

Musterlösungen zu den Hausaufgaben auf  
Blatt 2  
der Übungen zur Vorlesung  
“Grundlagen Betriebssysteme und Systemsoftware

---

G.Groh, 31.10.2008

## Aufgabe 1.1

```
all: # es gibt nur ein target
```

```
    export LD_LIBRARY_PATH=. # der library path dient dem gcc  
dazu festzustellen wo sich die libs befinden
```

```
    gcc -S main.c -o main.s # die option -S compiliert statt  
in ein .o file in ein assembler file (.s)
```

```
    gcc '-DWORLD="world"' -S helloworld.c -o helloworld.s  
#präprozessoranweisungen (textersetzung) können mit der option  
-D auch im gcc angegeben werden.
```

```
    gcc '-DWORLD="universe"' -S helloworld.c -o  
hellouniverse.s #zweite version von helloworld
```

```
    gcc -c main.s -o main.o # aus dem assembler file von main  
wird ein object file. haette man auch direkt aus .c erzeugen  
koennen.
```

```
    gcc -c helloworld.s -o helloworld.o # aus dem assembler  
file von helloworld wird ein object file. haette man auch direkt  
aus .c erzeugen koennen.
```

## Aufgabe 1.1

```
gcc -c hellouniverse.s -o hellouniverse.o # aus dem  
assembler file von helloworld wird ein object file. haette man  
auch direkt aus .c erzeugen koennen.
```

```
gcc -shared -o libhelloworld.so helloworld.o # erzeugen  
einer shared library libhelloworld.so aus helloworld.o
```

```
gcc -shared -o libhellouniverse.so hellouniverse.o #  
erzeugen einer shared library libhellouniverse.so aus  
hellouniverse.o
```

```
ar cru libhelloworld.a helloworld.o # erzeugen einer  
statischen library libhelloworld.a aus helloworld.o
```

```
ar cru libhellouniverse.a hellouniverse.o # erzeugen einer  
statischen library libhellouniverse.a aus hellouniverse.o
```

```
gcc main.o helloworld.o -o main1a # erzeugen eines  
executables main1a aus den beiden object files
```

```
gcc main.o libhelloworld.a -o main1b # erzeugen eines  
executables main1b aus dem object file von main und der  
statischen bibliothek libhelloworld
```

## Aufgabe 1.1

```
gcc main.o -L. -lhelloworld -lhellouniverse -o main3a #  
erzeugen eines executables main3a aus dem object file von main  
und den dynamischen bibliotheken libhelloworld und  
libhellouniverse. library path ist das aktuelle directory.
```

```
gcc main.o -L. -lhellouniverse -lhelloworld -o main3b  
# dito
```

```
gcc main.o helloworld.o -o main4a # erzeugen eines  
executables main4a aus den beiden object files
```

```
gcc main.o -L. -lhelloworld -o main4b # erzeugen eines  
executables main4b aus dem object file von main und mit  
dynamischem linken der bibliothek helloworld
```

```
gcc main.o helloworld.o -o main5a # siehe main4a
```

```
gcc main.o -static -L. -lhelloworld -o main5b # erzeugen  
eines statisch gelinkten executables main5b aus dem object file  
von main und der bibliothek helloworld
```

## Aufgabe 1.2

Es gibt keinen wesentlichen Unterschied. Der Archiver `ar` macht mit den `.o` files nicht viel (außer komprimieren) (im wesentlichen ähnlich zu `tar zip jar` o.ä.).

## Aufgabe 1.3

Es wird ein Symbol immer aus der zuerst angegebenen Bibliothek genommen. Dies ist eine einfache Strategie um Namenskonflikte aufzulösen.

## Aufgabe 1.4

Der Hauptunterschied ist, dass im Fall von `main4a` die Bibliothek statisch hinzugelinked wird und im Fall von `main4b` die Bibliothek dynamisch hinzugelinked wird.

## Aufgabe 1.4

Ausgabe von `nm main4a` (statisch gelinked)

```
...  
  
000103d0 ? _init  
00010680 R _lib_version  
000103ec T _start  
          U atexit@@SYSVABI_1.3  
00010658 t call___do_global_ctors_aux  
00010518 t call___do_global_dtors_aux  
000105ac t call_frame_dummy  
00020848 b completed.1  
          U exit@@SYSVABI_1.3  
00010528 t frame_dummy  
000105bc T main  
0002084c b object.2  
00020840 d p.0  
          U printf@@SYSVABI_1.3  
000105d8 T say_hello
```

## Aufgabe 1.4

Ausgabe von `nm main4b (dynamisch gelinked)`

```
...  
  
000104a4 ? _init  
00010728 R _lib_version  
000104c0 T _start  
          U atexit@@SYSVABI_1.3  
00010704 t call___do_global_ctors_aux  
000105ec t call___do_global_dtors_aux  
00010680 t call_frame_dummy  
000208e0 b completed.1  
          U exit@@SYSVABI_1.3  
000105fc t frame_dummy  
00010690 T main  
000208e4 b object.2  
000208d8 d p.0  
          U say_hello
```

## Aufgabe 1.4

In der **statisch gelinkten Variante** kann der Linker bereits alle Symbole in allen Komponenten (main und Bibliothek) mit einer vorläufigen Adresse versehen, da kein anderes Executable auf die Bibliothek zugreifen muss / kann. Der Loader kann das “ganze Packet” dann zur Ausführungszeit im Speicher platzieren.

In der **dynamisch gelinkten Variante** kann der Linker nicht alle Symbole in allen Komponenten (main und Bibliothek) mit einer vorläufigen Adresse versehen, da andere Executables auf die Bibliothek zugreifen müssen können sollen (deswegen auch “shared library (.so)”). Insofern wird das Symbol in der Symboltabelle mit dem tag U (“unknown”) versehen (statt mit T wie vorher).

Der Loader kann nicht das “ganze Packet” zur Ausführungszeit “en-Block” im Speicher platzieren, sondern wird die shared library so platzieren, dass auch andere Executables darauf zugreifen können.

## Aufgabe 1.5

In `main5a` wird die “library” quasi auch “statisch dazugelinked” indem einfach das `.o` file der Bibliothek mit dem `.o`-File von `main` direkt zusammengefügt werden, während in `main5b` ALLE Bibliotheken (auch die `glibc`) statisch ins Executable eingefügt werden.

Deswegen ist `main5b` auch viel größer weil die automatisch immer dazugelinkte `glibc` nun komplett enthalten ist.

## Aufgabe 2

```
struct personalRecord{
    int salary;
    struct personalRecord* previousRecord;
    struct personalRecord* nextRecord;
};

int main(){
    int i;
    struct personalRecord firstRecord;
    struct personalRecord *theNewRecord;
    struct personalRecord *theRecordBeforeTheNewRecord;
    firstRecord.previousRecord = NULL;
    firstRecord.nextRecord = NULL;
    firstRecord.salary = 1000;
    theRecordBeforeTheNewRecord = <expr1> ;
    for(i=0; i<9; i++){
        theNewRecord = newRecord();
        theNewRecord->salary = 1000;
        <expr2> = theRecordBeforeTheNewRecord;
        <expr3> = theNewRecord;
        theRecordBeforeTheNewRecord = theNewRecord;
    }
    return 0;
}

struct personalRecord* newRecord(){
    struct personalRecord *newPersonalRecord;
    newPersonalRecord = (struct personalRecord *) malloc( <expr4> );
    return newPersonalRecord;
}
```

## Aufgabe 2

```
struct personalRecord{
    int salary;
    struct personalRecord* previousRecord;
    struct personalRecord* nextRecord;
};
```

```
int main(){
    int i;
    struct personalRecord firstRecord;
    struct personalRecord *theNewRecord;
    struct personalRecord *theRecordBeforeTheNewRecord;
    firstRecord.previousRecord = NULL;
    firstRecord.nextRecord = NULL;
    firstRecord.salary = 1000;
    theRecordBeforeTheNewRecord = <expr1> ;
    for(i=0; i<9; i++){
        theNewRecord = newRecord();
        theNewRecord->salary = 1000;
        <expr2> = theRecordBeforeTheNewRecord;
        <expr3> = theNewRecord;
        theRecordBeforeTheNewRecord = theNewRecord;
    }
    return 0;
}
```

```
struct personalRecord* newRecord(){
    struct personalRecord *newPersonalRecord;
    newPersonalRecord = (struct personalRecord *) malloc( <expr4> );
    return newPersonalRecord;
}
```

Zu ersetzen sind:

<expr1> mit &firstRecord;

<expr2> mit theNewRecord->previousRecord

<expr3> mit theRecordBeforeTheNewRecord->nextRecord

<expr4> mit sizeof(struct personalRecord)