SYSPROG BEISPIEL 1

Aufgabenstellung A

Man implementiere eine vereinfachte Version des Programms `cat`.

```
SYNOPSIS:
    mycat [-vET]
```

Das Programm soll Zeichen von der Standardeingabe lesen und auf die Standardausgabe schreiben. Zusätzlich sollen bei der Angabe entsprechender Optionen Escape-Sequenzen anstatt gewisser Zeichen ausgegeben werden.

Anleitung

Wird keine Option angegeben, so soll die Eingabe auf die Ausgabe kopiert werden. Wird die Option `-E` angegeben, so sollen Zeilenvorschübe (`\n`) durch die Zeihenfolge `$\n` ersetzt werden. Wird die Option `-T` angegeben, so sollen Tabulatorzeichen (`\t`) durch die Zeichenfolge `^I` ersetzt werden. Wird die Option `-v` angegeben, so sollen alle Zeichen außer den zuvor genannten beiden folgendermaßen dargestellt werden: Die Zeichen 0 bis 31 werden durch ein Circumflex und *Zeichen* + `'@'` dargestellt. So wird z.B. 0 zu `^@`, 1 zu `^A` usw. Zeichen von 32 bis 126 werden als ihre ASCII-Repräsentation dargestellt, das Zeichen 127 als `^?`. Die Zeichen von 128 bis 255 werden als `M-` und der Darstellung von *Zeichen* − 128 dargestellt (wobei hier die Ausnahmen für Zeilenvorschub und Tabulator nicht vorkommen). 138 wird zu `M-^J`, 139 zu `M-^K` usw. usf.
Anmerkung: Sie können davon ausgehen, dass Ein- und Ausgabe im ASCII-Zeichensatz erfolgen.

Testen

Testen Sie Ihr Programm mit mehreren Eingabefiles. Erstellen Sie z.B. ein Testfile *t1* mit folgendem Inhalt (`<TAB>` entspricht hier dem Tabulator-Zeichen):

```
<TAB>blabla
blub
```

Aufruf:

```
./mycat -vET < t1
```

Ausgabe:

```
^Iblabla$
blub
```

Als Testfiles bieten sich auch beliebige binäre Dateien an (z.B. `mycat.o`). Die Ausgabe sollte dann mit der Ausgabe des Programms `cat` mit den entsprechenden Optionen übereinstimmen.

Aufgabenstellung B

Implementieren Sie ein System, das es ermöglicht, eine Datei aus mehereren anderen Dateien zusammenzusetzen.
Das System besteht aus zwei Prozessen. Der erste Prozess ("Server") wird im Hintergund gestartet. Danach durchsucht er die Message Queue nach Daten, die in die angegebene Datei eingefügt werden sollen. Der Server terminiert, sobald die angegebene Dateigröße erreicht wurde.

```
SYNOPSIS:
    gluefile -s <size> <file>
```

Der zweite Prozess ("Client") liest Daten und weist den Server über die Message Queue an, diese an der angegebenen Position einzufügen.

```
SYNOPSIS:
    insertfile -o <offset> [<file>]
```

Anleitung

Da Nachrichten beim Lesen konsumiert werden und es aufwändig zu verfolgen ist, welche Positionen der Datei bereits beschrieben wurden, wenn die Daten ungeordnet geschrieben werden, ist es sinnvoll, den Messagetyp dazu zu verwenden, um die Position anzugeben. Der Server beginnt bei der kleinstmöglichen Position und empfängt eine entsprechende Nachricht. Danach erhöht er die Position jeweils so, dass er nur Daten empfängt die unmittelbar nach der äußersten beschriebenen Position oder davor eingefügt werden sollen (es können also Teile der Ausgabedatei eventuell mehrfach beschrieben werden).
Ignorieren Sie Nachrichten die dazu führen würden, dass über die angegebene Größe hinaus geschrieben werden würde. Der Client muss nicht auf eine Bestätigung des Servers warten, sondern soll unmittelbar nach erfolgtem Senden der Daten terminieren.
Achten Sie darauf, dass das Server-Programm auch bei der Beendigung mittels eines der Signale `SIGINT`, `SIGQUIT` bzw. `SIGTERM` sämtliche Ressourcen wieder freigibt.

Testen

Mit dem Programm `split` können Sie Dateien in andere Dateien teilen; mit dem Programm `wc` die Größe einer Datei feststellen.
Erstellen Sie also eine beliebige Datei, z.B. `test.txt` und teilen Sie dann die Datei mit z.B. `split -b 100 test.txt` in einige Teile auf. Starten Sie den Server (`./gluefile -s ‘wc -c < test.txt‘ test2.txt &`) und fügen Sie die Bestandteile wieder zusammen (z.B. `./insertfile -o 100 xab; ./insertfile -o 0 xaa`). Testen Sie ihr Programm auch mit ungültigen Eingaben!