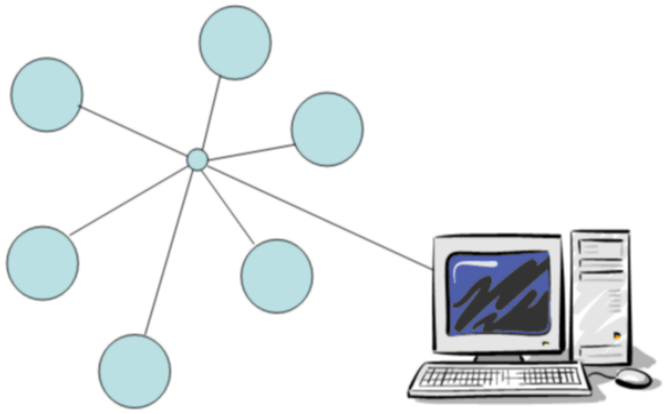
Projekt IPC

Interprozess-Kommunikation

Frühlingssemester 2011
M. Thaler, J. Zeman, Rev. 2011



Inhaltsverzeichnis

1 Einführung

Jedes Multitasking Betriebssystem stellt Mechanismen für die Kommunikation zwischen lokalen und entfernten Prozessen zur Verfügung. Diese Mechanismen unterstützen die Synchronisation von Prozessen sowie den Austausch von Daten. Dazu gehören Semaphore, Mutexes, Shared Memory, Message Queues, etc..

Die wichtigsten Mechanismen haben Sie bereits in der Theorie kennen gelernt. Das vorliegende Projekt soll Ihnen helfen, diese Synchronisations- und IPC-Mechanismen besser zu verstehen und in der Praxis anwenden zu können. Wir werden uns auf Mechanismen unter UNIX (Linux) beschränken, andere Betriebssysteme stellen jedoch gleiche oder sehr ähnliche Mechanismen zur Verfügung.

Als Aufgabenstellung werden Sie eine Anwendung aus dem Bereich der Prozessüberwachung und -Steuerung der Firma Green Systems & Co. lösen. Dabei geht es um das Erfassen und Aufbereiten von Sensorsignalen zur Steuerung und Regelung einer Produktionsanlage.

1.1 Ziele

- die wichtigsten Synchronisations- und IPC-Mechanismen kennenlernen
- die wichtigsten Synchronisations- und IPC-Mechanismen unter Unix/Linux anwenden können
- eine einfache verteilte Applikation mit Synchronisations- und IPC-Mechanismen implementieren können

1.2 Voraussetzungen

Im Praktikum zu Prozessen haben Sie die Funktionen zur Prozesserzeugung und -Steuerung kennen gelernt. Makefiles, C/C++ Programmierung, etc. kennen Sie aus der Einleitung. In der Theorie wurden Möglichkeiten und Konzepte zur Synchronisation und Interprozesskommunikation vorgestellt. Alles weitere werden Sie beim Durcharbeiten der Aufgabenstellung kennen lernen.

1.3 Was wir erwarten

Wir erwarten, dass Sie das Projekt vollständig und selbständig in Ihrer Projektgruppe implementieren. Wir werden das Resultat mit Hilfe von Testprogrammen und manuell überprüfen.

Ihre Arbeit bewerten wir mit einer Note, dabei berücksichtigen wir die Vollständigkeit und Korrektheit der Implementation. Wir werden selbstverständlich auch überprüfen, ob Sie selbständig gearbeitet haben, d.h. Sie müssen uns Ihren Source Code zur Verfügung stellen. Details zur Abgabe des Projektes werden wir Ihnen zusätzlich abgeben.

