182.023 Systemnahe Programmierung
2. Test

29. Juni 2010

KNr. | MNr. | Zuname, Vorname

Ges.)(80)	1.)(25)	2.)(25)	3.)(25)	4.)(30)	Zusatzblätter:

Allgemeines

Kopieren Sie sich die Angaben jedes Beispiels mit dem Shellkommando
`$ fetch <1|2|3>`
in Ihr Homeverzeichnis, wobei 1, 2, 3 die Nummer des Beispiels ist, das sie in Folge bearbeiten möchten. Sie können `fetch` mehrmals ausführen. Eventuelle Änderungen werden (`<Datei>`→`<Datei>~1~`) gesichert.

Wenn Sie Ihre Lösung prüfen wollen, führen sie
`$ deliver <1|2|3>`
aus. Alle notwendigen Dateien werden in ein Abgabeverzeichnis kopiert und dort geprüft. Sie erhalten binnen weniger Sekunden Ihre vorläufige Bewertung. Sie können `deliver` beliebig oft ausführen. Sind Sie mit der Bewertung zufrieden, melden Sie es einem Assistenten, der das Ergebnis entgegen nimmt.

Übrig gebliebene Resourcen (inklusive der Semaphore und des Shared Memory) können sie jederzeit mit dem Kommando
`$ cleanup`
löschen.

Achten sie darauf, das Sie die Ergebnisse möglichts bald melden. Am Ende der 80 min kann es eventuell zu Verzögerungen kommen

Sie können den Prüfungsbogen verwenden, um in der Vorbereitungszeit Notizen zu machen. Diese Eintragungen werden nicht bewertet.

1 Fork, Exec and Unnamed Pipes (25)

Erstellen Sie das Programm `count`, indem Sie die Datei `count.c` vervollständigen.

```
Usage: ./count inputfile outputfile
```

`count` schreibt für jede von der Eingabedatei `inputfile` gelesene Zeile die Anzahl der Zeichen in dieser Zeile (inklusive Zeilenumbruch) in die Ausgabedatei `outputfile`. Die Zeichenanzahl wird dabei mit Hilfe des bereits implementierten Programms `encode` im

```
$ ./encode <<EOF > enc.txt    $ cat enc.txt
> 1                           I
> 2                           II
> 5                           V
> abcd                        Keine Dezimalzahl: 'abcd'
EOF
-----------------------------------------
$ ./count input.txt output.txt
$ cat input.txt               $ cat output.txt
>                             I
> a                           II
> abcd                        V
```

Abbildung 1: Beispielaufruf von encode und count

römischen Zahlensystem dargestellt. `encode` gibt für jede von der Standardeingabe gelesene Zahl die dazugehörige römische Zahl aus, also beispielsweise "IX" für "9" und "XI" für "11".

`count` soll mit Hilfe von Unnamed Pipes mit `encode` kommunizieren. Weiters muss die Standardausgabe von `encode` zur angegebenen Ausgabedatei umgeleitet werden.

Implementierung Das Programm `count` erstellt zu Beginn eine Unnamed Pipe, und erzeugt dann mit Hilfe von `fork(2)` einen neuen Prozess.

Im Kindprozess wird die Standardeingabe zum Leseende der Unnamed Pipe umgeleitet. Weiters wird die Standardausgabe mit Hilfe von `dup2(2)` so umgeleitet, dass die Ausgabe in der spezifierten Ausgabedatei erfolgt. Schliesslich soll der Kindprozess durch das `encode` Programm ersetzt werden, z.B. mit Hilfe von `execlp(3)`.

Der Vaterprozess liest zeilenweise von der spezifierten Eingabedatei, solange bis das Ende der Eingabedatei erreicht wird. Für jede Zeile wird die Anzahl der Zeichen in der jeweiligen Zeile berechnet (`strlen(3)`). Dann wird die Zahl, gefolgt von einem Zeilenumbruch (`'\n'`) in das Schreibende der Unnamed Pipe geschrieben. Nachdem das Ende der Eingabedatei erreicht wurde, schließt der Vaterprozess das Schreibende der Pipe, und wartet mit Hilfe von `waitpid(2)` auf den Kindprozess.

