Synchronisation

1 Vorbereitung

Neben der allgemeinen Vorbereitung auf diesen Praktikumstermin analysieren Sie vorab die Ihnen zur Verfügung gestellten Implementierungen, mit denen Sie den Sensortag auslesen können. Beantworten Sie folgende Fragen:

- Wie wird der reale Sensortag ausgelesen?
- Wie funktioniert der Aufruf von CLI-Befehlen im C++ Source-Code?
- Können Sie die Abfrage auch direkt auf Kommandozeile herbeiführen?

2 Erstellen einer Framework-Klasse CNamedSemaphore

Ziel ist es, eine Klasse *CNamedSemaphore* zu implementieren, die die Systemrufe zu einem benannten Semaphor nach POSIX kapselt. Die Deklaration der Klasse ist bereits vorgegeben und soll nicht verändert werden.

1. Die Headerdatei *CNamedSemaphore.h* enthält die Deklaration der Klasse. Implementieren Sie in der Datei *CNamedSemaphore.cpp die* entsprechenden Funktionen.
2. Als Fehlerbehandlung beenden Sie den Prozess, sollte ein Systemruf fehlschlagen. Hierzu finden Sie bereits die Funktion *exitproc*, die für einige Fehlerzustände bereits eine entsprechende Fehlermeldung auf dem Terminal ausgibt.
3. In den beiden Dateien finden Sie Fragen, die Ihnen eine Hilfestellung geben, was Sie bei der Implementierung bedenken müssen.
4. Überlegen Sie, welche Maßnahmen Sie zu einer besseren Fehlerbehandlung benötigten, um statt abzubrechen jeden Fehler durch Ihr Programm behandeln zu können?

3 Prozesssynchronisation mittels der Framework-Klasse

Ziel ist es, dass Sie zwei Prozesse über einen benannten Semaphor synchronisieren. Über einen zweiten Semaphor teilen die Prozesse einen gemeinsamen Zustand. Anhand des Zustands soll jeweils ein Prozess als aktiv gelten. Dieser aktive Prozess ändert den Zustand, so dass der zweite Prozess erkennen kann, dass er aktiv wird. Der jeweils aktive Prozess liest den Bewegungssensor des SensorTags und gibt die Werte aus. Markieren Sie bei der Ausgabe, ob es sich um den Eltern- oder Kindprozess handelt, so dass eine Folge aus jeweils wechselnden Ausgaben sichtbar wird (ping-pong). Zustandswechsel sollten immer synchron erfolgen. Als Hilfestellung steht Ihnen die Datei *sync_proc.cpp* zur Verfügung. Testen sie Ihr Programm erst mit Ihrer Simulation des SensorTags auf dem Labor PC und lassen Sie ihr Programm anschließend mit den beigefügten Klassen *CSensorConfiguration* und *CSensorCommunication* auf dem Raspberry Pi laufen, um den echten Sensor auszulesen.

1. Fork'en Sie einen Kindprozess vom Elternprozess in *main*. Beide Prozesse rufen dann *pingpong* auf.

2. Implementieren Sie *pingpong*. Vorgegeben sind bereits zwei Semaphoren. Der eine Semaphor (semaphore) wird zum Schutz kritischer Abschnitte benötigt. Der zweite Semaphor enthält den gemeinsamen Zustand der beiden Prozesse. Der Zustand wechselt wie in Abb 1 beschrieben. Die Zustände sind bereits als `#define` vorhanden.

3. Lesen Sie nur jeweils ein Mal den Sensor aus, wenn ein Prozess die aktive Rolle übernimmt.

4. Verwenden Sie ein geeignetes C-Konstrukt zur Implementierung der State Machine.

5. Verwenden Sie den Übergabeparameter von *pingpong*, um zwischen den auszuführenden Codeteilen von Kind- und Elternprozess zu unterscheiden.

