Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik
Prof. Dr. W.-P. de Roever
Martin Steffen


# Informatik III (OS)

Wintersemester 2003/04 — **Serie 3** — 11. November 2003

**Thema : Laufzeitsystem, Produzent-Konsument**

**Ausgabetermin: 11. November 2003**

**Abgabe: 19. November 2003**

**Aufgabe 1 (Stack (3 Punkte))** Zeichnen sie den Verlauf des Laufzeitstacks für folgende kleine Prozedur. Beim Aufruf mit dem Wert 3 auf.[1]

```
int fib(int i) {
  if (i==0) return 1;
  if (i==1) return 1;
  return (fib(i-1) + fib(i-2));
};
```

**Aufgabe 2 (Interrupts (2 Punkte))** Interruptverktoraddressesn heißen *programmierbar,* wenn sie in Software änderbar sind. Warum sind programmierbare Adressen sinnvoll?

**Aufgabe 3 (Produzenten/Konsumenten-Problem mit binären Semaphoren (5 Punkte))** Das Problem der Produzenten-Konsumenten-Koordination über einen beschränkten Puffer soll ein weiteres Mal angegangen werden. Diesmal soll nur eine eingeschränkte Form von Semaphoren zur Verfügung stehen, nämlich sogenannte *binäre* Semaphore. Diese sind gegenüber der allgemeinen Form eingeschränkt, indem sie nur *zwei Werte* annehmen können: 0 und 1.[2] Geben Sie eine Lösung für das Problem mit beschränktem Puffer an, indem Sie folgendes Codefragment vervollständigen! Erläutern Sie die Korrekheit Ihrer Lösung!

```
/* producer/consumer Pseudocode-Fragement                    */

int Buf[N];
```

[1] Richtig. Fibonnacci soll man nicht auf diese Weise programmieren.

[2] Läßt man negative Werte zu, z.B., wenn man, der Konvention in Xinu folgend, (mehrere) an der Semaphore wartende Prozesse mit einem negativen Zählerstand vermerkt, müßte man präziser sagen, binäre Semaphore lassen zwei *nicht-negative* Werte zu. Binäre Semaphore können als Spezialfall der "zählenden" Semaphore, wie sie unter anderem in Xinu realisiert sind, angesehen werden. Der Spezialfall besteht darin, daß die Semaphore nur angeben kann, ob eine Resource vorhanden ist (Wert 1) oder nicht (Wert 0), und nicht die Anzahl der freien Resourcen zählen kann.

```
producer ()
{
  int datum, in;

  while (TRUE) {

    datum    = produce();          /* Produzent tut seine Aufgabe      */
    Buf[in] = datum;                    /* Einfuegen                        */
    in       = (in + 1) mod N;          /* und modulo weiterz"ahlen         */

  }
}

consumer ()
{
  int datum, out;

  while (TRUE) {

    datum =  Buf[out];                  /* Datum auslesen                   */
    out    =  (out + 1) mod N;          /* weiterz"ahlen                    */
    consume(datum);

  }
}
```

---

## Literatur

[BA90] M. Ben-Ari. *Principles of Concurrent and Distributed Programming.* International Series in Computer Science. Prentice Hall, 1990.