Mumm, Harald

Working Paper

Die Wirkungsweise von Betriebssystemen am Beispiel der Tastatur-Eingabe

Wismarer Diskussionspapiere, No. 02/2004

Provided in Cooperation with:
Hochschule Wismar, Wismar Business School

Suggested Citation: Mumm, Harald (2004) : Die Wirkungsweise von Betriebssystemen am Beispiel der Tastatur-Eingabe, Wismarer Diskussionspapiere, No. 02/2004, ISBN 3910102433, Hochschule Wismar, Fachbereich Wirtschaft, Wismar

This Version is available at:
https://hdl.handle.net/10419/39819

Hochschule Wismar
University of Technology, Business and Design
Fachbereich Wirtschaft

Hochschule Wismar
University of Technology, Business and Design
Faculty of Business

Harald Mumm

Die Wirkungsweise von Betriebssystemen am Beispiel der Tastatur-Eingabe

Heft 02 / 2004

W D P

Wismarer Diskussionspapiere / Wismar Discussion Papers

Der Fachbereich Wirtschaft der Hochschule Wismar, University of Technology, Business and Design bietet die Präsenzstudiengänge Betriebswirtschaft, Management sozialer Dienstleistungen, Wirtschaftsinformatik und Wirtschaftsrecht sowie die Fernstudiengänge Betriebswirtschaft, International Management, Krankenhaus Management und Wirtschaftsinformatik an. Gegenstand der Ausbildung sind die verschiedenen Aspekte des Wirtschaftens in der Unternehmung, der modernen Verwaltungstätigkeit im sozialen Bereich, der Verbindung von angewandter Informatik und Wirtschaftswissenschaften sowie des Rechts im Bereich der Wirtschaft.

Nähere Informationen zu Studienangebot, Forschung und Ansprechpartnern finden Sie auf unserer Homepage im World Wide Web (WWW): http://www.wi.hs-wismar.de/.

Herausgeber: Prof. Dr. Jost W. Kramer
Fachbereich Wirtschaft
Hochschule Wismar
University of Technology, Business and Design
Philipp-Müller-Straße
Postfach 12 10
D – 23966 Wismar
Telefon: ++49 / (0)3841 / 753 441
Fax: ++49 / (0)3841 / 753 131
e-mail: j.kramer@wi.hs-wismar.de

ISSN 1612-0884
ISBN 3-910102-43-3

JEL-Klassifikation Z00

Inhaltsverzeichnis

1. Einleitung 4

2. Der Aufbau der Software 5

3. Beschreibung und Funktionalität der Systemaufrufe 6

4. Die beteiligten Prozesse 7

4.1. Der Verwalter 7
4.2. Die Schale 7
4.3. Der Kunde 8
4.4. Der Leerlaufprozeß 8

5. Der Faden bei wichtigen Abläufen 8

5.1. Die Eingabe einer Zahl über eine simulierte Tastatur 8
5.2. Lesen aus dem Eingabestrom 8

6. Zusammenfassung 9

7. Quelltexte 10

8. Demonstrationsbeispiel und Literatur 15

Autorenangaben 17

1 Einleitung

Das Ziel dieser Arbeit ist die Offenlegung der prinzipiellen softwareseitigen Abläufe in einem Computer bei Eingabe von Zeichen über eine Computer–Tastatur für ein Anwendungsprogramm, in dem das Zusammenspiel von Betriebssystem und Anwendungsprogramm offengelegt wird. Neben dieser konkreten Aufgabe sollen auch weitere Aufgaben und Lösungen von Betriebssystemen vorgestellt werden. Warum wurde nun gerade diese Aufgabe zur Demonstration ausgewählt ? Andrew S. Tanenbaum schrieb in seinem Buch „Betriebssysteme“([3]):

„Die meisten Bücher und Vorlesungen behandeln ausführlich die Scheduling–Algorithmen, die in der Praxis normalerweise weniger als eine Seite des Codes beanspruchen, während die gesamte Ein- und Ausgabe ignoriert wird, die häufig dreißig Prozent oder mehr des gesamten Systems ausmachen.“

Neben der konkreten Zielstellung geht es allgemein um die Offenlegung von Betriebssystem–Software für die Lehre. Insofern ist die Tastatureingabe nur ein Mittel zum Zweck.

Ähnlich motiviert schienen Niklaus Wirth und Jürg Gutknecht gewesen zu sein. In [2] findet man:

„We wished not only to give advice on how a system might be built, but to demonstrate how one was built.“

Die erkannten Defizite waren der Ausgangspunkt für zwei neue Betriebssysteme „Minix“und „Oberon“. Beide Systeme haben den Vorteil, daß sie auf realen Computern funktionieren, aber den Nachteil, daß sie sehr groß sind. Einen anderen Weg geht deshalb Hans Röck in [1] mit seinem Mehrfach–Aufgaben–Betriebssystem Klax für seine didaktische Neumann–Maschine Paula. Hierbei handelt es sich um eine didaktisch orientierte konzeptionelle Lösung in der selbst entwickelten problemorientierten Programmiersprache (besser Entwicklungssprache) Anton. Das Hauptziel war dabei die Vorzeigbarkeit und Lehrbarkeit von Betriebssystemen in der Ausbildungsrichtung Wirtschaftsinformatik mit sehr geringem Stundenvolumen und geringen Vorkenntnissen seitens der Studierenden.

