Verteilte Systeme (I6, Bc)

Blatt 01: Arbeitsumgebung Linux, Threads, Synchronsation

Sommersemester 2017

Bearbeitung im Praktikum ab 06.04.2017

Entwicklungsplattform

- Starten des Linux Systems: Beim Booten des Arbeitsplatzrechners Lubuntu (Voreinstellung) auswählen.
- Einloggen mit Standard-Novell-Login
- Zugriff auf Standardlaufwerke der Hochschule: `Anna-Home` bzw. `Anna-Shared` können über Desktop-Icons und Eingabe des Passwortes eingebunden und gelöst werden.
- Nutzung der Entwicklungsumgebung Eclipse: Sie können im Praktikum Eclipse verwenden und erhalten auch immer wieder Hinweise zur Bedienung von Eclipse. Aber: Bei der Abgabe der Lösungen ist Eclipse außen vor. D.h. das Generieren von ausführbaren Programmen und deren Ablauf muss ohne Eclipse erfolgen. Bei der Befragung ist Eclipse zum Editieren zulässig. Das Ausführen der Programme muss allerdings ohne Eclipse erfolgen!
- Das Starten von Eclipse erfolgt über das Desktop-Icon oder von der Konsole aus über `/usr/lib/eclipse/eclipse`
- Alternative zu Eclipse: Konsole, gcc zum Compilieren und zum Editieren "Vim" oder "Geany" oder anderer Editor.

***Hinweis*:**

- Typischer Fehler beim Portieren von Lösungen von einer Linux-Maschnine auf eine andere (z.B. bei Übernahme einer Lösung von zuhause auf die Labor-VM):
  - Bei der Portierung gehen auf der Dateiebene die Ausführungsrechte verloren. Problemlösung: mit `chmod +x dateiname` Ausführungsrechte einrichten.
  - Das ausführbare Programm wird auf dem Zielrechner nicht neu kompiliert sondern direkt ausgeführt. Beim Ausführen des Programms, das auf der ursprünglichen Maschine funktioniert hat, erfolgt eine Fehlermeldung, die ggf. irreführend ist, wie z.B. `Datei oder Verzeichnis nicht gefunden`! Der wahre Grund ist aber, dass der Maschinencode der ursprünglichen Maschine nicht auf dem Zielrechner ausführbar ist. Problemlösung: Programm neu bauen.

Aufgabe 1: C-Programmierung

1.1 "Hello World"-Programm

Schreiben Sie in der Programmiersprache C ein "Hello World"-Programm, übersetzen Sie dieses und führen Sie die binäre Datei aus.

1.2 C-Programm mit zwei Quelldateien sowie #defines und Funktionsprototypen

Schreiben Sie ein C-Programm, welches die Kommandozeilenparameter ausliest und auf der Konsole wieder ausgibt. Außerdem soll die Größe von Variablen ausgewählter Datentypen der Maschine, auf dem das Programm ausgeführt wird, ausgegeben werden.
Datentypen:

```
char     short     int      long      long long
char *   int *     void*
void
```

Ausgabe z.B.:

```
Konsolenprogramm von "Max Mustermann"
sizeof(char)          = 2
sizeof(short)         = 2
.......
1.Uebergabeparameter = "<paramterwert1>"
2.Uebergabeparameter = "<paramterwert2>"
etc.
```

Der Name des Autors des Programms soll dabei als `#define` in eine #include-Datei ausgelagert werden.
Das Ausgeben auf Konsole soll in einer eigenen Funktion (nicht in `main()`) erfolgen. Diese Funktion soll in einer eigenen Datei liegen. Definieren Sie den Prototypen zu dieser Ausgabefunktion und schreiben Sie diesen auch in eine externe #include Datei.

Umgang mit Eclipse:

- Eclipse C/C++ (Button auf dem Desktop) starten
- `/home/benutzername/workspace` bestätigen
- Projekt anlegen mit `File` → `New` → `C Project`.
- `C/C++` → `C Project` → `NEXT` → `Hello World ANSI C Project` auswählen, Name eingeben und abschließen.
- Vorhandene C-Datei im Bereich `src` löschen und über rechte Maustaste darauf `New` → `Source-File` anlegen. In dieser Quelldatei dann den eigenen Code ablegen.
- Angabe von Kommandozeilenparametern: Geben Sie dazu die Argumente in Eclipse über rechte Maustaste auf `Projektnamen` → `Properties` → `Run/Debug Settings` → `Edit` → `Arguments` ein.

Lernziele:

- Entwicklungsplattform kennen lernen
- Kenntnisse in C-Programmierung wieder auffrischen

Aufgabe 2: Gleichzeitige Summenbildung

Geben sie das folgende C-Programm (siehe auch Skript bzw. Zulieferung) ein:

```
#include "pthread.h"                          // oder <pthread.h> ?
#define LENGTH 4
int field [LENGTH][2];                        // Feld von zu addierenden Paaren
int sums [LENGTH];                            // Feld der Paarsummen
void add_op_thread(pthread_addr_t arg)
{
     int * input = (int*) arg;                // Eingabeparameter des Threads
     int result;
     result = input[0] + input[1];
     pthread_exit((pthread_addr_t) result);   //  Rückmeldung Ergebniscode
} // add_op_thread()
main (void)
{    // Erzeugerthread
     pthread_t thread[LENGTH];                // Feld für Threaddatenstrukturen
     int i,result;                            // Einzelergebnis
     for (i=0; i< LENGTH;i++) {
          pthread_create(&thread[i], NULL, add_op_thread,
                  (pthread_addr_t) field[i]);
     }
     for (i=0; i< LENGTH;i++) {
          pthread_join(thread[i], &result);   // Warten auf Threadergebnis
          printf("A: result=%d\n",result);
          sums[i]=result;
          printf("B: result=%d\n",result);
          pthread_detach(thread[i]);}         // Thread zerstören ?!
} // main()
```

2.1 Korrekter Ablauf

Ergänzen bzw. modifizieren Sie dieses Programm, so dass es ohne Warnungen kompilierbar und ohne Absturz ablauffähig ist. Außerdem sollten Konventionen der C-Programmierung eingehalten werden.

2.2 Funktionserweiterung

Ergänzen Sie im zweiten Schritt das Programm um folgende Funktionalität:

- Ausgaben auf Konsole:
  - Zu Programmstart wird das aktuelle Datum und die aktuelle Uhrzeit sowie die aktuelle Systemzeit in Sekunden ausgegeben.
  - Unmittelbar vor Ende wird das aktuelle Datum und die aktuelle Uhrzeit sowie die aktuelle Systemzeit in Sekunden ausgegeben.
- Die Threads erhalten applikationsspezifische, eindeutige Namen in Form von ganzen Zahlen. Beginnend mit `0` werden diese Namen aufsteigend vergeben.
- Jeder Thread gibt am Anfang seiner Threadfunktion eine Threadstartmeldung und seinen Namen, sowie die Meldung aus, wie lange er wartet. Dann wartet er bei Wert des Namens = n für n+1 Sekunden. Am Ende seiner Threadfunktion gibt er eine Endmeldung, sein Berechnungsergebnis und seinen Namen aus.
- Der Hauptthread gibt für jeden Thread dessen Ergebnis aus und berechnet dann das Gesamtergebnis, welches er auch ausgibt.

***Hinweise*:**

- Fehlerbehandlung:
  - Prüfen Sie die Returnwerte von Systemfunktionen (z.B. `malloc()`) auf Fehler ab! Werten Sie die globale Variable `errno` aus.
  - Prüfen Sie die Returnwerte von POSIX-Threadfunktionen (z.B. `pthread_create()`) auf Fehler ab. Schlagen Sie nach, wo im Fehlerfall die

Fehlernummer abgerufen werden kann. Überlegen Sie, wo Sie nachschlagen können. Beachten Sie: POSIX-Funktionen nutzen die globale Variable `errno` nicht.

- Zusatzbibliotheken einbinden:
  - Die Option `-lpthread` bzw. `-pthread` dient zum Linken mit der pthread-Bibliothek.
  - In Eclipse:
    - Menü rechte Maustaste auf `Projektnamen` → `Properties`
    - `Settings` → `Tool-Settings`→`GCC-C-Linker Libraries (-l)` `pthread` hinzufügen.
  - Konsole und gcc: Bei Nutzung von `gcc` auf der Konsole muss diese Option direkt angegeben werden. Z.B. je nach Compilerversion funktioniert
    - `gcc -Wall -lpthread -o a0201 a0201.c`
    - `gcc -Wall -pthread -o a0201 a0201.c` (ohne l am Anfang)
    - `gcc -Wall -o a0201 a0201.c -lpthread`
- Systemzeitermittlung mit Funktionsaufruf `time(NULL)` verwenden.
- Umrechnung der Systemzeit in Datum und Uhrzeit mit `ctime(long *)`.
- Threads warten lassen mit Funktionsaufruf `sleep(sekunden)`
- Evtl. hilft auch `fflush(NULL)`, um gefüllte Ausgabepuffer zu leeren.
- Die Datei `pthread.h` liegt im Verzeichnis `/usr/include`.

Aufgabe 3: Threads erzeugen, Thread kontrollieren, Threads stoppen

Schreiben Sie ein C-Programm, welches einen Hauptthread hat, der zwei neue Threads (Kindthreads) startet. Der erste Kindthread soll jede `halbe Sekunde` einen ihn identifizierenden Namen (den Sie festlegen) und eine Iterationsnummer auf der Konsole ausgeben. Der zweite Kindthread soll alle `zwei Sekunden` einen ihn identifizierenden Namen und eine Iterationsnummer auf der Konsole ausgeben.

3.1 Threads terminieren sich selbst

Die beiden Kindthreads beenden sich nach `6 Sekunden` selbst und liefern einen Statuscode zurück. Der Hauptthread wartet bis sich beide Kindthreads beendet haben, liest dann den jeweiligen Exitcode der Kindthreads und gibt diese ebenfalls auf der Konsole aus und beendet dann das Programm.

3.2 Threads werden von außen terminiert

Der Hauptthread wartet nach Starten der beiden Kindthreads `5 Sekunden` und beendet dann die beiden Kindthreads. Der Hautpthread liest wieder die Exitcodes der Kindthreads und gibt diese auf der Konsole aus.
Was ist der Nachtteil der Terminierungsart in 3.2 bzw. auf was muss man aufpassen?

***Hinweis*:**

- Prozess für einen Zeitraum "schlafen legen" (Zustand blockiert) mit Funktion:
  - `sleep(time)` Einheit für `time` in Sekunden
  - `usleep(time)` Einheit für `time` in Mikrosekunden
  - `nanosleep()` API und Semantik siehe Man-Pages

Optionale Aufgabe 4: Datenaustausch und Synchronisation zwischen Threads

Implementieren Sie folgende Funktionalität durch einen Hauptthread, der zwei (Kind-) Threads startet. Thread 1 erzeugt Zufallszahlen im Bereich `0..19`. Die erzeugten Zufallszahlen sollen dann zusammen mit der Ordnungszahl, die angibt, um die wievielte Zufallszahl es sich jeweils handelt, von Thread 2 auf der Konsole ausgegeben werden.
Zur Kontrolle, ob Thread 2 wirklich alle erzeugten Zufallszahlen ausgibt, gibt Thread 1 seine Zufallszahlen und deren Ordnungszahl ebenfalls auf der Konsole aus. Zum Datenaustausch zwischen den Threads soll ein Puffer verwendet werden, der nur ein Element/Zahl aufnehmen kann. Testen Sie, ob Zufallszahlen verloren gehen!
Der Hauptthread wartet nach Starten der Threads nur noch bis sich die Threads beendet haben und gibt zum Abschluss deren Exitcodes aus.
Messen Sie die Laufzeiten Ihrer Programmdurchläufe in Mikrosekunden.

4.1 Endebedingung: 0 wird als Zufallszahl erzeugt

Falls die Zufallszahl gleich `0` (Null) ist, dann soll Thread 2 nach Ausgabe der Zahl auf der Konsole Thread1 mitteilen, dass er (Thread2) sich umgehend beenden wird. Dann soll Thread2 sich beenden. Thread 1 soll nach Erhalt der Information, dass Thread2 sich beenden will, sich ebenfalls beenden.

4.2 Endbedingung: n.-te Zufallszahl ausgegeben

Lesen Sie als Kommandozeilenargument eine Zahl `n>0` ein. Thread 1 und Thread 2 beenden sich nach der n.-ten Zufallszahl.
Randbedingung: Achten Sie auf die Synchronisation der Threads.
Realisieren Sie in allen Teilaufgaben die Synchronisation ausnahmeweise mit "**aktivem Warten/busy waiting**", d.h. insbesondere **ohne** Semaphore, Mutexe etc.

