Dyn.Speicher (Stream)

Dienstag, 22. März 2022 15:32

Grundlagen

Streams

Wikipedia: Mit **Datenströmen** (englisch ***data streams***) bezeichnet man in der Informatik einen kontinuierlichen Fluss von Datensätzen, dessen Ende meist nicht im Voraus abzusehen ist; die Datensätze werden fortlaufend verarbeitet, sobald jeweils ein neuer Datensatz eingetroffen ist.

```
Java: system.out.print(hello)
C++: cout << "This " << " is a " << "single C++ statement" << endl;
```

Da der Speicherplatz im Vorhinein nicht bekannt ist, kann der Speicherplatz folglich nicht über lokale/globale Arrays abgebildet werden. Vielmehr wird Speicher vom Heap hierfür reserviert. Dies erfolgt über die malloc() Funktion, welcher einen Zeiger auf die Startadresse des zugewiesenen Speichers bekommt.

```
char *stream=malloc(511+1);
```

Dieser Zeiger entspricht einem Array, so dass auf diesen Speicher genauso zugegriffen werden kann, wie auf ein Array:

```
stream[0]='H';
stream[1]='a';
stream[2]='\0';
strcat(stream,"llo");
printf("'%s'\n",stream);
```

Wird der Speicher nicht mehr benötigt, so muss dieser Speicher wieder freigegeben werden. Andernfalls reduziert diese bis zum Programmende den verfügbaren Speicherbereich und kann nicht anderweitig benutzt werden.

```
free(stream);
```

Ein Stream könnte unter Nutzung des Heaps wie folgt realisiert werden

```
char *stream=NULL;   //Globale Variable
void stream_append(char *append) {
    if(stream==NULL) {    //Erstmaliger Aufruf
        stream=malloc(strlen(append)+1+1); //Passend Speicher
        reservieren
        strcpy(stream,append);             //String in Speicher
        kopieren
        strcat(stream,"\n");               //Zeilenende anhängen
    }
    else {                //Fortlaufender Aufruf, d.h. append
    anhängen
        char *new=malloc(strlen(stream)+   //Passend Speicher
        reservieren
                         strlen(append)+1+1);
        strcpy(new,stream);                //Alten String kopieren
        strcat(new,append);                //Neuen String anhängen
        strcat(new,"\n");                  //Zeilenende anhängen
        free(stream);                      //Alten Speicher freigeben
        stream=new;
    }
}
```

Die malloc() Funktion ist sehr rechenintensiv, da mit jedem Aufruf von malloc() der gesamte verfügbare Speicher durchsucht werden muss und der kleinste passende Bereich zurückgegeben

wird.
Eine Optimierungsmöglichkeit (welche von Streams genutzt wird) ist, nicht nur passend Speicher zu reservieren, sondern Speicher im vielfachen einer zuvor definierten Größe (im nachfolgenden Alloc_Size bezeichnet). Die Idee hierbei ist, dass beim Anhängen eines Strings an den Stream nicht neu Speicher reserviert werden muss, sondern der bereits allozierte Speicherbereich für die Aufnahme des vorhanden und des anzuhängenden Strings genutzt werden kann.
Dies würde für den ersten Aufruf von stream_append() wie folgt aussehen:

```
          char  *stream=NULL;    //Globale Variable
          size_t size_max=0;     //Größe des vom Streams benötigten
    Speicherplatzes
    const size_t alloc_size=64;  //Speicher in ganzzahligen Vielfachen
    dieser Größe

    void stream_append(char *append) {
        if(stream==NULL) {
            size_max= (((strlen(append)+1)/alloc_size)+1)*alloc_size;
            //62(+1)->(((        62    +1)/   64     )+1)* 64 =  (0+1)*64
            =64
            //63(+1)->(((        63    +1)/   64     )+1)* 64 =  (1+1)*64
            =64
            //64(+1)->(((        64    +1)/   64     )+1)* 64 =  (1+1)*64
            =128
            stream=malloc(size_max);
            strcpy(stream,append);
            strcat(stream,"\n");   //Zeilenende, zum Trennen der
            einzelnen
        } else                     //Eingaben
          //Ihre Aufgabe
    }
```

