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

Sys. Inf. III (Betriebssysteme)

Wintersemester 2004/05 — **Serie 12** — 17. Januar 2005

Thema: E/A in Xinu

Ausgabetermin: 17. Januar 2005

Abgabe: 24. Januar 2005

Zur Erinnerung: der Endsemestertest findet am Samstag, den **29.1.2005** um **10:00** in CAP3-R2 statt. Unterlagen sind gestattet.

Wir haben in der Vorlesung anhand eines einfachen „Teletype"-Terminaltreibers gesehen, welch wichtige Rolle das Produzenten-Konsumenten-Problem in Betriebssystemen spielt. Man kann es abstrakt als einen grundlegenden Kommunikationsmechanismus betrachten, und zwar als asynchrone, also gepufferte, Kommunikation zur Datenübertragung vom Produzent zum Konsument.

Aufgabe 1 (Produzent/Konsument (3 Punkte)) Die erste Aufgabe gehe, in einigermaßen unrealistischer Weise, von einem *unbeschränkten* Puffer aus.[1] Der Puffer zur Entkoppelung von Produzenten und Konsumenten sei als array `Buff` vereinbart. Vervollständigen sie folgendes Codefragment.

```
/* producer/consumer pseudocode                              */

producer ()
{
  while (TRUE) {                    /* Produzent                    */

    datum    = produce_item();      /* Produzent tut seine Aufgabe  */
    Buf[in] = datum;                /* Einfuegen                    */
    in       = in + 1;              /* und weiterz"ahlen            */
```

[1] Unbeschränkter Puffer ist nicht ganz so unrealistisch wie es zunächst scheint. In Kommunikationsprotokollen hat man es manchmal mit sehr stark variierenden Produzentenverhalten zu tun. In diese Fällen benötigt man *dynamischen* Pufferplatz, der sich in erster Näherung als unendlich betrachten läßt. Natürlich kann der Speicher immer noch "volllaufen". Falls man das Problem dort nicht einfach vollständig vernachlässigt, indem man beispielsweise neue Nachrichten wegwirft, verwendet man in der Regel dort andere Mechanismen als die hier besprochenen. Soviel zur Rechtfertigung.

```
  }
}

consumer ()                                  /* Konsument                    */
{
  while (TRUE) {

    datum = Buf[out];
    out   = out + 1;
    consume(datum);
  }
}
```

Aufgabe 2 (Endlicher Puffer (4 Punkte)) Spezialisieren Sie die Lösung aus Aufgabe 1 zu einem *endlichen* Puffer der festen Größe n. Wie verhält sich Ihre Lösung, wenn man nicht nur einen Produzenten und einen Konsumenten, sondern mehrere erlauben möchte?

Unter *Verklemmung* versteht man intuitiv das unerwünschte Phänomen, wenn ein Prozeß nicht fortschreiten kann. Im Zusammenhang mit der Modellierung des Produzenten-Konsumenten-Szenarios bedeutet dies, daß der Produzent kein neues Datum abliefern kann, obwohl er er eines vorrätig hat, oder der Konsument keines lesen kann, obwohl ungelesene Daten im Puffer vorhanden sind. Anders ausgedrückt: daß die beiden Prozesse zu arbeiten aufhören, weil der Produzent schlicht nichts mehr produziert, ist *keine* Verklemmung. Frage: kann sich Ihre Lösung verklemmen?

Aushungern (*starvation*) ist ein weiteres unerwünschtes Prozeßverhalten. Intuitiv läßt es sich als der Umstand beschreiben, daß ein Prozeß zwar im Prinzip eine Aufgabe ausführen kann, aber ständig *übergangen* wird, beispielsweise, ein Prozeß ist zwar ständig im Zustand *ready*, es werden beim Kontextwechsel aber immer andere Prozesse ausgewählt. In Zusammenhang mit der Aufgabe interessiert uns: können sich Produzent und Konsument gegenseitig aushungern? Wie sieht es aus, falls man mehrere Produzenten (oder Konsumenten) hat?

Aufgabe 3 (Produzent/Konsument (3 Punkte)) Schaut man sich in Xinu den Kontrollblock für Tty's an ([**?**], S. 163), stellt man erstens fest, daß die Kommunikation mittels endlicher Puffer geschieht und zweitens, daß pro Produzenten/Konsumenten-Paar nur je *eine* Semaphore implementiert ist. Erklären Sie, warum zwei Semaphoren dort nicht verwendet werden können und auf welche Weise dennoch eine korrekte Funktionsweise sichergestellt ist. Warum verwendet man keine unbeschränkten Puffer?

Aufgabe 4 (Echo (2 Punkte)) Neben dem Eingabe– und dem Ausgabepuffer gibt es noch den "Echo"-Puffer. Dieser ist ganz ohne eigene Semaphore implementiert. Warum?