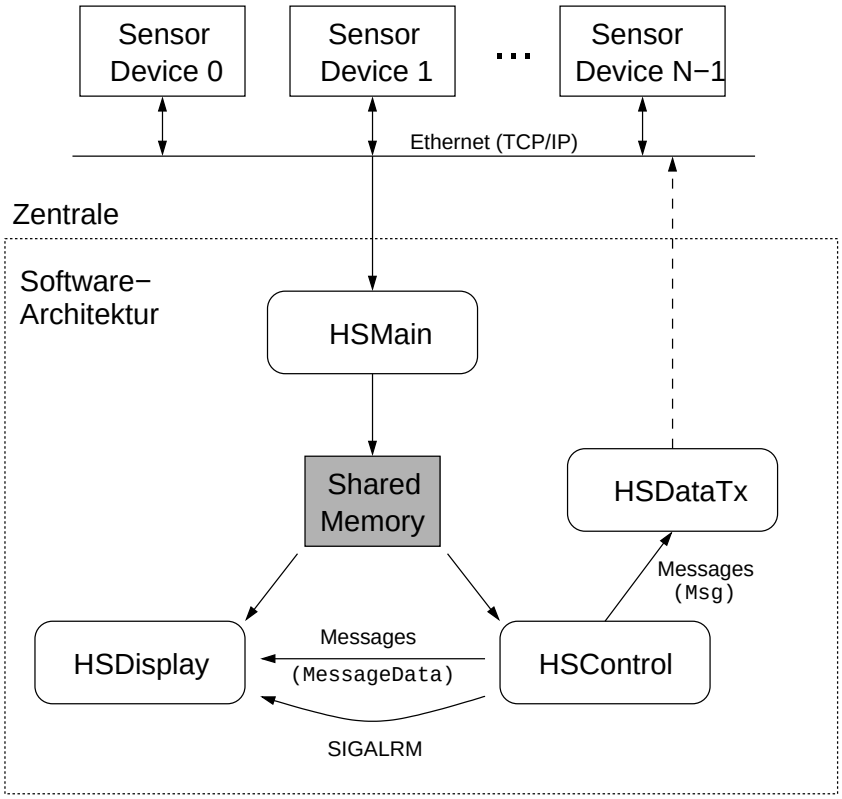
1.4 Literatur

[1] Helmut Herold, Linux-Unix Systemprogrammierung, Addison-Wesley, 3. Auflage, 2004.
[2] Online Manual Pages zu Synchronisations- und IPC-Funktionen.

2 Aufgabenstellung

Die Firma Green Systems baut Mess- und Regeltechniksysteme für Produktionsanlagen und ist Spezialist für vernetzte Grosssysteme. Bei einer neuen Anlage müssen an mehreren Orten Messwerte erfasst werden und zusammen mit einem entsprechenden Referenz- resp. Sollwert an ein zentrales Verarbeitungssystem übermittelt werden. Das zentrale System verarbeitet die Daten und regelt damit die Anlage.

Die folgende Figur gibt einen Überblick zum Aufbau des gesamten Systems:



Die Software der Zentrale selbst besteht aus den vier Softwarekomponenten HSMain, HSDiaplay, HSControl und HSDataTx. Die Komponenten sind als selbständige Prozesse organisiert und kommunizieren miteinander über verschiedene IPC-Mechanismen. Ihre Aufgabe ist es diese vier Komponenten (Prozesse) gemäss unten stehender Spezifikation zu implementieren.

Wichtig

Für Semaphore, Shared Memory und Message Queues müssen SYSTEM V IPC-Mechanismen verwendet werden.

2.1 Sensor Devices

Die *Sensor Devices* sind Teil der Produktionsanlage und müssen von Ihnen nicht implementiert werden, Sie erhalten dazu ein Simulationsprogramm. Allerdings müssen Sie die Funktionsweise der Devices verstehen, um das Hauptprogramm `HSMain` implementieren zu können.

Die Sensor Devices messen jeweils einen Prozessparameter und übertragen ihn zusammen mit

einem Referenzwert an das zentrale System über TCP/IP-Sockets (Port 44444). Die Sensor Devices können ebenfalls Daten über einen TCP/IP-Sockets (Port 55555) empfangen und damit die Produktionsanlage steuern (*dieses Feature müssen Sie nicht implementieren*).

