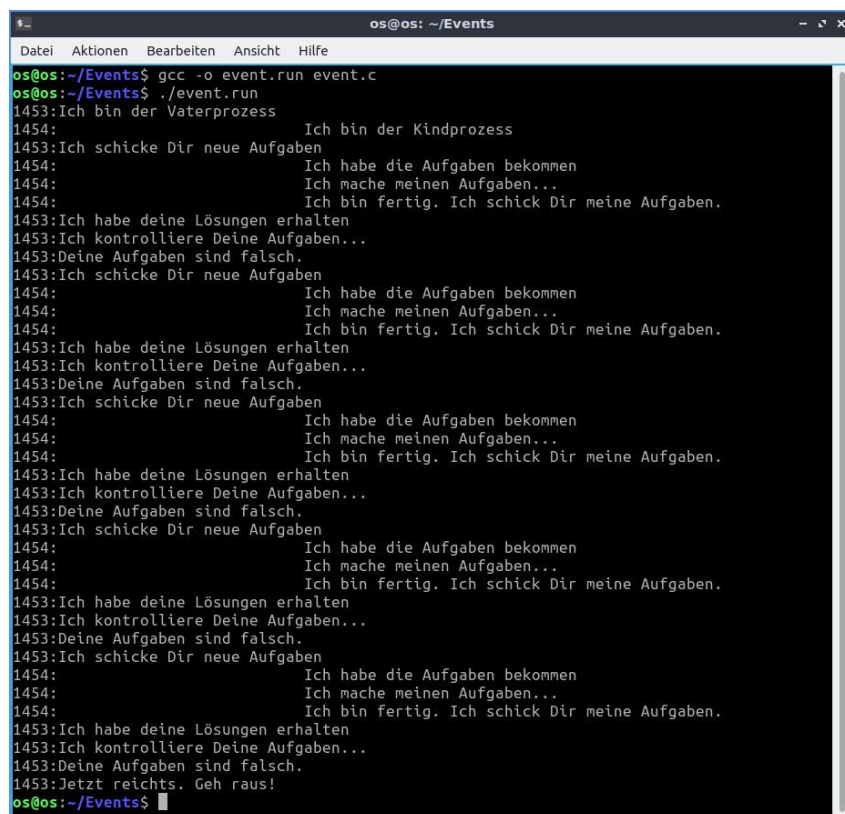
SYNCHRONISATION

Hinweise: Sämtliche Ausführungen gelten für [unixoide Betriebssysteme](#). Lesen Sie bitte auch die zugehörige [Praktikumsdokumentation](#) durch.

1. EVENTS

Die Kontrolle der Hausaufgaben von Kindern ist für Eltern ein schweres Los. Die Eltern geben dem Kind Aufgaben. Das Kind bearbeitet diese und meldet an die Eltern zurück fertig zu sein. Die Eltern kontrollieren die Aufgabe. Leider sind viele Fehler in der Lösung, woraufhin die Eltern dem Kind wieder die Aufgaben geben. Der Kreislauf wiederholt sich. Nach 5 erfolglosen Versuchen, die Hausaufgaben zu kontrollieren, brechen die Eltern ab.

Aufgabe 1.1: Programmieren Sie den geschilderten Sachverhalt mit Hilfe von Signalen. Ihr Programm sollte folgenden Ausgaben erzeugen:



```
os@os: ~/Events
Datei Aktionen Bearbeiten Ansicht Hilfe
os@os:~/Events$ gcc -o event.run event.c
os@os:~/Events$ ./event.run
1453:Ich bin der Vaterprozess
1454:           Ich bin der Kindprozess
1453:Ich schicke Dir neue Aufgaben
1454:           Ich habe die Aufgaben bekommen
1454:           Ich mache meinen Aufgaben...
1454:           Ich bin fertig. Ich schick Dir meine Aufgaben.
1453:Ich habe deine Lösungen erhalten
1453:Ich kontrolliere Deine Aufgaben...
1453:Deine Aufgaben sind falsch.
1453:Ich schicke Dir neue Aufgaben
1454:           Ich habe die Aufgaben bekommen
1454:           Ich mache meinen Aufgaben...
1454:           Ich bin fertig. Ich schick Dir meine Aufgaben.
1453:Ich habe deine Lösungen erhalten
1453:Ich kontrolliere Deine Aufgaben...
1453:Deine Aufgaben sind falsch.
1453:Ich schicke Dir neue Aufgaben
1454:           Ich habe die Aufgaben bekommen
1454:           Ich mache meinen Aufgaben...
1454:           Ich bin fertig. Ich schick Dir meine Aufgaben.
1453:Ich habe deine Lösungen erhalten
1453:Ich kontrolliere Deine Aufgaben...
1453:Deine Aufgaben sind falsch.
1453:Ich schicke Dir neue Aufgaben
1454:           Ich habe die Aufgaben bekommen
1454:           Ich mache meinen Aufgaben...
1454:           Ich bin fertig. Ich schick Dir meine Aufgaben.
1453:Ich habe deine Lösungen erhalten
1453:Ich kontrolliere Deine Aufgaben...
1453:Deine Aufgaben sind falsch.
1453:Ich schicke Dir neue Aufgaben
1454:           Ich habe die Aufgaben bekommen
1454:           Ich mache meinen Aufgaben...
1454:           Ich bin fertig. Ich schick Dir meine Aufgaben.
1453:Ich habe deine Lösungen erhalten
1453:Ich kontrolliere Deine Aufgaben...
1453:Deine Aufgaben sind falsch.
1453:Jetzt reicht's. Geh raus!
os@os:~/Events$
```