6. Achten Sie darauf, dass Ihr Programmverhalten beim Beenden dem Zustandsgraphen entspricht.

7. Verwenden Sie notfalls Debug-Ausgaben, um das Verhalten im Detail nachvollziehen zu können. Bedenken Sie die Änderung der Zeiteigenschaften hierdurch.

8. Überlegen Sie:
   Wie müssen sich die beiden Prozesse verhalten, wenn sie nicht die aktive Rolle haben, damit sie auf einen sich ändernden Zustand reagieren können?

9. Kann es zu Deadlocks kommen? Wie sieht die Situation beim Beenden aus? Welcher Prozess terminiert wann? Kann es zu ungültigen Zuständen kommen?

10. Verwenden Sie *perf*, um eine Statistik über den Programmablauf zu erstellen. Was sagen die einzelnen Werte der Statistik aus?

11. Verwenden Sie *perf*, um eine Analyse der konsumierten Rechenzeit zu erhalten. Welche Funktion konsumiert am meisten CPU-Ressourcen?

12. Warum verwendet dieses Programmbeispiel keine Mutexes? Welche Unterschiede gibt es zwischen Mutexes und Semaphoren?

13. Kennen Sie ein Betriebsmittel, mit dem Sie eine bessere Implementierung der Aufgabenstellung erreichen können als mit einem benannten Semaphor? Welchen Vorteil können Sie so erreichen?

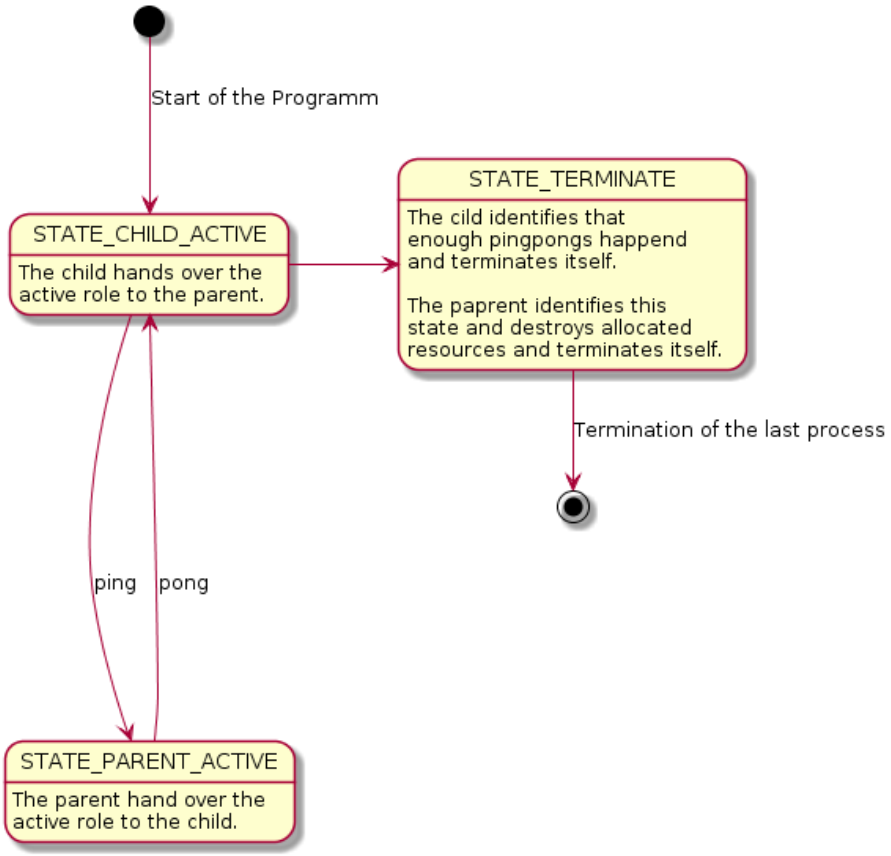

Figure 1: State Machine