2.1.1 Messdaten und Datenübertragung

Die einzelnen Sensor Devices übermitteln folgende Daten, die in der Datenstruktur `SensorData` in `defs.h` definiert sind:

Variable	Typ	Beschreibung
`deviceID`	`unsigned`	die Sensor Nummer, die Sensoren sind von 0 an aufsteigend nummeriert und dienen der Identifikation eines Sensors
`sequenceNr`	`unsigned`	nach jedem übertragenen Datenwert wird die Sequenznummer inkrementiert, damit lässt sich die korrekte Reihenfolge der Messwerte ermitteln
`valIS`	`float`	der aktuelle IST-Wert des Sensors
`valREF`	`float`	der SOLL-Wert des Sensors
`status`	`int`	Information zum Zustand des Sensors, (0: arbeitet korrekt, -1: ist ausgefallen)

Die Sensor Devices tasten ihre Messfühler zwar periodisch ab, sind aber sonst nicht synchronisiert. Deshalb ist es nicht garantiert, dass die Daten der verschiedenen Devices in einer strikt vorgegebenen Reihenfolge abgeschickt werden. Jeder Device generiert aber eine fortlaufende Sequenznummer zu jedem Daten- resp. Messwert und sendet sie in dieser Reihenfolge. Zusätzlich wird mit einem verteilten Snooping Verfahren dafür gesorgt, dass alle Devices ihre Daten mit Sequenznummer n abgeschickt haben, bevor Daten mit Sequenznummer $n+2$ übertragen werden.

Hier ein Bespiel mit vier Devices ($N = 4$), wobei die Notation $D_{deviceID@sequenceNr}$ verwendet wird:

$$\ldots \quad D_{2@3} \quad D_{3@3} \quad D_{1@3} \quad D_{1@4} \quad D_{4@3} \quad D_{1@4} \quad D_{2@3} \quad D_{1@5} \quad D_{2@4} \quad D_{4@4} \quad \ldots$$

Dies ist eine zulässige Sequenz, weil die Daten von allen Devices mit Sequenznummer 3 abgeschickt worden sind, bevor der Datenwert von Device Nr. 1 mit Sequenznummer 5 übertragen wird.

2.1.2 Rücksetzen der Sensoren

Alle Sensoren lassen sich gemeinsam zurücksetzen oder neu starten, in diesem Fall beginnen die Sequenznummern für jeden Sensor wieder bei 0.

2.1.3 Ausfall von Sensor Devices

Fällt ein Sensor Device aus, wird dies vom verteilten Snooping Verfahren detektiert. Ein anderer Sensors übernimmt dann die Verantwortung und sendet für den ausgefallenen Sensor Device

Datenpakete mit der entsprechenden `deviceID` und Sequenznummer jedoch mit Statuswert -1 (Referenz- und IST-Werte sind auf Null gesetzt). Wird der Sensor Device repariert, übernimmt er wieder das Übermitteln der Sensor Daten und der *Aushilfs-Device* gibt selbständig die Verantwortung ab.

2.1.4 Anzahl Sensor Devices

Die aktuelle Anzahl N von Sensor Devices ist beim Hochfahren des Systems bekannt, muss aber als Parameter an die Systemprozesse übergeben werden. Die maximale Anzahl Sensordevices ist in `defs.h` mit `SENSOR_MAX_NUM` definiert, d.h. benötigte Arrays, etc. können mit dieser Grösse initialisiert werden. Die Sensor Devices sind immer von 0 bis $N-1$ nummeriert (deviceID).

2.2 Die Softwarekomponenten

2.2.1 Das Shared Memory

Das Quality Assurance Departement der Firma Green hat festgelegt, dass die Daten im Shared Memory als Array von Sensor Device Strukturen abgelegt sein müssen und zwar in aufsteigender Reihenfolge der Sensor ID's (`deviceID`).

