Praktikum Betriebssysteme – Übung 7

Aufgabe 1 – Verständnisfragen

a) Warum ist die Synchronisation mit Hilfe von einfachen Sperrvariablen, ohne spezielle Hardware- oder Betriebssystemunterstützung, problematisch?

b) Welche Forderungen werden an einen guten Algorithmus zum wechselseitigen Ausschluss gestellt?

c) Welche Anwendungsfälle für Semaphore kennen Sie?

d) Wie unterscheidet sich das Konzept des Semaphor vom Mutex?

e) Welche Vorteile bieten Monitore gegenüber Semaphoren?

Aufgabe 2 – Beispiel Synchronisationsproblem

Laden Sie das Beispielprogramm aus der Vorlesung (→ git) herunter uns versuchen Sie den Quellcode zu verstehen. Das Programm soll ein Bankkonto (Kontostand in der globalen Variable `balance`) simulieren, dessen Kontostand nicht negativ werden darf. Mehrere Threads versuchen abwechselnd bzw. gleichzeitig zufällige Geldbeträge auszugeben, sofern der Kontostand hoch genug für die Ausgabe ist.

Übersetzen Sie das Programm und führen Sie es aus. Lassen Sie das Programm mehrmals mit nur einem Thread und mit mehreren Threads laufen. Was stellen Sie fest?

Aufgabe 3 – Synchronisation mittels Mutex

Verwenden Sie eine Mutex-Variable um den kritischen Abschnitt aus Aufgabe 2 (→ git) zu schützen. Sie sollten erkennen, dass kein negativer Kontostand mehr auftreten kann, falls Sie den kritischen Abschnitt korrekt implementieren.

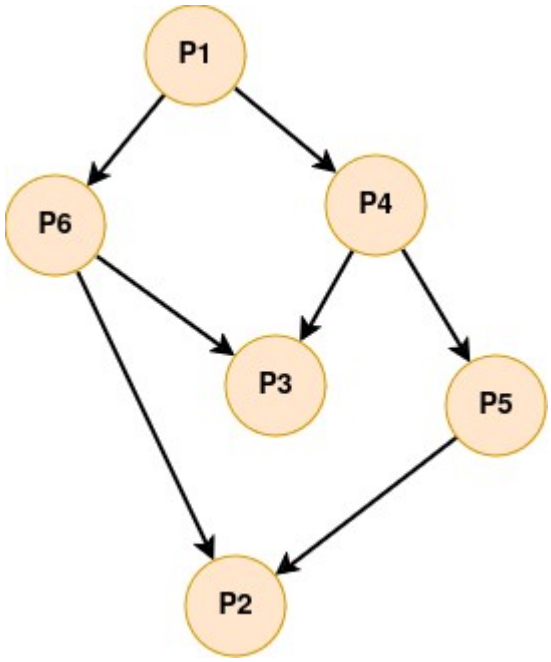
Sehen Sie sich dazu die Manpages zum Thema Mutex (`pthread_mutex_init`, `pthread_mutex_lock`, `pthread_mutex_trylock`, `pthread_mutex_unlock`, `pthread_mutex_destroy`) an.

Aufgabe 4 – Synchronisation mittels Semaphore

Verwenden Sie POSIX Semaphore um den kritischen Abschnitt aus dem Beispielprogramm aus Aufgabe 2 (→ git) zu schützen. Haben Sie eine Idee, wie man den kritischen Abschnitt kleiner halten und die Bildschirmausgabe außerhalb realisieren kann?

Hinweis: Die Manpage `sem_overview` gibt einen Überblick über POSIX Semaphore und verweist auf die Manpages der relevanten Semaphore-Funktionen. System V Semaphore behandeln wir an dieser Stelle nicht weiter. Falls Sie daran interessiert sind, sehen Sie hier: Buch Linux-UNIX-Programmierung.

Aufgabe 5 – Vorrangrelation



Betrachten Sie das im Bild gegebene Prozesssystem.

a) Geben Sie die notwendigen Semaphore und deren Schaltung für die Vorrangrelation (in Pseudocode) an.

b) Versuchen Sie die Vorrangrelation in C umzusetzen. Starten Sie für jeden Prozess (P1 – P6) einen Thread und legen Sie für jede Kante einen Semaphor an. Die Threads können zum Beispiel jeweils einfach ihre ID ausgeben und sich danach wieder beenden.

Sie sollten erkennen, dass die korrekte Reihenfolge IMMER eingehalten wird, sofern Sie die Vorrangrelation korrekt implementiert haben.

Aufgabe 6 – Linux futex

Linux stellt seit 2003 einen Mechanismus namens `Futex` (Fast user-space locking) zur Verfügung um eine effiziente Synchronisation mit minimaler Kernelunterstützung (Kontextwechsel) durchzuführen. Die zuvor diskutierten `pthread_mutexes`, sowie die meisten anderen Methoden zur Synchronisation (`condvars`, `semaphores`, `rwlocks`, and `barriers`) werden in Linux mittels `Futex` implementiert.

Informieren Sie sich über futex (hier eine Auswahl an Resourcen):

→ man futex

→ https://docs.kernel.org/locking/robust-futexes.html

→ https://kernel.org/doc/ols/2002/ols2002-pages-479-495.pdf

→ https://akkadia.org/drepper/futex.pdf