J. Gutenberg-Universität Mainz
Prof. Dr. Reinhold Kröger
Fabian Meyer M.Sc.

Betriebssysteme
WS 2012/13

Übungsblatt 6
Praktische Übungen

In dieser Übung wird die UNIX-Prozesssynchronisation mittels Semaphoren angewendet (vgl. Kap. 5.2.3, Folien 5-37ff).

Semaphoren werden durch den UNIX-Kern als sogenannte "semaphor sets" bereitgestellt. Hierdurch ist es prinzipiell möglich, mehrere Semaphore der Menge in einer unteilbaren (atomaren) Operation zu manipulieren. Semaphore werden (wie auch die anderen System V IPC-Objekte Shared Memory Segmente und Message Queues) über Schlüssel (keys) aus einem allen Prozessen gemeinsamen Key-Namensraum referiert, vgl. semget(). Der Wert, den semget zurückgibt, ist der Bezeichner, semid genannt, den das Programm für den Aufruf der anderen Semaphor-Operationen verwenden muss. Semaphoren erlauben die Festlegung von Zugriffsberechtigungen wie bei Dateien. Wie in der Vorlesung besprochen (vgl. Folie 5-38), stehen für den Umgang mit Semaphoren die in der folgenden Tabelle zusammengefassten Systemdienste zur Verfügung. Konstanten, usw. sind in den include-Dateien `<sys/ipc.h>` und `<sys/sem.h>` enthalten (bzw. werden dort aus einer anderen include-Datei eingefügt, z.B. `<linux/ipc.h>`).

int semget(key_t key, int nsems, int semflag);	Erzeugen eines Sets von nsems Semaphoren
int semop(int semid, struct sembuf *sops, unsigned int nsops);	Durchführen von nsops Semaphor-Operationen auf Semaphoren des Sets semid
int semctl(int semid, int semnum, int cmd, union semun arg);	Steuerfunktionen: z.B. Anfangswert setzen, Zustand erfragen, löschen

Zur Beschreibung der Systemaufrufe sei auf die Manual Pages verwiesen.

Achtung: Die System V IPC-Objekte (Semaphore, Shared Memory Segmente, Message Queues) überleben die sie erzeugenden Prozesse. Löschen Sie daher entweder im Programm (`semctl()` mit dem Kommando `IPC_RMID`) bzw. nach Programmabsturz oder bei Beendigung der Sitzung mittels des Kommandos `ipcrm` alle von Ihnen angelegten derartigen Objekte. (Status)-Informationen über existierende Semaphoren usw. erhalten Sie mittels `ipcs`.

Aufgabe 6.1 (Beispiel für wechselseitigen Ausschluss):

In dieser Aufgabe wird ein Semaphor zur Sicherung des wechselseitigen Ausschlusses mehrerer Prozesse in der Benutzung eines gemeinsamen Betriebsmittels eingesetzt. Das Betriebsmittel ist das Bildschirmfenster, in dem die Prozesse gestartet werden und in das sie jeweils Ausgaben (über ihre Standardausgabe) durchführen. Zunächst erfolge die Ausgabe der Prozesse unsynchronisiert.

(a) Schreiben Sie ein Programm `gibaus.c`, dem beim Aufruf über `argv[]` ein Zeichen (z.B. 'A') und ein Wartezeit-Parameter sleep_time (in sec, z.B. 1) übergeben werden. Das

Programm gebe zyklisch ITERATIONS=10 mal eine Zeile mit jeweils ANZ_ZEICHEN=20 Zeichen aus. Nach der Ausgabe eines Zeichens lege sich der Prozess für sleep_time Sekunden schlafen, bevor er das nächste Zeichen (oder "\n") ausgibt. Das auszugebende Zeichen soll das bei Programmstart übergebene Zeichen sein. (Hierdurch können Sie die Ausgaben von verschiedenen Prozessen später unterscheiden).