2.2.2 Das Hauptprogramm `HSMain`

Funktion

`HSMain` ist der Hauptprozess des gesamten Systems. Er startet die Prozesse resp. Programme `HSDisplay` und `HSControl` mit einer der exec-Varianten und verwaltet die Daten der Sensor Devices, sowie sämtliche IPC-Ressourcen. Die Daten von den Sensor Devices müssen nach Device ID's sortiert und mit konsistenten, d.h. gleichen Sequenznummern in einem Shared Memory Bereich abgelegt werden, der Zugriff auf dieses Shared Memory muss mit Semaphoren synchronisiert werden.

Achtung: bei einem Restart der Sensoren beginnen die Sequenznummern erneut bei 0, dies muss von `HSMain` korrekt behandelt werden können (einzelne Sensoren können nicht zurückgesetzt werden).

`HSMain` ist für das korrekte Arbeiten des gesamten Software-Systems verantwortlich, d.h. er startet `HSDisplay` und `HSControl` und überwacht dauernd, ob die beiden Prozesse noch arbeiten. Sollte einer dieser Prozesse aus irgendwelchen Gründen terminieren, muss das System wieder in einen lauffähigen Zustand gebracht werden, d.h. der Prozesse muss neu gestartet werden. Wenn `HSMain` selbst abgebrochen wird, muss das gesamte System geordnet heruntergefahren werden. Dazu sendet `HSMain` das Signal `SIGUSR1` an `HSDisplay` und `HSControl` und räumt sämtliche allozierten Ressourcen auf.

Hinweis: verwenden Sie für die Überwachung der Programme `HSDisplay` und `HSControl` nicht das Signal `SIGCHILD` sondern die Systemfunktion `waitpid(....)`.

Parameter

Input Parameter: die aktuelle Anzahl Sensor Devices.

Daten

Sämtliche Daten der Sensor Devices müssen sortiert nach Device ID's und mit jeweils gleicher Sequenznummer im Shared Memory abgelegt werden (Device ID's von 0 bis $N-1$). Die Verzögerung zwischen Empfang und Ablegen der Daten im Shared Memory muss so kurz wie möglich gehalten werden.

IPC

IP-Sockets	Kommunikation mit Sensor Devices, `HSMain` ist Server, Port 44444
Shared Memory	gemeinsam mit `HSDisplay` und `HSControl`, Zugriffssynchronisation mit Semaphoren
Message Queues	Message Queue (aufsetzen und initialisieren) für den Austausch von Messages zwischen `HSControl` und `HSDisplay` sowie `HSDataTx` (Definitionen in `defs.h` müssen verwendet werden)
Signale	`SIGTERM` und `SIGINT` → sendet `SIGUSR1` an `HSDisplay` und `HSControl`, räumt IPC-Ressource auf und terminiert selbst

2.2.3 Das Display-Programm `HSDisplay`

`HSDisplay` ist für die Datenausgabe auf dem Bildschirm zuständig, ist ein eigenständiger Prozess und wird von `HSMain` gestartet. Die Datenausgabe soll alle 0.5 Sekunden aufdatiert werden, wobei auch Daten von ausgefallenen Sensoren angezeigt werden sollen.

Funktion

`HSDisplay` muss sowohl die Sensor Daten aus dem Shared Memory als auch die Daten des Steuerprozesses HSContrl (über eine Message Queue empfangen) darstellen.

Die Daten (`valIS`, `valREF`) der aktuellen Sensor Devices müssen nach folgendem Format als Balkengraphik dargestellt werden (keine Skalierung notwendig):

```
Device 3 @ 4: 0  V act   ......................
                 V ref   ----------------------
```

Das Beispiel zeigt Device Nr. 3 mit Sequenznummer 4, Statuswert 0 und einem Messwert `V`, der z.Z. etwas kleiner als der Referenzwert ist.

