Übungsblatt 7

Dr. Matthias Frank, Dr. Matthias Wübbeling

Ausgabe Mittwoch, 22. November 2023
Abgabe bis Freitag 1. Dezember 2023, 23:59 Uhr
Vorführung vom 4. bis zum 8. Dezember 2023

Alle Programme müssen unter **Ubuntu 22.04** kompilierbar bzw. lauffähig und ausreichend kommentiert und mit **Makefile** (C, Assembler) versehen sein, um Punkte zu erhalten. Als Compiler sollen **clang** (C) und **nasm** (Assembler) verwendet werden. Die Lösungen sind bei Ihrem Tutor während Ihrer Übungsgruppen vorzuführen. Alle Gruppenmitglieder sollten die Abgabe erklären können. Die Abgabe erfolgt mittels Ihres Git-Repositories und der vorgegebenen Ordnerstruktur.

Die Punkte der Aufgaben sind relevant für die Zulassung. Die Punkte der Bonusaufgaben werden auf Ihren Punktestand addiert, werden aber nicht auf die für die Zulassung benötigten Punkte addiert.

Abgabestruktur: Jede Abgabe, die die folgende Struktur nicht umsetzt, wird **NICHT** bepunktet bzw. mit 0 Punkten bewertet

- die zu korrigierenden Lösungen müssen bis zur Deadline auf dem `master`-Branch liegen. Lösungen auf anderen Branches werden nicht gewertet
- alle Lösungen müssen in der vorgegebenen Ordnerstruktur (**blattXX/aufgabeYY**) abgelegt werden, wobei `XX` und `YY` durch die jeweiligen Nummern des Zettels und der Aufgaben ersetzt werden sollen. Der Name soll exakt nur aus diesen Zeichen bestehen und achtet auf Kleinschreibung
- sämtliche Aufgaben, mit Ausnahme der theoretischen Aufgaben, die eine PDF erfordern, benötigen zwingend ein **Makefile**. Abgaben ohne dieses werden nicht gewertet

Hinweis:	Manche Browser sind nicht dazu in der Lage die in die PDFs eingebetteten Dateien anzuzeigen. Mittels eines geeigneten Readers, wie dem Adobe Reader, lassen sich diese jedoch anzeigen und verwenden.

Aufgabe 1 Prozessunterbrechungen messen (4 Punkte)

Schreiben Sie ein C-Programm, das in einer Schleife eine gegebene Zeitspanne wartet und die tatsächlich gewartete Zeit misst (`clock()` eignet sich hierfür nicht). Die zu wartende Zeitspanne und die Zahl der Iterationen soll über einen Kommandozeilenparameter in Nanosekunden übergeben werden. Der Aufruf, wenn das Programm 100 ns warten und 10 000 Iterationen laufen soll, soll aussehen wie

```
./timing 100 10000
```

.

Schreiben Sie in jedem Durchlauf die tatsächlich gemessene Zeitdifferenz in Nanosekunden in eine neue Zeile einer Datei, die sie mit dem Namen **wartezeit_iterationen.txt** abspeichern; im obigen Beispiel soll sie also **100_10000.txt** heißen.

Experimentieren Sie mit verschiedenen Angaben für die zu wartende Zeitspanne und protokollieren sie entsprechend. Lassen Sie das Programm laufen und protokollieren Sie die Ausgaben, während Sie dem Rechner parallel weitere Aufgaben geben, beispielsweise rechenintensive Operationen (Archiv entpacken, cpuburn, etc.). Was für Veränderungen stellen Sie fest?

Da die Veränderungen von Betriebssystem und Hardware abhängen, gibt es keine falschen Antworten. Begründen Sie ihre Erkenntnisse daher durch eine Auswertung der Daten und geben Sie die Auswertung zusammen mit dem Programm ab.

Aufgabe 2 Speicherlayout von Prozessen (4 Punkte)

Das Ziel dieser Aufgabe ist es, sich das Layout von Prozessen im Speicher anhand eines einfachen Beispiels anzuschauen und dabei das Tool `objdump` kennenzulernen.