(b) Erzeugen Sie mit dem Editor in einer Datei `run` ein kleines Shell-Skript, in dem Sie mehrere gleiche Prozesse über dem Programm `gibaus` im Hintergrund starten, z.B.

```
gibaus A 1 &
gibaus B 1 &
gibaus C 1 &
```

Was beobachten Sie bzgl. der Ausgabe der verschiedenen Prozesse?

(c) Erweitern Sie das Programm aus (a) zu einem Programm `gibaus_ex.c`, das sicherstellt, dass ein Prozess die Zeichen einer Zeile atomar ausgibt (und damit ausschließlich vollständige Zeilen mit gleichen Zeichen entstehen). Nutzen Sie dazu ein Semaphor.

Aufgabe 6.2 (Semaphor-Bibliothek):

In dieser Aufgabe soll, basierend auf Ihren in Aufgabe 6.1 gesammelten Erfahrungen, eine wiederbenutzbare Bibliothek für Einzelsemaphore (Semaphor-Menge mit einem Semaphor) auf der Basis der Semaphor-Systemaufrufe entwickelt werden. Diese Bibliothek wird in den weiteren Aufgaben zu verwenden sein.

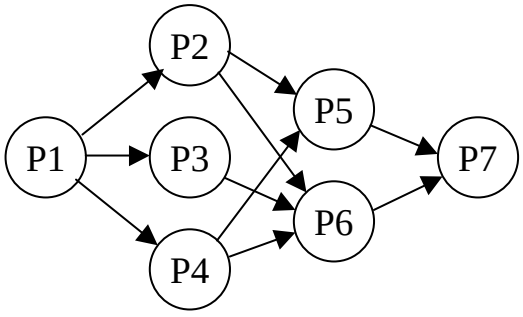
(a) Schreiben Sie eine Bibliothek für Einzelsemaphore mit den in der Vorlesung besprochenen Semaphor-Operationen `P()` und `V()` sowie den unterstützenden Operationen `sem_create()` zur Erzeugung und Initialisierung, `sem_delete()` zur Zerstörung, `sem_open()` zur Öffnung eines existierenden Semaphors sowie `sem_read()` zum Auslesen des aktuellen Werts eines Semaphors.

(b) Modifizieren Sie Ihre Lösung zu Aufgabe 6.1 (c), so dass Ihre Bibliothek eingesetzt wird.

Aufgabe 6.3 (Vorranggraph):

In dieser Aufgabe werden Semaphoren zur Durchsetzung einer Vorrangrelation entsprechend der Vorlesung (Folie 5-34/35) angewendet.

(a) Geben Sie für den folgenden Vorranggraphen zunächst auf Papier die Programm-Skelette der beteiligten Prozesse zur Durchsetzung der geforderten Synchronisation an. Verwenden Sie dazu Semaphore.



(b) Schreiben Sie ein Programm `semcreate.c`, das die Semaphoren entsprechend Ihrer für (a) vorgesehenen Lösung erzeugt.

(c) Schreiben Sie Programme `p1.c` ... `p7.c` für die Prozesse P1-P7 entspechend dem in (a) entwickelten Pseudocode. Benutzen Sie zum Umgang mit den Semaphoren die in Aufgabe 6.2 entstandenen Operationen. Jeder Prozess soll die Ausgabe einer Zeile vor jeder Aktivität (d.h. vor jeder Semaphor-Operation, vor der eigentlichen Arbeitsphase (work), am Ende) durchführen. Zur Modellierung der eigentlichen Arbeit des Prozesses und zwischen den Semaphor-Operationen können Sie den Prozess schlafen legen. Die Ausgaben jedes Prozesses sollen als Spalte erkennbar sein, so dass Sie den ineinander verzahnten Ablauf aller Prozesse gut beobachten können. Zum Beispiel könnte sich ergeben:

```
p1start
                  P3start
         p2start
work
                              ...
         P(sem_a)
                  P(sem_d)
V(sem_a)
...
ende!
         work
```

(Zufällige Zeitdauern können Sie durch einen Zufallszahlengenerator auf Basis der Library-Funktion `int rand(void)` mit der Initialisierung `void srand(unsigned seed)` für einen Anfangswert `seed` mit anschließender Normierung erreichen).