Klax ist im Gegensatz zu Minix, das eine Mikrokernarchitektur besitzt, ein monolithisches System. Der Wechsel von einem Anwendungsprogramm zum Betriebssystem erfolgt über sogenannte Systemaufrufe. Durch die Systemaufrufe kann ein Anwendungsprogramm gewisse Unterprogramme des Betriebssystems aufrufen, als wenn es seine eigenen wären. Wenn ein Anwendungsprogramm ein Unterprogramm des Betriebssystems Klax abarbeitet, spricht er vom Dienst- oder Kernmodus, in dem es sich dann befindet. Beim Kernmodus arbeitet das Anwendungsprogramm im Code–Adressraum des Betriebssystems.

Anwendungsprogramme müssen i.allg. auch Zahlen von der Tastatur eines Personal–Computers übernehmen können. Dieser Vorgang wird auch Einlesen genannt. Damit ist sowohl die Eingabe von Zahlen an einer Tastatur als auch die Übernahme der eingegebenen Zahl durch das Anwendungsprogramm gemeint. Mindestens zwei Szenarien sind dabei denkbar. Erstens, die Zahl wird zu früh an der Tastatur eingegeben (vorzeitige Eingabe), d.h. im Anwendungsprogramm wurde die Stelle des Einlesens noch gar nicht erreicht und zweitens, die Zahl wird erst nach dem Erreichen dieser Stelle eingegeben (nachzeitige Eingabe). Im ersten Fall därf die Zahl nicht verloren gehen und im zweiten Fall sollte der Computer

nicht durch aktives Warten blockiert werden.

Die konkrete Zielstellung dieser Arbeit liegt nun darin, eine Beispiellösung der Tastatureingabe aus [1] abzukoppeln und die Tragfähigkeit der verwendeten Konzepte an einer Experimentallösung nachzuweisen. Auf Grund der Teilfunktionalität wird im Folgenden anstelle vom Betriebssystem nur vom Verwalter–Programm gesprochen.

Als Implementierungssprache dient die in [4] vorgestellte objektorientierte Erweiterung von Anton. Variablen für Objekte sind hier Zeiger, die mit dem Adreßoperator & erzeugt werden. Objekte gibt es nur im Keller–Bereich. Dummy–Variable sorgen für den erforderlichen Speicherplatz (siehe im Quelltext im siebenten Abschnitt bei *).

Der dafür selbst entwickelte Compiler dient der Übersetzung der Programme in die Maschinensprache der Neumann–Maschine Paula und ein in Java implementierter Simulator, der unter `http://www.wi.hs-wismar.de/~mumm` auch ausprobiert werden kann, erlaubt die Abarbeitung des Beispielprogramms.

2 Der Aufbau der Software

Die Demonstrationssoftware soll objektorientiert entwickelt werden. Dazu werden die Klassen „Prozess“, Strom und Betrieb für die Verwalter–Software und S_Neutral für die Anwendungssoftware definiert. Die Schnittstellen der vom Programm MiniVerwalter bereitgestellten Unterprogramme werden einem Anwendungsprogramm in der Klasse S_Neutral bekanntgegeben. Ihr Aufruf aus einem Anwendungsprogramm heraus wird, wie bereits oben erwähnt, Systemaufruf genannt.

Die Klasse S_Neutral ist insofern eine besondere Klasse, als daß sie nur aus Schnittstellendefinitionen der Methoden der Klasse Betrieb besteht. Sie ist notwendig, weil die Systemaufrufe hier nicht in Dienstaktionen versteckt werden sollen und der Compiler ihre Schnittstelle für die Syntaxprüfung ihres Aufrufes benötigt.

Wenn der Compiler in einem Anwendungsprogramm den Aufruf einer Methode aus dieser Klasse entdeckt, generiert er neben Anweisungen zur Kopie der Werte der aktuellen Parameter eine prozeßwechselnde Paula–Maschinen–Anweisung mit einem entsprechenden Parameter. Dieser Parameter heißt im Verwalter–Programm Aufruf und wird in der Methode setzeInDienstModus ausgewertet. Bevor die Unterprogramme des Programmes MiniVerwalter im Auftrag eines Kunden abgearbeitet werden können, wird ihnen in der Methode setzeInDienstModus noch ein zusätzlicher Parameter übergeben, der die sogenannten Betriebsdaten bereitstellt.

In dieser Arbeit dient die Klasse Betrieb der Beschreibung der verwendeten Betriebsdaten, die hier nur aus den folgenden vier Variablen bestehen: Der Adresse des geraden laufenden Prozesses (bereit), zwei Zeigern auf je eine Stromdatei (st1 und st2) und der Nummer der gerade fokussierten Stromdatei (focus). Ihre wichtigsten Nachrichten heißen F1 bis F5. Schickt ein Anwendungsprogramm eine dieser Nachrichten an ein in jedem Anwendungsprogramm vorhandenes Objekt vom Typ S_Neutral, erfolgt ein Systemaufruf. Seine genaue Beschreibung erfolgt

im nächsten Abschnitt.