`objdump` ist Teil der GNU binutils. Hinweise zu dessen Verwendung finden Sie in der dazugehörigen man-Page oder in der Dokumentation der GNU binutils (`http://sourceware.org/binutils/docs/`).

1. Schreiben Sie ein einfaches Programm (die eigentliche Funktionalität des Programms ist unerheblich), das verschiedene Arten von Variablen verwendet:
   - Globale Variable ohne Initialisierungswert
   - Globale Variable mit 0 initialisiert
   - Globale Variable mit Initialisierungswert ungleich 0.
   - Globale Datenkonstante

   Hinweis: Der AddressSanitizer greift stark in das Speicherlayout und die Verwaltung dynamischen Speichers ein. Unter anderem werden Pufferzonen um Speicherelemente angelegt, um Buffer Overflows zu erkennen. Außerdem können Variablen anders angeordnet werden. Verwenden Sie daher **nicht** den AddressSanitizer (`--fsanitize=address`). Schalten Sie ebenfalls alle Optimierungen aus, sofern Sie sonst welche verwenden.

2. Verwenden Sie das Tool `objdump`, um das vom Linker erstellte Speicherlayout Ihres Programms zu analysieren. Beantworten Sie folgende Fragen:
   - An welchen Adressen wurden Ihre Variablen platziert?
   - Zu welchen Abschnitten (z.b. `.data`) gehören die Variablen jeweils? Wie groß sind diese Abschnitte insgesamt?
   - Worin liegt der Unterschied zwischen dem `.data`- und dem `.bss`-Abschnitt? Recherchieren Sie die Antwort gegebenenfalls im Internet.
   - Mit welchen Parametern haben Sie `objdump` aufgerufen, um die geforderten Informationen zu erhalten?

3. Erweitern Sie Ihr Programm so, dass es die Adressen Ihrer Variablen im Speicher auf dem Bildschirm ausgibt und vergleichen Sie die Ergebnisse mit den Ergebnissen aus dem vorherigen Teil.

   Fügen Sie nun noch Variablen auf dem Heap und auf dem Stack hinzu. Geben Sie ebenfalls die Speicheradressen dieser Variablen aus. Demonstrieren Sie, dass der Stack tatsächlich nach unten wächst.

Aufgabe 3 Eigenschaften von Threads (2 Punkte)

In dieser Aufgabe sollen Sie sich mit der Verwendung und den Eigenschaften von Threads vertraut machen. Dazu verwenden wir die pthread-Bibliothek, die dem POSIX-Standard folgt. Informationen zur pthread-Bibliothek finden Sie in den entsprechenden Manpages. Betrachten Sie das folgende Programm, welches in die PDF-Datei eingebettet ist.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void* printer();
int cnt = 0;

int main(void) {
        pthread_t thread1, thread2;
        printer();
        pthread_create(&thread1, NULL, printer, NULL);
        sleep(1); // Sleep-Aufruf 1
        pthread_create(&thread2, NULL, printer, NULL);
        pthread_join(thread1, NULL);
        pthread_join(thread2, NULL);
}

void* printer() {
        // sleep(1); // Sleep-Aufruf 2
        printf("cnt = %d, ppid = %5d, my_pid = %5d\n",
                    cnt, getppid(), getpid());
        cnt++;
}
```

1. Beschreiben Sie zunächst, was das Programm als Ausgabe liefert und wie die Verwendung der Threads zu diesem Ergebnis beiträgt. Gehen Sie dabei auf die Beantwortung folgender Fragen ein: Warum ergibt sich jeweils für die Ausgabe `ppid` bzw. `my_pid` immer derselbe Wert? Warum liefert dieses Programm im Gegensatz zu den vorherigen Programmen mit `fork()` nur drei Ausgabezeilen? Warum ergibt sich für den Wert der Variablen `cnt` immer die Folge 0, 1, 2?

2. Wie verändert sich die Ausgabe des Programms, wenn man den Aufruf der Funktion `sleep(1)` in Zeile 12 auskommentiert und stattdessen den Aufruf in Zeile 19 einkommentiert? Erläutern Sie Ihre Beobachtung.