(d) Schreiben Sie in einer Datei `run` ein kleines Shell-Skript, das alle Prozesse P1-P7 im Hintergrund nahezu zeitgleich startet.

(e) Führen Sie mehrere Testläufe durch, und beobachten Sie das Geschehen zunächst am Bildschirm.

(f) Leiten Sie dann die Ausgabe von 3 Testläufen mit unterschiedlich gewählten Zeitspannen in Dateien um. Markieren Sie auf der Ausgabe die Vorrangbeziehungen des Graphen durch entsprechende Pfeile zwischen den Arbeitsphasen (work) der Prozesse. Hält Ihre Lösung alle geforderten Beziehungen ein?

Aufgabe 6.4 (optional, Prozesssynchronisation):

In dieser Aufgabe werden Semaphoren zur Synchronisation von Prozessen bei der Evaluierung eines arithmetischen Ausdrucks eingesetzt, wobei die Operanden in Dateien gespeichert werden.
Betrachten Sie den arithmetischen Ausdruck $y = a^2 + 2ab + b^2$. Vereinfachen Sie den Ausdruck für die Durchführung nicht in $(a+b)^2$ (dies können Sie zur Überprüfung Ihres Ergebnisses später tun). Als Wertemenge seien nur natürliche Zahlen für a und b betrachtet. Die konkret zu verwendeten Werte seien jeweils in einer Datei gleichen Namens als Zeichenketten vor Beginn der Berechnung gespeichert (z.B. mit einem Editor dort hineingeschrieben). Die Berechnung und Anzeige von y soll jetzt durch mehrere nahezu gleichzeitig gestartete Prozesse geschehen, die sich bzgl. der Verfügbarkeit der Zwischenergebnisse bzw. zur Anzeige des Endergebnisses synchronisieren müssen. Das Endergebnis werde in einer Datei y gespeichert und von einem speziellen Viewer-Prozess,

der ebenfalls gleichzeitig mit den Berechungsprozessen gestartet wird, auf Standardausgabe ausgegeben.

(a) Überlegen Sie, welche Synchronisationsbedingungen Sie einhalten müssen, wenn die drei Summanden nebenläufig bestimmt werden sollen und die Additionsfunktion mit genau zwei Operanden arbeiten kann. Wie können Sie diese Synchronisationsbedingungen insgesamt modellieren?

(b) Schreiben Sie Programme (1) für die Quadratbildung eines Operanden, (2) für die Bildung von 2ab, (3) für die Summe zweier Operanden, (4) für die Ausgabe des Ergebnisses. Namen für Parameterdateien und evtl. notwendige Keys für Semaphoren sollen per Kommmandozeile übergeben werden. Nutzen Sie Ihre Semaphoren-Bibliothek aus Aufgabe 6.2.

(c) Schreiben Sie ein Initialisierungsprogramm `expinit.c`, das alle notwendigen Semaphoren erzeugt und korrekt initialisiert.

(d) Schreiben Sie ein Skript, mit dem Sie alle für die Berechnung und Ausgabe des Ergebnisses notwendigen Programme starten. Geben Sie dabei die für die Bestimmung notwendigen Dateinamen für Parameter, Zwischenergebnisse und das Endergebnis sowie die Keys der zu benutzenden Semaphoren als Kommandozeilenparameter an.

(e) Schreiben Sie ein Programm `expclean.c`, mit dem Sie am Ende alle Semaphoren zerstören können.

(f) Führen Sie mehrere Berechnungen für konkrete Werte von a und b durch. Stimmen Ihre berechneten Ergebnisse für y? Wird korrekt synchronisiert?