Praktikum:
Entwicklung von Kommunikationssystemen (SoSe 2000)

Prof. Dr. Reinhard Gotzhein, Philipp Schaible

Aufgabe 5 (drei Wochen: Abgabe 27.06.00)

And now for something completely different (Monty Python).

Nachdem das Kommunikationssystem spezifiziert worden ist, wollen wir uns seiner Implementierung zuwenden. In dieser Aufgabe sollt Ihr Euch zunächst das Grundwissen verschaffen, welches nötig ist, um aus dieser Spezifikation automatisch C-Code zu generieren und Komponenten per UDP miteinander kommunizieren zu lassen. Die erworbenen Kenntnisse werden gleich zur Implementierung des Servers eingesetzt.

Aufgaben

a) ***(Codegenerierung, Erzeugen eines lauffähigen Programms)***

Kopiert die Dateien unter *sep1:~eksys/CODEGEN* in Euer Arbeitsverzeichnis. Die Executables geben Euch einen groben Überblick, wie die Programme später aussehen sollen.

i) *(Signale ohne Parameter)*

Seht Euch die PingPong-Spezifikation mit SDT an. Das dargestellte SDL-System funktioniert wie folgt: Bei jedem tick-Signal, das vom Environment kommt, schicken sich die Blöcke ein *ping*- bzw. ein *pong*-Signal. Die Signale haben keine Parameter. Generiert für dieses SDL-System entsprechenden C-Code und vervollständigt die entsprechenden Environmentfunktionen so, daß ein ausführbares und funktionstüchtiges Programm entsteht. Das tick-Signal soll dabei z.B. nach Eingabe eines Zeichens über die Tastatur erzeugt werden. Dabei ist zu beachten, daß die Tastatureingabe mit *scanf* oder *getchar* blockierend ist. Unter *sep1:~eksys/NonBlockingKeyboardInput/* ist ein Beispiel für eine nicht-blockierende Tastatur-Abfrage.

Beachtet bei der Codegenerierung folgendes:

1. Im Make-Dialogfenster sollten folgende Optionen aktiviert werden: ***Cadvanced, Full, No, As defined, Generate environment header file, Generate environment functions, Generate signal number file, Generate makefile, Use standard kernel Application***.

2. Diese erste Compilierung liefert einen Fehler, weil das Environment-File Beispiel1_env.c im Makefile Beispiel1.m nicht berücksichtigt wurde. Hier müßt Ihr selbst Hand anlegen und das Makefile entsprechend erweitern (am Besten die Dateien sinnvoll umbenennen, da bei erneutem Anstoßen des Make-Vorgangs Eure Änderungen schnell überschrieben sind). Um ein make-Kommando in der Shell auszuführen, solltet ihr folgende Zeilen an den Anfang Eures Makefiles setzen:

   sdtdir=/sep05/sfb_tools_05/sdt/sdt/sdtdir/sunos5sdtdir
   sctdir=$(sdtdir)/SCTAAPPLCLENV

3. Ein nun durchgeführtes *make* klappt immer noch nicht, weil das Environment-File noch unvollständig ist. Schreibt die Funktion *xInEnv* und löscht ggf. sonstige Zeilen, in denen Fehler auftreten. *xInEnv* sollte es erlauben, von außen (z.B. Tastatur) tick-Signale ins System zu schicken. Als Einstieg in die Materie sei auch [1] und [2] empfohlen; Wenn Ihr noch nie mit make gearbeitet habt, sind auch die ersten paar Seiten von [3] recht sinnvoll!

ii) *(Signale mit Parametern)*

Bei *SDL-Rechner*, dem zweiten Demosystem, kommen nun einerseits Parameter und andererseits die Ausgabe von Signalen und Signalparametern ins Spiel. Das SDL-System stellt einen einfachen Rechner dar, der zwei Integers und einen Operator vom Environment entgegennimmt und das berechnete Ergebnis zurück an das Environment schickt. Bringt zunächst das System zum Laufen, ohne Euch um die Parameter der Signale zu kümmern. Zum Testen kann man #CODE-Anweisungen ins SDL-System einbauen (s. FAQ´s). Ordnet dann den Signalen Parameter zu und schreibt passende *xInEnv*- und *xOutEnv*-Funktionen. Relevante Informationen dazu findet man in FAQ´s und im erzeugten File *Add.ifc*.

b) ***(Einstieg in UDP-Netzwerkprogrammierung mit Sockets unter Unix)***

Als Einstieg in die Materie sei [Gau87] empfohlen (wurde bereits mit der Aufgabe4 ausgeteilt.) Unter sep1:*~ektsys/UDP/* liegen auch zwei Beispiele. Eure Aufgabe ist nun, die Spezifikation des Systems aus der Aufgabe a)(ii) dahingehend zu erweitern, daß dieses einen Server abgibt, den man von beliebigen Rechnern aus erreichen kann. (Ein Vorschlag, um dies zu ermöglichen, ist, weitere Signalparameter für den Rechnernamen und die Portnummer der Clients einzuführen.) Die Kommunikation soll über UDP mit Sockets erfolgen.

c) ***(Vorbereitung des Servers zur Codegenerierung):***

Nachdem Ihr eine Einführung sowohl in Codegenerierung mit SDT als auch in Netzwerkprogrammierung mit Sockets erfolgreich hinter Euch gebracht habt, seid Ihr nun in der Lage, aus der SDL-Spezifikation, die Ihr in Aufgabe 4 erstellt habt, ein lauffähiges Verteiltes System zu erstellen. In dieser Aufgabe soll zunächst nur der Server implementiert werden. Die Implementierung der Clients und deren Anbindung an die Applikationen wird in der nächsten Aufgabe vorgenommen. Bevor man für den Server Code generieren kann, muß die entsprechende Entwurfsspezifikation vorbereitet werden. Die notwendigen Veränderungen betreffen folgende Punkte:

i) Bis jetzt sind wir von einer festen Anzahl der Clients, die der Server bedienen soll, ausgegangen. Für eine automatische Codegenerierung ist diese vereinfachende Annahme in der Spezifikation nicht mehr vertretbar.

ii) Der Server soll über genau eine, allen Clients bekannte, Adresse (ip + port) erreichbar sein. Deswegen soll der Server auch in SDL mit nur einem SDL-Kanal zur Umgebung spezifiziert werden. (Außerdem erleichtert dies auch die Codegenerierung)

iii) Zur Codegenerierung wird die sog. “light integration” benutzt. (aus einer SDL-Spezifikation wird genau ein ausführbares Programm erstellt.) Dazu teilt man das spezifizierte System in unabhängige Laufzeit-Komponenten auf und erstellt eine SDL-Systemspezifikation für jede dieser Komponenten. Da wir im Moment nur den Server betrachten, wird genau eine SDL-Systemspezifikation, und zwar die des Servers, erstellt.

d) ***(Codegenerierung für den Server)***

Generiert anschließend Code für den Server wie in Aufgabe a). Da der Code automatisch generiert wird, kann man davon ausgehen, daß es korrekt funktioniert und deswegen auf den Komponententest für den Server verzichten. Was auf jeden Fall getestet werden soll, sind die von Hand programmierten Umgebungsfunktionen wie *xInEnv* und *xOutEnv*.