Vorbereitung

- Booten Sie den Rechner unter Linux und melden Sie sich mit Ihrem „`ifw`...“- oder „`ibw`...“-Account an (Passwort ist evtl. die Matrikelnummer).
- Legen Sie (falls nicht bereits geschehen) in Ihrem Home-Verzeichnis ein Unterverzeichnis `bspraktikum` an (**`mkdir bspraktikum`**) und wechseln Sie in das neue Verzeichnis (**`cd bspraktikum`**).
- Laden Sie das Archiv mit den Aufgaben-Dateien herunter:
  `$ wget http://fhm.hgesser.de/bs-ws2006/prakt2.tgz`
- Entpacken Sie das Archiv mit `tar`:
  `$ tar xzf prakt2.tgz`
- Dadurch entsteht ein neues Unterverzeichnis `prakt2`, in das Sie mit **`cd prakt2`** hinein wechseln. Jetzt kann es los gehen...

5. Inter Process Communication (IPC) mit Sockets

a) Programmieren Sie (in Python oder einer Sprache Ihrer Wahl) zwei Programme, die die folgenden Aufgaben übernehmen:

Ein Server (`server.py`) öffnet einen TCP-Port (z. B. 5555; SOCK_STREAM) und wartet auf Verbindungen.

Ein Client (`client.py`) baut eine Verbindung zum Server auf und schickt ihm einen Dateinamen.

Der Server prüft, ob die Datei existiert – wenn ja, schickt er als Antwort mehrere Zeilen Text mit folgenden Informationen zurück:

- Größe der Datei (wenn es eine reguläre Datei ist)
- erste Zeile der Datei (wenn es eine reguläre Datei ist)
- Anzahl der Einträge im Verzeichnis (wenn es ein Verzeichnis ist)
- wenn es sich weder um eine Datei noch um ein Verzeichnis handelt (aber ein Eintrag unter dem angegebenen Namen existiert), schickt der Server die Meldung „E_NEITHERFILENORDIR“ zurück
- existiert gar kein Eintrag, schickt der Server die Meldung „E_NOSUCHFILE“ zurück.

Der Client wertet die Rückgabe aus.

- Falls eine Datei erkannt wurde, schickt er an den Server die Aufforderung „linetwo“; der Server reagiert darauf, indem er als Antwort die zweite Zeile der Datei schickt.
- Für den Fall, dass ein Verzeichnis erkannt wurde, schickt er an den Server die Aufforderung „listdir“; der Server reagiert darauf, indem er als Antwort eine Liste aller Dateien schickt.
- In den Fällen E_NEITHERFILENORDIR und E_NOSUCHFILE erfolgt keine weitere Kommunikation.

Der Client gibt schließlich die erhaltenen Daten aus. Beide Programme bauen die Verbindung ab.

b) Passen Sie Ihr Programm aus a) so an, dass es mit UDP (statt TCP) arbeitet. Vorsicht: In dem Fall gibt es keine Verbindung, und Sie müssen mit SOCK_DGRAM arbeiten.

6. IPC mit Pipes

Ändern Sie die Programme aus Aufgabe 5 so ab, dass sie zwei benannte Pipes zur Kommunikation verwenden, die erste Pipe (`/tmp/pipe-request`) ist für die Übertragung der Anfrage vom Client zum Server gedacht, die zweite Pipe (`/tmp/pipe-reply`) für die Übertragung der Antwort in umgekehrter Richtung.

Dabei soll der Server die beiden Pipes erzeugen. (Er startet auch als erstes der beiden Programme.)

7. Synchronisation

In der Vorlesung haben Sie das folgende (fehlerhafte) Programm `test.py` gesehen:

```
#!/usr/bin/python

# Betriebssysteme, FH Muenchen, H.-G. Esser, WS 2006/07
# lock.py
# Version 1.0 (2006-12-12)

from threading import Thread

class testthread(Thread):
   def __init__ (self):
      Thread.__init__(self)
   def run(self):
      global globalcount
      for j in range(0,99999):
         globalcount += 100
         globalcount -= 100

globalcount=0
threads = []
for i in range(0,100):
  t = testthread()
  threads.append(t)
  t.start()

for t in threads: t.join()

print "Ergebnis:",globalcount
```

Ebenfalls aus der Vorlesung kennen Sie das Konzept der `test_and_set`-Funktionen, welche atomar eine Variable ändern und ihren vorherigen Wert zurückgeben. In der Datei *test_and_set.py* finden Sie obiges Programm um eine `test_and_set`-Funktion ergänzt, die innerhalb der *testthread*-Klasse definiert ist:

```
def test_and_set(self,x):
  # x muss 0 oder 1 sein
  global tslvar
  tsl_lock.acquire()
  oldvalue = tslvar
  tslvar = x
  tsl_lock.release()
  return oldvalue
```

a) Beheben Sie das Problem im Programm mit einem Aufruf von `test_and_set()`. Hinweis: Da die Funktion in der Klasse definiert ist, müssen Sie sie mit `self.test_and_set(self,x)` aufrufen.

b) Vergleichen Sie die Laufzeit mit den Lösungen aus der Vorlesung (die mit Locks oder Semaphoren arbeiten; Sie finden diese Dateien im Aufgabenpaket als `lock.py` und `sem.py`. Warum ist die Variante mit *test_and_set* so langsam?