Vater und Kind sollten Sie mit zwei unabhängigen Prozessen simulieren (`fork`). Die beiden Prozesse stimmen sich gegenseitig mit den Signalen `SIGUSR1` und `SIGUSR2` ab. Das Kind hat die Aufgabe, die Hausaufgaben zu erledigen. Die Aufgabe des Vaters ist die Korrektur der Aufgaben. Beides geht nicht gleichzeitig. Das Erledigen der Aufgaben und die Korrektur der Aufgaben können Sie mit einer definierten Wartezeit simulieren. Nach fünf Durchläufen sendet der Vaterprozess an den Kindprozess ein `SIGKILL`-Signal und beendet sich selbst.

Hinweis: Um die kommunizierenden Prozesse nicht zu stören, sollten Sie `write` statt `printf` nutzen. Beispiel:

```
char msg[] = „Dies ist die Nachricht“;  
write (STDOUT_FILENO, msg, sizeof(msg));
```

Damit Sie die Signale verwenden können, müssen Sie die Header-Datei `signal.h` einbinden.

2. MUTEX

Im Rahmen des Aufgabenblattes 3 haben Sie ein Programm geschrieben, welches Threads zur Berechnung von Primzahlen verwendet. Dieses Programm ist Ausgangspunkt für die folgenden Aufgaben.

Aufgabe 2.1: Schreiben Sie Ihr Programm zum Zählen von Primzahlen derart um, dass beim Finden einer Primzahl eine global definierte Variable erhöht wird. Lassen Sie Ihr Programm mehrmals laufen. Welchen Effekt können Sie beobachten und worauf führen Sie ihn zurück?

Aufgabe 2.2: Beheben Sie das identifizierte Problem mit einem Mutex. Den Rest des Programms sollten Sie unverändert lassen. Sowohl die Zählmethode aus dem vorangegangenen Aufgabenblatt als auch die neue Zählmethode sollten nun die gleichen Ergebnisse liefern.

Aufgabe 2.3: Nehmen Sie an, Sie würden mit einem gemeinsamen Speicherbereich und unabhängigen Prozessen zur Berechnung der Primzahlen arbeiten. Würden Sie dann immer noch das Konstrukt Mutex verwenden?

3. SEMAPHOR

Aufgrund der pandemischen Sondersituationen mussten 2020 Geschäfte wie zum Beispiel Möbelhäuser schließen. Später durften die Möbelhäuser wieder öffnen, allerdings sehr reglementiert. Die Anzahl der Kunden, die ein Möbelhaus betreten durften, war stark begrenzt. Deswegen kam es immer wieder zu langen Warteschlangen vor den Türen.

Aufgabe 3.1: Bilden Sie diesen Umstand mit einem Programm nach. Jeden Kunden sollten Sie mit einem eigenen Thread oder Prozess abbilden. Zur Vereinfachung gehen wir davon aus, dass eine zufällige Zahl zwischen 0 und 20 an Kunden jede Stunde das Möbelhaus erreicht. Im Möbelhaus dürfen sich maximal 15 Kunden befinden. Die Kunden bleiben eine zufällige Zeit zwischen 1 und 5 Stunden im Möbelhaus. Kunden, die das Möbelhaus nicht betreten dürfen, reihen sich in die Warteschlange vor der Türe ein. Die Öffnungszeiten sollen in dieser Aufgabe nicht weiter relevant sein. Anstelle der Öffnungszeiten existiert eine maximale Zahl von 75 Kunden, die pro Tag das Möbelhaus betreten dürfen.

Nutzen Sie Semaphore zur Nachbildung der geschilderten Umstände. Eine `while`-Schleife startet so lange eine zufällige Zahl (zwischen 0-20) von Threads oder Prozessen, bis die Maximalanzahl an Kunden erreicht ist. Danach wartet das Programm 1 Sekunde (\triangleq 1 reale Stunde) und startet wiederum eine zufällige Anzahl von Threads oder Prozessen. Jeder Thread oder Prozess reiht sich zunächst in die Warteschlange vor der Türe ein. Befindet sich sonst kein Kunde in der Warteschlange, kann der Thread oder Prozess sofort in das Möbelhaus. Sobald der Thread oder Prozess im Möbelhaus ist, wartet er zufällig zwischen 1 und 5 Sekunden (Stunden), verlässt das Möbelhaus und terminiert sich.

Hinweis: Damit Sie Semaphore nutzen können, müssen Sie die Header-Datei `sys/shm.h` einbinden. Nutzen Sie einen Semaphor, um in den Kunden-Threads nebenläufige Zugriffe auf gemeinsam genutzte Variablen zu unterbinden.

Aufgabe 3.2 (optional): Erweitern Sie Ihr Programm zur vorhergehenden Teilaufgabe derart, dass der Umstand simuliert wird, dass neu ankommende Kunden sofort wieder heimgehen, wenn die Anzahl der vor der Tür des Möbelhauses wartenden Kunden 15 überschreiten würde. Erzeugen Sie dazu einen zusätzlichen Semaphor zum Zählen der Kunden/Threads in der Warteschlange. Vermeiden Sie jedoch ein Blockieren auf diesen Semaphor durch Prüfen des Zählerwerts auf 0. Ist der Zählerwert bereits 0, so soll sich der Thread wieder beenden und nicht in der Warteschlange einreihen.