Sofern beim Anhängen eines weiteren Teilbereich noch genügend Speicher vorhanden ist, kann dieser ohne Neu-Allokation einfach angehängt werden. Erst wenn der vorhandene Speicher nicht mehr ausreicht, muss neuer Speicher angefordert werden (natürlich im Vielfachen von Alloc_size).
Beispiel (hier alloc_Size=32Byte)

```
    stream_set("Hallo Welt");
    //Mit dem erstmaligen Aufruf von stream_set() wird vom Heap 32-Byte
    Speicher
    //angefordert, von dem aber zunächst nur 10+1 Byte benötigt werden

    stream_set("zweiteZeile");
    //Da für den zweiten Aufruf weitere 11 Bytes benötigt werden, kann der
    ergänzende //String hinten angehängt werden. Damit der resultierende
    String ein
    //zusammenhängender String ist, beginnt "zweiteZeile" am
    Stringendezeichen des
    //vorherigen Strings.

    stream_set("dritteZeilelang");
    //Der Stream ist zu diesem Zeitpunkt mit 10+11+1 Belegt, d.h. es sind
    noch
    //32-(10+11+1)=10 Zeichen Frei. Dies reicht somit nicht für die hier
    benötigten
    //15 Zeichen aus, so dass nun der reservierte Speicher mittels realloc
    //vergrößert werden muss.
```

Beim Entnehmen von Zeichen aus dem Stream stehen zwei Varianten zur Wahl:

- Gesamten Stream entnehmen
- In der Reihenfolge des Anhängens entnehmen, d.h. beginnend von Vorne bis zum '\n' Zeichen

entnehmen (entsprechend der Standardeingabe).
Im ersten Fall kann nach Entnahme der gesamte reservierte Speicher freigegeben werden. Im zweiten Fall empfiehlt sich der Einfachheit halber den verbleibenden Stream nach vorne zu verschieben.

Aufgabe

Erweitern sie stream_append() und stream_get(), so dass diese die obigen Funktionalität für Streams darstellen.
Zum Testen der Funktion benutzen sie 'integrierten' Kommandozeileninterpreter, welcher nachfolgende Befehle zur Verfügung stellt:

```
>>help        --> Ohne Worte
>>quit        --> Programm beenden
>>apend xxx   --> xxx an Stream anhängen
>>getl        --> Eine Zeile aus Stream auslesen
>>geta        --> Gesamten Stream auslesen
>>debug       --> Darstellung des Inhalts des Streams
```

Beim CompilerExplorer könnten sie dann komfortable mehrere Testfälle in einem Rutsch durchführen lassen

```
>>set Hallo
>>set  1234567890
>>set  extraLaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaangerString
>>geta
>>set  Von vorne mit einem extra langen String, der sogar noch Leerzeichen enthält
>>getl
>>getl
>>quit
```

Wenn sie das Programm händisch per GCC compilieren und händisch starten, könnten sie oben aufgeführte Testfälle z.B. in der Datei test.txt speichern und beim Aufruf des Programms die Standardeingabe wie folgt von dieser Datei 'lesen' lassen:
>>./a.out <text.txt

Hinweis:

- Berücksichtigen sie alle Fehlerfälle. D.h. wenn Funktionen fehlschlagen können (hier insbesondere malloc()) prüfen sie den Rückgabewert und handeln sie entsprechend
- Sollte stream_get() oder stream_set() fehlschlagen (z.B. aufgrund eines fehlgeschlagenen malloc() oder weil noch kein String in dem Stream gespeichert wurde), so teilen sie dies über dem Rückgabewert dem Aufrufer mit.
- Bedenken sie bei stream_get, wo der Speicher für den Rückgabestring reserviert wird

```c
#include <stdio.h>     //fuer printf() size_t stderr
#include <string.h>    //fuer strcpy() memcpy()
#include <stdlib.h>    //fuer malloc() free()
//Compilerschalter: -fsanitize=address -Wall
struct stream {
    char   *str;
    size_t  size_max;     //Größe des derzeit reservierten Speichers
    size_t  size_act;     //Benötigter Speicher
    size_t  alloc_size;   //Heap-Blöcke in vielfachen dieser Größe
} stream;
void stream_open(size_t alloc_size) {
    stream.str = NULL;
    stream.size_act=0;  //Tatsächliche Größe
    stream.size_max=0;  //Reservierte Speichergröße
    stream.alloc_size=alloc_size;
}
void stream_close( void ) {
    if(stream.str)
        free(stream.str);
    stream.str=NULL;
    stream.size_act=0;
```

```
    stream.size_max=0;
}
void stream_debug(void)
{
    unsigned char *ptr_start=(unsigned char *)stream.str;
    unsigned char *ptr_end  =(unsigned char *)stream.str+stream.size_act;
    printf("\n-----------------------------------------\n");
    printf("size_max  =%8zu size_act  =%8zu \n"
           "alloc_size=%8zu buffer    =%p\n",
           stream.size_max,stream.size_act,
           stream.alloc_size,stream.str);
    if(stream.str)
        //Je alloc_size eine Zeile ausgeben!
        for( ;ptr_start<ptr_end ; ptr_start++)
            printf("'%c':%02x %s",
                   *ptr_start>=' '?*ptr_start:'?',
                        *ptr_start,
                             ((ptr_start-(unsigned char*)stream.str)%
stream.alloc_size)==(stream.alloc_size-1)?"\n":" ");
    printf("\n-----------------------------------------\n");
}
typedef enum {STATUS_OK,STATUS_MALLOCFAILED} STATUS;
STATUS stream_append(const char *append) {
    if(stream.str==NULL) {
        stream.size_max= (((strlen(append)+1)/stream.alloc_size)+
1)*stream.alloc_size;
        //62(+1+1)    -> (((       62     +1)/   64            )+1)* 64 =  (0+
1)*64=64
        //63(+1+1)    -> (((       63     +1)/   64            )+1)* 64 =  (1+
1)*64=64
        //64(+1+1)    -> (((       64     +1)/   64            )+1)* 64 =  (1+
1)*64=128
        stream.str=malloc(stream.size_max);
        if(stream.str == NULL)
            return STATUS_MALLOCFAILED;
        strcpy(stream.str,append);
        strcat(stream.str,"\n");
        stream.size_act=strlen(stream.str)+1;
    } else {
        //Ihre Aufgabe
        //Hinweis
        //Vergrößerung des Speichers (sofern notwendig) wahlweise
        //über malloc()+free() oder über realloc()
    }
    return STATUS_OK;
}
typedef enum {STREAM_MODE_LINE,STREAM_MODE_ALL} STREAM_MODE;
char *stream_get(STREAM_MODE mode)
{
    //Ihre Aufgabe

    return NULL; //Für den Fehlerfall
}
const  char *HELP_STRING =
    "Help\n"
    ">>help        --> Ohne Worte\n"
    ">>quit        --> Programm beenden\n"
    ">>apend xxx   --> xxx an Stream anhängen\n"
    ">>getl        --> Eine Zeile aus Stream auslesen\n"
    ">>geta        --> Gesamten Stream auslesen\n"
    ">>debug       --> Darstellung des Inhalts des Streams\n";
int main(int argc, char *argv[]){
```

```
    enum status {OK,KO_FGETS,KO_END} ret=OK;
    printf("File:"__FILE__ " erstellt am "__DATE__" um "__TIME__ " gestartet
\n");
    stream_open(16);
    while(1) {
        char input[100];
        char *arg1,*arg2;

        if(fgets(input,sizeof input,stdin)==NULL) {
            ret=KO_FGETS;
            break;
        }
        arg1=strtok(input," \n");

        if(arg1==NULL)
            continue;
        else if(!strcasecmp(arg1,"help"))
            printf("%s",HELP_STRING);
        else if(!strcasecmp(arg1,"quit"))
            break;
        else if(!strcasecmp(arg1,"debug"))
            stream_debug();
        else if(!strcasecmp(arg1,"append")) {
            arg2=strtok(NULL,"\n");
            if(arg2!=NULL) {
                if(stream_append(arg2)!=STATUS_OK)
                    fprintf(stderr,"\e[31mstream_append() Error:\e[30m\n");
            }
            else
                fprintf(stderr,"\e[31mstream_append() Error:\e[30m Missing
Operand\n");
        }
        else if(!strcasecmp(arg1,"getl")) {
            char *ret=stream_get(STREAM_MODE_LINE);
            if(ret!=NULL) {
                printf("<<%s>>\n",ret);
                free(ret);
            } else
                fprintf(stderr,"\e[31mstream_get() Error:\e[30m\n");
        }
        else if(!strcasecmp(arg1,"geta")) {
            char *ret=stream_get(STREAM_MODE_ALL);
            if(ret!=NULL) {
                printf("<<%s>>\n",ret);
                free(ret);
            } else
                fprintf(stderr,"\e[31mstream_get() Error:\e[30m\n");
        }
        else
            printf("Invalid Command\n");
    }
    stream_close();
    const char *retstring[]={
        [KO_FGETS]   ="Programm mangels Eingabe beendet",
        [OK]         ="Programm ordnungsgemäß beendet",
        [KO_END]     ="Unbekannter Rückgabewert",
    };
    printf("%s\n",retstring[ret>KO_END?KO_END:(ret<0?KO_END:ret)]);
    return (int)ret;
}
```