Die Objekte der Klasse „Prozess“verwalten die Registerinhalte eines Prozesses nachdem er unterbrochen wurde. Vor einem Wiederanlauf müssen diese Werte wieder in die Register der Neumann–Maschine geladen werden. Diese Aufgabe übernimmt das Mikroprogramm der Neumann–Maschine Paula bei Interpretation der dafür vorgesehenen Maschinenanweisung. Sie wird vom Compiler bei der Übersetzung der Methode werdeAktiv beim Erreichen der Codegenerierung für den Befehl aktiviere generiert.

Die Klasse Strom dient der Hantierung von Eingabepuffern, auf die Eingabewerte solange abgelegt werden, bis sie von einem Anwendungsprogramm angefordert werden. Die Objekte von diesem Typ werden auch Stromdateien genannt, obwohl sie im Hauptspeicher liegen. (Eine Stromdatei wurde hier analog zu [2] als Ringpuffer implementiert.)

Ihre wichtigsten Methoden sind bedieneTastatur und ZahlVP. Die erste Methode wird aufgerufen, wenn ein Tastatur–Interrupt aufgetreten ist und die zweite, wenn ein Anwendungsprogramm Daten aus der Stromdatei übernehmen will.

3 Beschreibung der Funktionalität der Systemaufrufe

Das Programm MiniVerwalter stellt die Methoden F1 bis F5 den Anwendungsprogrammen als Dienstleistung zur Verfügung, die über Systemaufrufe von den Anwendungsprogrammen ausgeführt werden können.

Die erste Methode F1 dient der Übernahme einer Zahl von einem Datenbereich des Programmes MiniVerwalter in einen Datenbereich des Anwendungsprogramms.

Dabei wird von der Fähigkeit des Simulators Gebrauch gemacht, eine Zeichenkette von Ziffern–Zeichen in eine Binärzahl umzuwandeln, so daß der Code dazu in dieser Methode entfallen kann.

Wie bereits oben erwähnt, besteht das Einlesen einer Zahl von der Tastatur aus einem produzierenden Teil (die Zahl wird eingetippt) und einem konsumierenden Teil, diese Zahl wird in ein Anwendungsprogramm kopiert.

Sofern die Zahl schon vorliegt (vorzeitige Eingabe), braucht sie nur ausgeliefert zu werden. Ansonsten (nachzeitige Eingabe) muß der Kunde solange warten, bis die Zahl eingegeben wurde.

Zur Programmierung dieses Prozederes dienen Objekte der Klasse Strom. Sie enthalten einerseits den Datenpuffer buf für maximal drei Zahlen und einen Zeiger Kunde auf einen eventuell schon wartenden Kunden. Wenn also noch keine Zahl vorliegt, wird der gerade laufende Kunde (seine Adresse steht in der Variablen bereit) als Wartender hier eingetragen. Im Verwalterprogramm MiniVerwalter kommen zwei Stromdateien zum Einsatz, weil nur die Schale und das Anwendungsprogramm die Zahlen von der Tastatur einlesen wollen.

Die zweite Methode F2 dient der Ausgabe einer Binärzahl. Dazu muß sie zuerst in eine Zeichenkette umgewandelt werden. Zeichen für Zeichen wird dann anschließend (durch Angabe des ASCII–Codes) an den Simulator übergeben, der es in ein Ausgabefenster schreibt.

Die dritte Methode F3 dient der Fokussierung einer Stromdatei bei Vorgabe ihrer Nummer (eins oder zwei).

Die vierte Methode F4 wird verwendet, um den Kunden zum Laufen zu bringen. Da seine Startadresse in diesem Szenario von vornherein feststeht, kann bei diesem Systemaufruf auf den Parameter verzichtet werden. In realen Systemen liefert die Methode, die für das Laden des Codes zuständig ist, natürlich diese Adresse.

Die fünfte Methode F5 ähnelt der Zweiten. Sie soll lediglich die Ausgabe eines Zeichens (bei F2 ging es um eine Binärzahl) in einen Ausgabestrom simulieren, der dann von einem Gerät (z.B. Bildschirm) angezeigt wird. Da der Fokus dieser Arbeit nur auf der Tastatureingabe liegt, dient F5 lediglich der Übergabe eines ASCII–Codes an den Simualtor, damit er das dazugehörige Zeichen in einem Ausgabefenster darstellt. Die Übergabe des ASCII–Codes an den Simulator erfolgt mit der Aktion gibAus.

4 Die beteiligten Prozesse

Die Abarbeitung eines Programmes auf geeigneter Hardware heißt Prozeß.

