h_da	Embedded SW-Praktikum	
Termin 3	Synchronisation	Prof. Akelbein

Teil 1: Erstellen einer Frameworkklasse 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?

Teil 2: Prozesssynchronisation mittels der Frameworkklasse

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 macht dies durch Ausgabe eines Zeichens auf der Konsole sichtbar, so dass eine Folge aus jeweils wechselnden Zeichen sichtbar wird (ping-pong). Zustandswechsel sollten immer synchron erfolgen. Als Hilfestellung steht Ihnen die Datei sync_proc.cpp zur Verfügung.

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 dem Zustandsgraphen (State Machine) unten beschrieben. Die Zustände sind bereits als #define vorhanden.
3. Geben Sie nur jedes Mal ein Zeichen 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?

Tipps:

Synchrone Ausgabe

Für iostreams leeren Sie mit flush den internen Puffer.

Semaphoren

Benannte Semaphoren müssen Sie wieder explizit löschen (rm …), sollte Ihr Programm nicht kontrolliert beendet worden sein.

CTRL-C

Mit dieser Kombination senden Sie ein Signal zum terminieren des Programms.

kill -9 pid

Mit diesem CLI-Tool senden Sie ein Kill-Signal, welches den angegebenen Prozess ohne weitere Prüfung beendet.

htop

Dies zeigt Ihnen die aktuell laufenden Prozesse an. Sie können Prozesse suchen und diese per F-Tasten terminieren. Verwenden Sie hierfür eine xterm-Konsole.

top

Zeigt Ihnen die aktuell laufenden Prozesse nach CPU-Bedarf sortiert an.

dhex

Ein einfacher Hex-Editor, mit Sie Inhalte von Dateien oder auch von Semaphoren lesbar anzeigen lassen können.

perf

Mit `perf stat ./my_prog` erstellen Sie eine Statistik während der Laufzeit. Die Man-Page erhalten Sie mit man perf-stat.

Mit `perf record ./my_prog` zeichnen Sie einen Report über die verwendeten CPU-Ressourcen auf. Mit `perf report` zeigen Sie den erstellten Report an.

Zustandsänderungen zwischen den beiden Prozessen als State Machine: