Christian-Albrechts-Universität zu Kiel
Wintersemester 1998/99
Informatik III

Übung 7:

Ausgabetermin: 14. Dezember 1998

Abgabe: 22. Dezember 1998, bis 11:00 Uhr (Einzelabgabe diesmal)

Aufgabe 1: [Rendez-Vous]

Wir haben in der Vorlesung *message passing* als eine Möglichkeit behandelt (Kapitel 7 aus [Com83]). Die Implementierung in Xinu ist nur eine (recht einfache) Variante wie sich Kommunikation durch Nachrichtenaustausch in Betriebssystemen realisieren läßt. Eine Variante des *message passing* ist Kommunikation durch *Rendez-Vous*-Kommunikation. Ihr Kennzeichen ist, daß sowohl das Senden wie das Empfangen *blockierend* sind.

Nehmen wir an, wir hätten —wie beim *message-passing* Konzept von Xinu— einen Sendeprozeß und einen Empfangsprozeß, dann "blockiert" der Sender einer Nachricht solange, bis der Empfänger sie liest und umgekehrt.

Implementieren sie zwei Systemaufrufe

- `SYSCALL sendrv (pid, msg)` und
- `SYSCALL receiverv(pid)`

mit dem beschriebenden Verhalten. Beachten Sie, daß der Systemaufruf für das Empfangen nun den Sender als Parameter bekommt. Überlegen sie, warum das im Zusammenhang mit Rendez-Vous-Kommunikation sinnvoll ist. Überlegen Sie auch, was passierten soll, wenn ein Prozeß sich selbst eine Rendez-Vous-Nachricht zukommen lassen will.

Aufgabe 2: [Produzent/Konsument]

Ein klassisches Prozeßsynchronisationsproblem ist das *Produzenten/Konsumenten*-Problem[2] Gegeben sei folgendendes Programmfragment.

```
/* producer/consumer Pseudocode                          */

#define N   100                  /* Plaetze im Puffer  */
```

[2] Auch als das Problem des beschränkten Puffers bekannt (auf Englisch *producer/consumer, bounded-buffer*).

```
int count   0                     /* belegte Plaetze    */

producer ()
{
  while (TRUE) {
    produce_item();                          /* Produzent tut seine Aufgabe  */
      if (count == N) suspend(currentpid);   /* Falls Puffer voll:           */
                                             /* Produzent suspendiert sich   */
        enter_item();
        count = count+1;
        if (count == 1) resume(consumer_pid); /* Konsument soll weitermachen */
                                              /* da Puffer nicht mehr leer   */
       }
}

consumer ()

{
        while (TRUE) {
          if (count == 0) suspend(currentpid);   /* Konsument suspendiert sich selbst */
          remove_item();
          count=count-1;
          if (count== N-1) resume(producer_pid); /* Produzent soll weitermachen    */
          consume_item();
        }

}
```

Die Absicht ist, die notwendige *Synchronisation* zwischen Konsument und Produzent durch wechselseitiges Suspendieren und Wiederaufwecken zu bewerkstelligen. So wie angegeben, funktioniert die Lösung nicht. Begründen sie warum und beheben Sie den Fehler. Geben Sie eine Lösung mit Semaphoren und eine ohne.

Literatur

[Com83] Douglas Comer. *Operating System Design, The Xinu Approach.* Prentice Hall, 1983.