4.1 Der Verwalter

Beim Booten (Starten) werden die Maschinencode–Dateien der Programme MiniVerwalter, Schale und Leerlauf in den Hauptspeicher der Neumann–Maschine Paula geladen. Das Programm MiniVerwalter sorgt dafür, daß die Schale anlaufen kann und daß auf Unterbrechungen, entweder von der Tastatur (Hardware–Interrupt) oder durch Systemaufrufe (Software–Interrupt) reagiert werden kann.

4.2 Die Schale

Das Programm Schale dient der Bedienung des Computers durch den Menschen. Zuerst wird der Text `KLAX>` ausgegeben. Danach will die Schale eine Zahl einlesen. Diese Zahl übernimmt die Rolle des Dateinamens der Code–Datei eines Kundenprogrammes.

Das Laden der Code–Datei erfolgt zwar zur Laufzeit, wird jedoch ebenfalls vom Simulator übernommen. Ausgelöst wird es durch Nutzereingabe der Zahl 30, die die Schale einliest (F1) und sofort wieder ausgibt (gibAus). Die Ausgabe dieser Zahl wird vom Simulator abgefangen und veranlaßt ihn zum Einlesen der Code–Datei des Kundenprogrammes. Momentan ist nur eine derartige Code–Datei vorhanden.

Nachdem der Code des Programmes KundeA geladen wurde, muß der Fokus auf die Stromdatei für den Kunden (Nummer 2) gesetzt werden. Das wird in der Schale mit dem Aufruf F3(2) ausgelöst. Danach gelangen bis auf Widerruf alle Tastatureingaben in die zweite Stromdatei.

Wenn der Kunden–Prozeß fertig ist, muß der Fokus der Eingabestromdatei wieder auf die Nummer der Stromdatei der Schale (Nummer 1) gesetzt werden, damit. alle Tastatureingaben wieder von der Schale gelesen werden können.

Über den Systemaufruf F4 wird der Kunde durch die prozeßwechselnde Aktion S_Kunde aktiviert. Nach der Beendigung des Kunden wird der Fokus wieder auf die Stromdatei der Schale gesetzt (F3(1)).

4.3 Der Kunde

Der Kunde sagt HALLO und will danach zwei Zahlen von der Tastatur einlesen. Ist dies geschehen, wird deren Summe berechnet und ausgegeben. Der Kunde kehrt über die prozeßwechselnde Aktion S_Ende (wird vom Compiler erzeugt) zum Verwalter zurück, der daraufhin wieder die Schale aktiviert. Sie macht bei F4 weiter, was aber durch S_Fertig nur die Rückkehr in den Kundenmodus bewirkt.

4.4 Der Leerlaufprozeß

Da i.allg. die Kundeneingaben erst vorgenommen werden nachdem das Nutzerprogramm eine Eingabe vornehmen will (Erreichen des zuständigen Systemaufrufes), muß die Wartezeit überbrückt werden, damit der Prozessor nicht aktiv wartet und dadurch blockiert ist. Diesem Zweck dient hier der Leerlaufprozeß, der das Wort PAUSE ausgibt. Da die letzte Anweisung jedes Kundenprogrammes durch einen prozeßwechselnden Maschinenbefehl mit dem Aufrufwert 0 umgesetzt wird, läuft der Leerlaufprozeß immer wieder neu an, bis ein Hardwareinterrupt ihn unterbricht.

5 Der Faden bei wichtigen Abläufen

Sowohl die Eingabe einer Zahl als auch das Lesen aus dem Eingabestrom führen zum Prozeßwechsel, der im folgenden beschrieben werden soll.

5.1 Die Eingabe einer Zahl über eine simulierte Tastatur

Wenn eine Zahl komplett im Tastaturpuffer vorliegt, wird ein Interrupt mit der Nummer 50 ausgelöst. Der laufende Prozeß wird zu Gunsten des Verwalters unterbrochen, der in Zeile ** wieder anläuft. Der zuständigen Stromdatei wird die Nachricht bedieneTast geschickt, die daraufhin die Zahl von der Tastatur entgegennimmt (jede Stromdatei kann drei Zahlen im voraus entgegennehmen). (Die Umwandlung der Zahl von einer Zeichenkette in eine Binärdarstellung wird vom Simulator durchgeführt.) Wartet schon ein Kunde auf die Zahl, wird er aktiviert, falls nicht, wird der gerade unterbrochene Prozeß reaktiviert.

5.2 Lesen aus dem Eingabestrom

Ein Kunde liest mit dem Systemaufruf F1 aus seiner (der fokussierten) Stromdatei. Das geht natürlich nur, wenn sie nicht leer ist. Ansonsten wird seine Adresse im Feld Kunde der Stromdatei hinterlegt, und er muß durch den Systemaufruf S_Warte solange warten, bis Eingaben verfügbar sind.

6 Zusammenfassung

Das Ziel dieser Arbeit war eine Offenlegung des prinzipiellen Ablaufes an einem Computer bei Interaktion mit peripheren Geräten. Als Beispielgerät wurde die Tastatur gewählt, weil ihre Bedienung von vielen Menschen beherrscht wird.

Im Rahmen dieser Arbeit ist es gelungen, extrem kurze Systemsoftware hundertprozentig problemorientiert zu entwickeln und vorzeigbar zu machen. In kommerziellen Systemen werden für diese Programme mehrere tausend problemorientierte und maschinenorientierte Anweisungen benötigt. Dadurch könnten sie nur mit sehr großem Zeitaufwand in der Lehre vorgezeigt werden, was i.allg. einen Verzicht der Vorstellung dieser wichtigen Programme in der Lehre bedeutet. Der Text des Programmes MiniVerwalter in dieser Arbeit besteht dagegen lediglich aus 300 Zeilen Quelltext.

Durch Einsatz einer selbst entwickelten objektorientierten Programmiersprache konnte der Quelltext der Systemsoftware und der Kundenprogramme in die Maschinensprache einer extrem kleinen und damit sehr effektiv arbeitenden Neumannmaschine übersetzt werden. Die Abarbeitung dieses Maschinencodes wurde mit einem in der Programmiersprache Java entwickelten Simulator–Programmes für die Neumannmaschine ermöglicht und kann über das Internet als Applet genutzt werden.

Ein Demonstrationsbeispiel, das im Internet lauffähig ist, soll die Tragfähigkeit der verwendeten Konzepte beweisen (s.`http://www.wi.hs-wismar.de/~mumm`). Inwieweit es gelungen ist, kann nur die Nutzung zeigen.

Das Verständnis dieser Arbeit soll auch dem Verständnis kommerzieller Programmiersprachen dienen, besitzen sie doch i.allg. Dienstaktionen für die Tastatureingabe.

Ohne die Arbeiten von Prof. Hans Röck aus Rostock wären diese Untersuchungen nicht möglich gewesen, weil sie nur eine Teilfunktionalität seines Betriebssystems Klax näher beleuchten. Deshalb gilt ihm der ganz besondere Dank.

Nachfolgend wird der Quelltext der beteiligten Programme: MiniVerwalter, Schale, KundeA und Leerlauf offengelegt. Im Programm MiniVerwalter werden die beiden Funktionen qgs und mods benutzt. Die Funktion qgs berechnet den ganzzahligen Quotienten zweier positiver ganzer Zahlen und die Funktion mods den Rest bei ihrer Division. Sie werden vom Linker hinzugebunden. In der Methode setzeInDienstModus der Klasse „Prozess“werden von der Variablen Aufruf neun mögliche Werte abgefragt. In dieser Arbeit werden aber nur die ersten fünf dieser Werte benutzt, und zwar für den Aufruf der Methoden F1 bis F5 der Klasse Betrieb durch die Programme Schale und KundeA. Die Startadressen dieser Methoden im Maschinencode können natürlich im Quelltext noch nicht bekannt sein. Sie müssen nach der Übersetzung des Quelltextes und der Verbindung des Objektcodes per Hand im Maschinencode eingetragen werden. Damit der Quelltext übersetzt werden konnte, wurden für diese Adressen die Platzhalter 9999 bis 9991 verwendet.

7 Quelltexte

```
Programm MiniVerwalter;//300 LOC
Konstante OffsetEpa = 3;
Konstante Kunden   =  1999999;
Konstante Schale   = 16777102;
Konstante Leerlauf = 16777211;
Typ Prozess = Klasse (
  Epa   : Adresse;
  Nia   : Adresse;
  Basis : Adresse;
  Akkuw : ganzzahl;

Methode setzeNia(-> N: ganzzahl):
 ganzzahl;
Beginn
 dieses := dieses - OffsetEpa;
    Nia := N ;
 liefereAlsFunktionswert(0);
Ende;
Methode holeBasis(-> N: ganzzahl):
 Adresse;
Beginn
 dieses := dieses - OffsetEpa;
 liefereAlsFunktionswert(Basis);
Ende;
Methode setzeEpaNegativ: ganzzahl;
Beginn
 dieses := dieses - OffsetEpa;
    Epa := 0-Epa;
 liefereAlsFunktionswert(0);
Ende;
Methode setzeAufStart: ganzzahl;
  Beginn
    dieses := dieses - OffsetEpa;
    Nia := Epa ;
    liefereAlsFunktionswert(0);
Ende;
Methode setzeInDienstModus(->Aufruf:
 ganzzahl;->bv: Adresse) : ganzzahl;
Variable Uebergabe: Adresse;
Variable  KDA: Adresse;
Variable rc: ganzzahl;
Beginn
    Uebergabe  := holeBasis+2;
    //RSA,FW,dieses,1.Par.
    Uebergabe^ := bv;
falls Aufruf=1 dann
                rc:=setzeNia(9999);
falls Aufruf=2 dann
                rc:=setzeNia(9998);
falls Aufruf=3 dann
                rc:=setzeNia(9997);
falls Aufruf=4 dann
                rc:=setzeNia(9996);
falls Aufruf=5 dann
                rc:=setzeNia(9995);
falls Aufruf=6 dann
                rc:=setzeNia(9994);
falls Aufruf=7 dann
                rc:=setzeNia(9993);
falls Aufruf=8 dann
                rc:=setzeNia(9992);
falls Aufruf=9 dann
                rc:=setzeNia(9991);
    rc := setzeEpaNegativ;
Ende;
Methode werdeAktiv: ganzzahlImAkku;
Beginn
   aktiviere;
Ende;
                                  );

Typ Strom = Klasse(
buf : Reihe[1..3] von ganzzahl;
pi: 1..4;
po: 1..4;
pp: 0..3;
Kunde: Prozess;

Methode holepp : ganzzahl;
Beginn
liefereAlsFunktionswert(pp);
Ende;

Methode setzeKunde(->p: Prozess):
 ganzzahl;
Beginn
 Kunde := p;
 liefereAlsFunktionswert(0);
```

```
Ende;

Methode initStrom : ganzzahl;
Beginn
 pp := 0;
 pi := 1;
 po := 1;
 Kunde := 0;
 liefereAlsFunktionswert(0);
Ende;

Methode bedieneTast : Prozess;
Variable ch: ganzzahl;
Variable bereit: Prozess;
Beginn
  bereit := 0;
  liesEin(ch); //Zahl von Tastatur
  falls pp < 3 dann  //pk=3
  Beginn
    buf[pi] := ch;
     pi := mods(pi+1,4); //4=pk+1
    falls pi=0 dann pi :=1;
    pp := pp + 1;
  Ende;
  falls Kunde <> 0 dann
    Beginn
       bereit:= Kunde;
       Kunde := 0;
    Ende;
liefereAlsFunktionswert(bereit);
Ende;

 //Zahl vom Puffer
Methode ZahlVP: ganzzahl;
Variable ch: ganzzahl;
Beginn
 pp := pp -1;
 ch := buf[po];
 po := mods(po+1,4);
 falls po=0 dann po := 1;
liefereAlsFunktionswert(ch);
Ende;
);

Typ Betrieb = Klasse(
bereit     : Prozess;
st1        : Strom; //Schale
st2        : Strom; //Nutzer
focus      : ganzzahl;
Methode holeBereit : Prozess;
Beginn
liefereAlsFunktionswert(bereit);
Ende;

Methode holeFocus: ganzzahl;
Beginn
liefereAlsFunktionswert(focus);
Ende;

Methode hoStr1 : Strom;
Beginn
liefereAlsFunktionswert(st1);
Ende;

Methode hoStr2 : Strom;
Beginn
liefereAlsFunktionswert(st2);
Ende;

Methode setzeBereit(-> p: Prozess):
 ganzzahl;
Beginn
bereit:= p;
liefereAlsFunktionswert(0);
Ende;

Methode F1: ganzzahl;
Variable ch: ganzzahl;
Variable pp: ganzzahl;
Variable rc: ganzzahl;
Variable stv: Strom;
Beginn
 falls focus=1 dann stv := st1
              sonst stv := st2;
 pp:= stv.holepp;
 falls pp >= 1 dann  ch:= stv.ZahlVP
   sonst Beginn
 rc := stv.setzeKunde(bereit);
           S_Warte;
           ch := stv.ZahlVP ;
         Ende;
liefereAlsFunktionswert(ch);
S_Fertig;
```

```
Ende;

//Binaer zu Zeichenkette
Methode F2(->z:ganzzahl):ganzzahl;
Variable lauf: ganzzahl;
Variable q: ganzzahl;
Variable r: ganzzahl;
Variable lae: ganzzahl;
Variable  bu: Reihe[1..8] von
 ganzzahl;
Variable flag: ganzzahl;
Beginn
flag:= 0;
lauf:= 0;
falls z < 0 dann Beginn
 z:= 0-z;flag:=1;Ende;
falls z >= 1 dann
Beginn
  solange z >= 1   wiederhole
  Beginn
    q := qgs(z,10);
    r := mods(z,10);
    z:= q;
    lauf := lauf + 1;
    bu[lauf]:=48+r;
    lae := lauf;
  Ende;
  falls flag=1 dann gibAus(45);
  solange lauf >=1 wiederhole
  Beginn
    gibAus(bu[lauf]);
    lauf := lauf -1;
  Ende;
Ende
  sonst
    Beginn
     gibAus(48);  lae:=1; Ende;
gibAus(13);  //CR
liefereAlsFunktionswert(lae);
S_Fertig;
Ende;

Methode F3(->f:ganzzahl):ganzzahl;
Beginn    //fokussiere Stromdatei
focus := f;
liefereAlsFunktionswert(f);
S_Fertig;
Ende;

Methode F4 : ganzzahl;
Beginn  //Schale nach Kunden
  S_Kunde;
  S_Fertig;
liefereAlsFunktionswert(0);
Ende;


Methode F5(-> f: ganzzahl) : ganzzahl;
Beginn
gibAus(f);
liefereAlsFunktionswert(f);
S_Fertig;
Ende;


Methode initBD(-> a1: Adresse;
-> a2: Adresse) : ganzzahl;
Variable rc    : ganzzahl;
Variable hi: Strom;
Beginn
  st1:= a1;
  st2:= a2;
  focus := 1;
  hi := st1;
  rc := hi.initStrom;
  hi := st2;
  rc := hi.initStrom;
 liefereAlsFunktionswert(0);
Ende;
                        );
Variable Kunde  : Prozess;
Variable pro    : Prozess;
Variable Aufruf : ganzzahl;
Variable rc     : ganzzahl;
Variable bv     : Betrieb;
Variable dummy  : Reihe[1..4] von
 ganzzahl;   //Platz fuer Objekte
Variable dummy1 : Reihe[1..7] von
 ganzzahl;
Variable dummy2 : Reihe[1..7] von
 ganzzahl;
//Fuer Stroeme
Variable Uebergabe : Adresse;
```

```
Variable ad1 : Adresse;
Variable ad2 : Adresse;
Variable stv : Strom;
Variable foc: ganzzahl;
Beginn
bv  := &dummy;  //*
ad1 := &dummy1; //*
ad2 := &dummy2; //*
rc  := bv.initBD(ad1,ad2);
rc  := bv.setzeBereit(Schale);
solange wahr wiederhole
 Beginn
  Kunde := bv.holeBereit;
  Aufruf := Kunde.werdeAktiv;
   falls Aufruf=0 dann Beginn //**
     rc := bv.setzeBereit(Leerlauf);
     rc := Kunde.setzeAufStart;
                      Ende
   sonst
   falls Aufruf=50 dann
    //Hardware-Interrupt
    Beginn //Tastatur
       foc := bv.holeFocus;
       falls foc=1 dann stv :=
        bv.hoStr1
                 sonst  stv :=
                  bv.hoStr2;
       pro:= stv.bedieneTast;
        //evt wartet schon einer
     falls pro <> 0 dann
     rc := bv.setzeBereit(pro);
                        Ende
   sonst
   falls Aufruf=95 dann  //S_Ende
      rc := bv.setzeBereit(Schale)
   sonst
   falls Aufruf=96 dann  //S_Kunde
     Beginn
      rc := bv.setzeBereit(Kunden);
      Kunde := bv.holeBereit;
      rc := Kunde.setzeAufStart;
                         Ende
   sonst
   falls Aufruf=97 dann  //S_Warte
      rc := bv.setzeBereit(Leerlauf)
   sonst
   falls Aufruf=98 dann Beginn
     //Sonstiger Hardwareint,
     // Wechsel zu Schale
       Kunde:= Schale;
       rc   := Kunde.setzeAufStart;
                            Ende
     sonst
    falls Aufruf = 99 dann // S_Fertig
         rc := Kunde.setzeEpaNegativ
  sonst  rc:= Kunde.setzeInDienstModus(
     Aufruf,bv);
   Ende;
  Ende!
```