`HSControl` sendet zudem Daten zum aktuellen Zustand des System mit Hilfe von Messages, die wie folgt unterhalb der Sensor Daten ausgegeben werden sollen (Beispiel):

```
sequence number         9
max diff @ device 1     11.43
status                  +++
```

Weiter meldet `HSControl` den Ausfall von Sensor Devices mit dem Signal `SIGALRM`, dabei soll in der Kopfzeile der Datenausgabe eine entsprechende Meldung angezeigt werden.

Für die Darstellung stehen in `defs.h` die beiden Macros `HomeScreen()` und `ClearScreen()` zur Verfügung.

Parameter

Input Parameter: die aktuelle Anzahl Sensor Devices.

Daten

Die Daten zu den Sensor Devices stammen aus dem Shared Memory, die Daten des Kontrollers sind Messages mit Typ 3333 (`MessageData`) definiert in `defs.h`: übertragen werden die Sequenznummer, die Device ID des Sensors mit der höchsten Abweichung zum Referenzwert und der Wert dieser Abweichung, sowie eine symbolische Meldung zum Zustand des Systems.

IPC

IP-Sockets	keine
Shared Memory	gemeinsam mit `HSMain` und `HSControl`, Zugriffssynchronisation mit Semaphoren
Message Queues	Message Queue, aufgesetzt von `HSMain`, Zugriff über gemeinsamen Key definiert in `defs.h`
Signale	terminiert mit `SIGTERM`, `SIGINT` und `SIGUSR1` gibt bei Empfang von `SIGALRM` für 2 Sekunden auf der Kopfzeile eine Fehlermeldung aus, z.B.: `---- Control Alarm -----"`

2.2.4 Das Kontrollprogramm `HSControl`

`HSControl` ist für die Steuerung des Systems zuständig, ist ein eigenständiger Prozess und wird von `HSMain` gestartet. `HSControl` sendet alle Sekunden eine Message (Type 3333) an `HSDisplay` und alle 2 Sekunden eine Message (Type 3334) an `HSDataTx`.

Funktion

`HSControl` liest die Daten aus dem Shared Memory und berechnet für alle Sensor Devices mit Status > 0 die Differenz zwischen Referenzwert und aktuellem Messwert:

$$delta[i] = valREF[i] - valIS[i] \qquad i = 0..N-1$$

sowie den Wert, der den grössten Betrag der Abweichung aufweist:

$$maxDelta = delta[i] \quad \text{mit} \quad abs(delta[i], \forall i) = maximal;$$

und die Summe der Abweichungen:

$$sum = \sum_{0}^{N-1} delta[i];$$

Mit einer Message (Typ 3333) wird alle Sekunden folgende Information an `HSDisplay` übermittelt (die Definition der Message ist in `defs.h` zu finden):

- die Device ID der Station mit der höchsten Abweichung
- die Sequenznummer
- den Status-Text
  - `+++` wenn $sum < -5$
  - `---` wenn $sum > 5$
  - `+/-` wenn $-5 \leq sum \leq 5$

Beträgt der Statuswert eines Sensor Devices -1, wird zusätzlich das Signal `SIGALRM` an `HSDisplay` gesendet.
Mit einer Message (Typ 3334) wird alle 2 Sekunden der Array $ctrl[i]$ mit Steuersignalen an `HSDataTx` übermittelt. Eine Definition der Message (`Msg`) ist in `defs.h` zu finden. Mit dieser Meldung wird auch die aktuelle Anzahl Sensor Devices (N) übermittelt. Der Array enthält bei Index i (device ID) die Abweichung des Messwertes der Station i multipliziert mit 0.3:

$$ctrl[i] = 0.3 \cdot delta[i];$$

Parameter

Input Parameter: die aktuelle Anzahl Sensor Devices und die Prozess ID (PID) des Prozesses `HSDiplay` (in dieser Reihenfolge). Die PID wird für das Senden von `SIGALRM` benötigt.