8. Synchronisation (Weihnachtsaufgabe)

Betrachten Sie das Programm im Anhang (`ipc.py` im Aufgabenpaket), das aus mehreren Threads besteht. Diese sollen folgende Aufgaben bearbeiten:

Die Threads t1 und t2 erzeugen

- 250 aufeinander folgende ungerade Zahlen (1,3,5,7,...,499) (t1)
- 250 aufeinander folgende gerade Zahlen (0,2,4,6,8,...,498) (t2)

und schreiben sie in eine gemeinsame Liste, die maximal 50 Elemente aufnehmen kann. (Diese Liste wird vom Hauptprogramm definiert, bevor die Threads starten. Jede Thread-Funktion muss die Variable im ersten Befehl mit **`global liste`** als globale Variable definieren!) Jeder der Threads erzeugt genau 250 Zahlen und beendet sich dann.

Threads t3 und t4 sind analog für das Auslesen ungerader und gerader Variablen aus der Liste zuständig; sie berechnen laufend die bisherige Summe und die bisherige Anzahl der gelesenen Werte. Beide Threads dürfen nur auf den jeweils ersten Eintrag in der Liste zugreifen – ist dieser vom richtigen Typ (für t3 also ungerade), dürfen sie den Eintrag verwerten und aus der Liste löschen. Ist der vorderste Eintrag vom falschen Typ (für t3 etwa gerade), wird der Eintrag ignoriert.

Wenn das Programm richtig arbeitet (was es gelegentlich tut), gibt es aus:

```
Anzahl gerader Zahlen: 250
Anzahl ungerader Zahlen: 250
Summe gerader Zahlen: 62250
Summe ungerader Zahlen: 62500
Durchschnittliche gerade Zahl: 249.0
Durchschnittliche ungerade Zahl: 250.0
```

Das Programm funktioniert aber nicht richtig – unter anderem, weil es keinerlei Synchronisation durchführt. Reparieren Sie das Programm. Dazu benötigen Sie:

- ein globales Lock für den Zugriff der vier Threads auf die Liste
- einen oder mehrere Semaphor für die Zustände der Liste (ganz leer, ganz voll, ...)
- eventuell Signale, mit denen sich Threads gegenseitig benachrichtigen.

In der heutigen Übung können Sie ein wenig mit dem Programm experimentieren und es beispielsweise an geeigneten Stellen die aktuellen Listeninhalte ausgeben lassen. Für den Einbau der Synchronisationsfunktionen benötigen Sie voraussichtlich mehr Zeit.

```
#!/usr/bin/python
# Betriebssysteme, FH Muenchen, H.-G. Esser, WS 2006/07
# synchr.py
# Version 1.0 (2006-12-19)
import threading; from time import sleep
MAXLIST=50; liste=[]; no_odds=0; no_evens=0; sum_odds=0; sum_evens=0

def is_odd(n):
  if (n/2)*2 == n: return False
  else: return True

class writer(threading.Thread):
  def __init__(self,odd_even):
    threading.Thread.__init__(self)
    if odd_even == "odd": self.odd = True
    else: self.odd = False
  def run(self):
    global liste, MAXLIST
    for i in range(250):   # Schleife 250x durchlaufen
      v=2*i
      if self.odd: v=v+1   # odd: 1,3,5,7,...;  not odd: 2,4,6,8,...
      if len(liste) >= MAXLIST: continue
      liste.append(v)
      sleep(0.001)

class reader(threading.Thread):
  def __init__(self,odd_even):
    threading.Thread.__init__(self)
    if odd_even == "odd": self.odd = True
    else: self.odd = False

  def run(self):
    global liste
    global no_odds   # Anzahl der gelesenen ungeraden Werte
    global no_evens  # Anzahl der gelesenen geraden Werte
    global sum_odds  # Summe der gelesenen ungeraden Werte
    global sum_evens # Summe der gelesenen geraden Werte
    for i in range(1500):  # Schleife
      if len(liste) == 0: continue
      v = liste[0]  # erstes Element der Liste
      if ( is_odd(v) and self.odd ) or ( not(is_odd(v)) and not(self.odd) ):
        neueliste = liste[1:]  # erstes Element der Liste entfernen
        if self.odd:
          no_odds += 1
          sum_odds += v
        else:
          no_evens += 1
          sum_evens += v
      liste = neueliste
      sleep(0.001)

# Hauptprogramm
t1 = writer ("odd"); t2 = writer ("even")   # vier Threads erzeugen, 2 Writer, 2 Reader
t3 = reader ("odd"); t4 = reader ("even")
for t in [t1,t2,t3,t4]: t.start()   # alle Threads starten
for t in [t1,t2,t3,t4]: t.join()    # alle Threads wieder einsammeln
print "Anzahl gerader Zahlen:", no_evens    # Ergebnis ausgeben
print "Anzahl ungerader Zahlen:", no_odds
print "Summe gerader Zahlen:",sum_evens
print "Summe ungerader Zahlen:",sum_odds
try: print "Durchschnittliche gerade Zahl:",float(sum_evens)/no_evens
except: ()
try: print "Durchschnittliche ungerade Zahl:",float(sum_odds)/no_odds
except: ()
print "Rest-Liste:",liste
```