***Hinweise*:**

- Erzeugung von Zufallszahlen mit `rand()`
  - Übliche Initialisierung des Zufallszahlengenerators:
    - ```
      startTime=time(NULL);
      srand(startTime);"
      ```
- Laufzeitmessung mit der POSIX-Funktion
  `int gettimeofday(struct timeval *ptr, NULL)`

Aufgabe 5: Datenaustausch und Synchronisation zwischen Threads

5.1 Endebedingung: 0 wird als Zufallszahl erzeugt

Implementieren Sie die Synchronisation zwischen Threads zur fachlichen Aufgabenstellung aus Aufgaben 4.1. Benutzen Sie jetzt Mutexe und ggf. Conditionvariablen. Aktives Warten ist jetzt verboten. Außerdem ist jetzt für die Übergabe der Daten (Zahlen) zwischen Erzeuger und Verbraucher ein Puffer der Größe `n` mit `n>=3` zu verwenden. Die Größe n des Puffers soll im Programmcode über ein `#define` einstellbar sein.
Messen Sie die Laufzeiten Ihrer Programmdurchläufe in Mikrosekunden.

Optional 5.2 Endebedingung: n.-te Zufallszahl ausgegeben

Implementieren Sie die Synchronisation zwischen Threads zur fachlichen Aufgabenstellung aus Aufgaben 4.2. Benutzen Sie jetzt Mutexe und ggf. Conditionvariablen. Aktives Warten ist jetzt verboten. Außerdem ist jetzt für die Übergabe der Daten (Zahlen) zwischen Erzeuger
      ```

und Verbraucher ein Puffer der Größe `n` mit `n>=3` zu verwenden. Die Größe n des Puffers soll im Programmcode über ein `#define` einstellbar sein.
Messen Sie die Laufzeiten Ihrer Programmdurchläufe in Mikrosekunden.

***Hinweise*:**

- Überlegen Sie sich zuerst eine Lösung in Form von Pseudocode.

Optionale Aufgabe 6: Synchronisation zwischen mehreren Threads

Pflichtaufgabe für Studierende nach SPO vor WS 2014/15

Erweitern Sie Ihre Lösung von Aufgabe 5 dieses Übungsblattes dahingehend, dass Sie zwar nur einen Erzeuger-Thread für Zufallszahlen, aber zwei Verbraucher-Threads haben. Es gibt nur einen Puffer, über den der Datenaustausch zwischen allen Threads erfolgt.
Messen Sie die Laufzeiten Ihrer Programmdurchläufe in Mikrosekunden.
Achten Sie auf eine korrekte Synchronisation zwischen den Threads und ein korrektes Beenden aller Threads.

Optionale Aufgabe 7: Race-Condition - Mutex-Objekte

Beim modifizierenden Zugriff von zwei oder mehreren Threads auf eine gemeinsame Variable können inkonsistente Variablenveränderungen vorkommen. Dies nennt man Race-Conditions. Das Problem besteht darin, dass z.B. eine Anweisung in einer Hochsprache wie C/C++, Java etc. der Form

```
var++  // Variablenwert um 1 inkrementieren
```

im Maschinencode nicht atomar bearbeitet wird, sondern durch eine Folge von Operationen realisiert wird. Finden nun mitten in solch einer Folge von Operationen Threadwechsel statt, so können inkonsistente Variablenveränderungen auftreten.

Beispiel: Gemeinsame Variable sei i. Operation sei i++

Zeitpunkte	Thread 1	Wert von i	thread 2
t_0		3	
.....	Laden von i in Akku	3	
.....	Akku um 1 erhöhen (Akku hat jetzt Wert 4)	3	
..... (Threadwechsel)		3	
.....		3	Laden von i in Akku
.....		3	Akku um 1 erhöhen (Akku hat jetzt Wert 4)
.....		4	Akku nach i schreiben
..... (Threadwechsel)		4	
	Akku nach i schreiben	4	

Die Variable wurde insgesamt von 3 auf 4 erhöht, obwohl sie zweimal inkrementiert wurde. Hier ist offensichtlich etwas schief gelaufen.
Messen Sie bei allen Teilaufgaben die Laufzeiten Ihrer Programmdurchläufe in Mikrosekunden.

7.1 Race-Condition sichtbar machen

Programmieren Sie ein Beispiel, bei dem eine Race-Condition zuschlägt.

7.2 Mutex-Objekt zur Verhinderung einer Race-Condition einsetzen

Benutzen Sie Mutex-Objekte, um die Race-Condition in Ihrer Lösung von Teil 1 der Aufgabe zu eliminieren.

7.3 Performanzvergleich

Vergleichen Sie die benötigte Zeit für Ihr Beispiel, einmal ohne und einmal mit Synchronisation über Mutexe.