Systemprogrammierung - AIN/2

Sommersemester 2022

Übungsaufgabe 3: C Strings und Funktionen

Abgabe bis 5./6.5.2022

Migration von Java nach C: Stringsort

Das Java-Programm ⤵ **Stringsort.java** erzeugt zufällige Ziffernstrings und gibt sie alphabetisch (nicht numerisch!) sortiert aus. Dabei werden mehrfach vorkommende Strings nur einmal ausgegeben. Jedes weitere Vorkommen wird mit einem angehängten Stern angezeigt. Stellen Sie dieses Java-Programm auf C um:

- Die Quelldatei soll `stringsort.c` heißen und analog zum Java-Programm die beiden Funktionen `main` und `bubblesort` enthalten.
- Bestimmen Sie die Array-Größe `n` wie in Aufgabe 2 mit der Funktion `atoi` aus dem Kommandozeilenargument `argv[1]`.
- Analog zum Array von String-Referenzen der Java-Vorlage brauchen Sie im C-Programm ein Array von String-Zeigern. Verwenden Sie die Bibliotheksfunktion `malloc` zum Reservieren des Speichers für das Array.
- Verwenden Sie wie in Aufgabe 2 die Bibliotheksfunktionen `srand` und `rand` zum Würfeln. Mit der Bibliotheksfunktion `sprintf` können Sie die gewürfelten Zahlen in Strings umwandeln. Vor jedem Aufruf von `sprintf` müssen Sie mit `malloc` ein eigenes Speicherstück für den jeweils zu erstellenden String reservieren. Geben Sie als Länge `m` des Speicherstücks die Länge des Strings `argv[1]` inklusive Stringendezeichen '\0' an. Warum ist das ein sinnvoller Wert?
- Verwenden Sie die `strxxx`-Funktionen aus der C-Standardbibliothek für Stringvergleiche und das Zusammenbauen des Ausgabestrings. Den Platzbedarf des Ausgabestrings können Sie mit Hilfe des Rückgabewerts von `sprintf` vorab berechnen.
- Schreiben Sie wie in der Java-Vorlage das sortierte Array als einen einzigen String auf die Standardausgabe.

Denken Sie an die Fehlerbehandlung nach den Speicherreservierungen sowie an die Freigabe des Speichers am Programmende.

Test

Speichern Sie die Datei ⤵ **Makefile** in Ihr Arbeitverzeichnis der Aufgabe 3 und verwenden Sie zum Testen die Befehle:

```
javac Stringsort.java
java Stringsort 200
make stringsort
```

```
./stringsort 200
valgrind ./stringsort 200
make cppcheck
```

Testen Sie Ihre C-Programm mit den Array-Größen 0, 1, 2, 20 und 200, und rufen Sie es auch ohne Angabe einer Array-Größe auf:

- Verhält sich das C-Programm bei allen Eingaben wie das Java-Programm?
- Meldet `valgrind` Fehler?
- Meldet `cppcheck` Probleme?

Bessern Sie gegebenenfalls nach.

Optimierung des Speicherbedarfs

Ihr nach der obigen Anleitung erstelltes C-Programm braucht unnötig viel Heapspeicher, weil es viele sehr kleine Speicherstücke reserviert. Pro Speicherstück muss sich die Speicherverwaltung die Länge merken. Außerdem liefert die Speicherverwaltung für beliebige Datentypen ausgerichtete Adressen, also typischerweise durch 16 teilbare Adressen. Das führt zu einer Aufrundung des tatsächlich belegten Speichers. Bei zum Beispiel angeforderten 4 Byte werden dadurch inklusive der Längeninformation vermutlich 32 Byte belegt. Und nur wegen der verstreuten Speicherung der einzelnen Strings in jeweils eigenen Heap-Stücken wird überhaupt das Array von String-Zeigern gebraucht.

Kopieren Sie Ihre Datei `stringsort.c` nach `stringsort-optimiert.c` und optimieren Sie das kopierte Programm wie folgt:

- Legen Sie die gewürfelten Strings in festem Abstand `m` hintereinander in einen einzigen zusammenhängenden Speicherbereich. Reservieren Sie diesen Speicherbereich vor der Schleife mit `malloc`. Der feste Abstand `m` ist die berechnete maximale String-Länge inklusive Endezeichen. Lassen Sie das ursprüngliche Array von String-Zeigern weg.
- Geben Sie der Funktion `bubblesort` den Typ der Bibliotheksfunktion `qsort` aus <stdlib.h> und rufen Sie `bubblesort` mit `strcmp` als Vergleichsfunktion auf.
- Die Funktion `bubblesort` muss nun die Speicherstücke der Strings vertauschen, was mit dem Zuweisungsoperator nicht möglich ist. Ersetzen Sie die Zuweisungen durch Aufrufe der Bibliotheksfunktion `memcpy`, um jeweils `m` Byte zu vertauschen.

Hinweis: der zusammenhängende Speicherbereich ist im Prinzip eine n-mal-m-Matrix von Zeichen, realisiert als Array von Arrays auf dem Heap. Weil beide Matrixdimensionen, die Anzahl der Strings und die maximale String-Länge, erst zur Laufzeit feststehen, müssen Sie statt mit Index-Operator mit Adressrechnung arbeiten.

Testen Sie das optimierte Programm:

```
make stringsort-optimiert
./stringsort-optimiert 200
valgrind ./stringsort-optimiert 200
```

Ist das speicher-optimierte Programm schneller als das nicht optimierte?

```
time ./stringsort 20000 > /dev/null
time ./stringsort-optimiert 20000 > /dev/null
```

Protokoll

Erstellen Sie ein Protokoll der Tests Ihrer beiden Programmvarianten. Gehen sie wieder so vor wie in **Aufgabe 1** beschrieben. Nennen Sie die Protokolldatei `protokoll-aufgabe3.txt` und ergänzen Sie darin Ihre Antworten auf alle Fragen.

Abgabe

Führen Sie die beiden Programme und Ihre Protokolldatei vor.

Hinweis:
Der Compiler `gcc` darf für Ihr Programm keine Fehler oder Warnungen mehr ausgeben.
Ihre Programme müssen außerdem ordentlich formatiert sein. Bessern Sie die Formatierung gegebenenfalls mit `astyle` nach:

```
astyle -p -H --style=ansi stringsort.c stringsort-optimiert.c
```

Freiwillige Zusatzaufgabe (2 Bonuspunkte)

Kombinieren Sie die beiden Versionen Ihres Sortierprogramms. Es soll wie in `stringsort.c` ein Array von String-Zeigern sortiert werden, aber die Strings sollen wie in `stringsort-optimiert.c` in festem Abstand in einem einzigen zusammenhängenden Speicherbereich liegen.

Ist das kombinierte Programm schneller als die beiden ursprünglichen Programme?

Prof. Dr. H. Drachenfels
Hochschule Konstanz** - **Impressum** - **Datenschutzerklärung

Letzte Änderung: 21.4.2022