Empfohlene Bibliotheksfunktionen

- `pipe(2)` zum Erstellen der Pipe
- `fork(2)` um einen neuen Prozess zu erzeugen
- `execlp(3)` um den Kindprozess durch `encode` zu ersetzen
- `dup2(2)` um die Standardeingabe und die Standardausgabe zu verbiegen
- `fdopen(3)` um einen File Stream für das Schreibende der Unnamed Pipe zu öffnen
- `fgets(3)` um eine Zeile von der Eingabedatei zu lesen

- `fprintf(3)` um in die Pipe zu schreiben
- `fclose(3)` und `close(2)` zum Schließen der File Streams und File Deskriptoren
- `waitpid(2)` um auf die Beendigung des Kindprozesses zu warten
- Verwenden sie die bereits implementierte Funktion `bail_out` um das Programm im Fehlerfall zu beenden

2 Synchronisation (25)

Gegeben ist folgendes Synchronisationsbeispiel: Ein Server erledigt Sortieraufgaben für mehrere Clients. Server und Client kommunizieren dabei über ein Shared Memory. Ein Client schreibt eine unsortierte Folge von Zahlen in das Shared Memory, die der Server sortiert. Anschliessend gibt der Client die sortierte Folge aus. Dazu müssen Client und Server den Zugriff auf das Shared Memory synchronisieren, wobei folgendes zu beachten ist.

1. Ein Client darf erst dann eine neue Zahlenfolge in das Shared Memory schreiben, wenn der vorige Client seine sortierte Folge vollständig ausgelesen hat.
2. Der erste Client darf nach dem Start des Servers sofort in das Shared Memory schreiben.
3. Der Server darf mit dem Sortieren erst dann beginnen, wenn der Client die Zahlenfolge vollständig in das Shared Memory geschrieben hat.
4. Der Client darf die Zahlenfolge erst dann auslesen, wenn der Server die Sortierung abgeschlossen hat.

Sie können das Beispiel entweder mit Semaphoren (Beispiel 2.1) oder mit Sequencern und Eventcountern (Beispiel 2.2) lösen. In beiden Fällen ist der Code des Clients bereits vorgegeben und darf nicht modifiziert werden. **Ihre Aufgabe ist es, den Server zu implementieren.** Beachten Sie, dass der Server sowohl für das Anlegen als auch das Initialisieren aller Synchronisationmechanismen (Semaphoren *oder* Sequencer/Eventcounter) verantwortlich ist.

Hinweise:

- Sie können zwischen Beispiel 2.1 *oder* Beispiel 2.2 wählen und müssen nicht beide realisieren!
- Verwenden Sie eine minimale Anzahl an Synchronisationskonstrukten. Die Synchronisation ist im wesentlichen durch den Code des Clients bereits vorgegeben.
- Achten Sie auf das richtige Anlegen, Initialisieren und Freigeben von Synchronisationsmechanismen und Shared Memory.
- Achten Sie auf die korrekte Synchronisation: Ein Server bedient mehrere gleichzeitig gestartete Clients. Weitere gestartete Server sollen mit einem Fehler terminieren.
- Verwenden Sie den vorgegebenen Signalhandler zum Freigeben der Ressourcen und zum Beenden des Servers.
- Verwenden Sie UNIX-konforme Exitcodes.
- Verwenden Sie die Semaphor (sem182.h) und Sequencer/Eventcounter (seqev.h) Bibliotheken der Laborübung.

2.1 Semaphoren

Führen Sie `fetch 2` aus, um die erforderlichen Dateien des Beispieles in ihr Arbeitsverzeichnis zu kopieren. Führen Sie `deliver 2` aus, um Abgaben durchzuführen. Sie können jederzeit mit `cleanup` alle belegten Ressourcen (Semaphoren, Eventcounter, Sequencer, Shared Memory) freigeben.

2.2 Eventcounter und Sequencer

Führen Sie `fetch 3` aus, um die erforderlichen Dateien des Beispieles in ihr Arbeitsverzeichnis zu kopieren. Führen Sie `deliver 3` aus, um Abgaben durchzuführen. Sie können jederzeit mit `cleanup` alle belegten Ressourcen (Semaphoren, Eventcounter, Sequencer, Shared Memory) freigeben.