IPC

IP-Sockets	keine
Shared Memory	gemeinsam mit `HSDisplay` und `HSDisplay`, Zugriffssynchronisation mit Semaphoren
Message Queues	die von `HSMain` aufgesetzt Message Queue, Meldungen mit Typ 3333 und Typ 3334
Signale	terminiert mit `SIGTERM`, `SIGINT` und `SIGUSR1` und sendet vor dem Terminieren das Signal `SIGINT` an `HSDisplay` sendet das Signal `SIGALRM` an `HSDisplay`, wenn eine Station den Statuswert -1 übermittelt.

2.2.5 Das Datenprogramm `HSDataTx`

`HSDataTx` gibt für Testzwecke in einem ersten Schritt die von `HSControl` gesendeten Steuerdaten (Typ 3334) auf dem Bildschirm aus.

Funktion

`HSDataTx` soll als autonomer Prozess in einem eigenen Fenster *von Hand* gestartet werden können und die von `HSControl` empfangenen Daten auf dem Bildschirm ausgeben. Die Message Queue muss über den in `defs.h` definierten Key initialisiert werden. Das System soll auch arbeiten, wenn `HSDataTx` nicht läuft.

IPC

IP-Sockets	keine
Shared Memory	keines
Message Queues	die von `HSMain` aufgesetzt Message Queue, Meldungen mit Typ 3334
Signale	terminiert mit `SIGTERM`, `SIGINT`, keine weiteren Aktionen notwendig

3 How to attack the problem

Beachten und befolgen Sie folgende Punkte:

- Das zur Verfügung gestellte Programm `SensorDevices.cc` simuliert die Messstationen, muss von Ihnen übersetzt werden und wird wie folgt aufgerufen:

  ```
  SensorDevices.e AnzahlDevices ServerAdresse Portnummer
  ```

- **Wir werden die Funktionlität Ihrer Programme mit Test-Prozessen überprüfen: halten Sie sich also strikte an die Spezifikationen!**
- Studieren Sie zuerst die Aufgabenstellung gründlich und erstellen Sie einen Anforderungskatalog ... ohne Papierarbeit geht es nicht.
- Verwenden Sie unverändert und ausschliesslich die Definitionen in `defs.h` für die vordefinierten Datenstrukturen.
- Verwenden Sie nur die `System V` IPC-Mechanismen für Shared Memory, Message Queues und Semaphore zusammen mit den in `defs.h` definierten Schlüsseln (key's) um die Ressourcen anzufordern (Systemfunktion `ftok()`).
- Machen Sie sich mit den benötigten IPC-Mechanismen (Anleitungen und Manuals) vertraut. Dazu stehen auch Beispielprogramme mit Übungen für die einzelnen IPC-Mechanismen zur Verfügung.
- Implementieren Sie *Wrapper* die IPC-Mechanismen jeweils als eigenständige Module, Sie ersparen sich damit Arbeit und die Programme werden übersichtlicher.
- Realisieren Sie alle Systemkomponenten als eigenständige Prozesse und verwenden Sie folgende Namensgebung für die Prozesse:

`HSMain`	→ `HSMain.e`	`HSControl`	→ `HSControl.e`
`HSDisplay`	→ `HSDisplay.e`	`HSDataTx`	→ `HSDataTx.e`

- Erstellen Sie nun ein Realisierungskonzept und implementieren Sie das System schrittweise, z.B. `HSMain` (IP-Sockets) und `HSDisplay`, in einem zweiten Schritt `HSContrl` und `HSDataTx`. Sie dürfen selbstverständlich Softwarekomponenten aus den Übungsbeispielen verwenden und anpassen.
- Beim Erzeugen von Ressourcen und Prozessen können Fehler auftreten, die abgefangen werden müssen. Überwachen Sie immer die Rückgabewerte der System Calls und veranlassen Sie bei Nichtgelingen die entsprechenden Massnahmen (aufräumen). Beim Schliessen und Terminieren von Prozessen müssen die allozierten Ressourcen freigegeben werden. Ein allfälliges Ignorieren dieser Tatsache kann Ihr System an den *Anschlag* bringen.

Viel Spass!