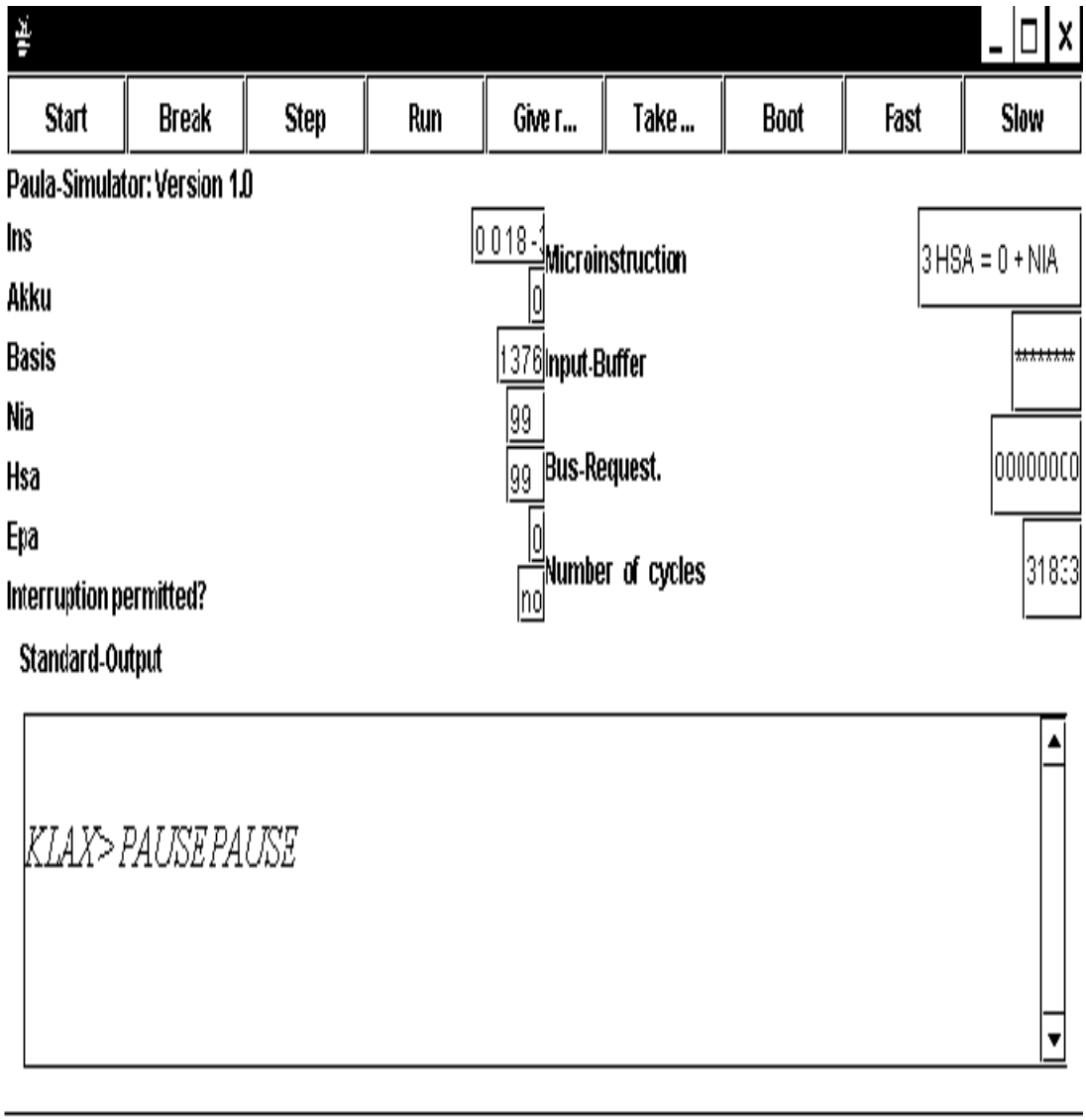
```
Programm Schale;
Typ S_Neutral= Klasse(
Methode F1: ganzzahl;
Methode F2(->p1:ganzzahl):ganzzahl;
Methode F3(->p:ganzzahl):ganzzahl;
Methode F4(->z:Adresse) :ganzzahl;
Methode F5(->as:ganzzahl):ganzzahl;
                  );
Variable v   : S_Neutral;
Variable rc  : ganzzahl;
Variable zahl: ganzzahl;
Beginn
  solange wahr wiederhole
  Beginn
  rc := v.F5(75); //K
  rc := v.F5(76); //L
  rc := v.F5(65); //A
  rc := v.F5(88); //X
  rc := v.F5(62); //>

  zahl := v.F1;
   //lies Zahlencode 30
  gibAus(zahl);
   //Nachricht an Simulator,
   //kunde.pau zu laden

  rc := v.F3(2);
   // setzeFocusAufKundenstrom
  rc := v.F4;
   // setzeKundenBereit
  rc := v.F3(1);
   // setzeFocusAufSchalenstrom
  Ende;
Ende!


Programm KundeA;
Typ S_Neutral= Klasse(
Methode F1: ganzzahl;
Methode F2(->p1: ganzzahl):ganzzahl;
Methode F3(->p : ganzzahl):ganzzahl;
Methode F4(-> z: Adresse) :ganzzahl;
Methode F5(->as: ganzzahl):ganzzahl;);
Variable v       : S_Neutral;
Variable rc      : ganzzahl;
Variable dummy   : ganzzahl;
Variable zahl    : ganzzahl;
Variable Ergebnis: ganzzahl;
Beginn
rc := v.F5(72); //H
  rc := v.F5(65); //A
  rc := v.F5(76); //L
  rc := v.F5(76); //L
  rc := v.F5(79); //0
  rc := v.F5(62); //>
  v       := &dummy;
  zahl    := v.F1; //lies
  rc      := v.F2(zahl)//Anzeige
  Ergebnis:= zahl;
  zahl    := v.F1; //lies
  rc      := v.F2(zahl)//Anzeige
  Ergebnis:= Ergebnis + zahl;
  rc      := v.F2(Ergebnis);
  //zeigeAn
Ende!


Programm Leerlauf;
Typ S_Neutral= Klasse(
Methode F1: ganzzahl;
Methode F2(->p1: ganzzahl):ganzzahl;
Methode F3(->p : ganzzahl):ganzzahl;
Methode F4(-> z: Adresse): ganzzahl;
Methode F5(->as: ganzzahl):ganzzahl;);
Variable v       : S_Neutral;
Variable rc      : ganzzahl;
Variable dummy   : ganzzahl;
Variable zahl    : ganzzahl;
Variable Ergebnis: ganzzahl;
Beginn
  rc := v.F5(80); //P
  rc := v.F5(65); //A
  rc := v.F5(85); //U
  rc := v.F5(83); //S
  rc := v.F5(69); //E
Ende!
```

8 Demonstrationsbeispiel und Literatur

Die Bedienung des Simulators ist relativ einfach. Durch Betätigung der Tasten Boot, Run und Fast(evt. mehrfach) läuft der Simulator mit einer wählbaren Geschwindigkeit an. Die Schale läuft sofort los und zeigt ihre Eingabebereitschaft mit dem Prompt `KLAX>` an. Wenn keine Eingabe erfolgt, wird automatisch der Leerlaufprozeß gestartet, der pausenlos das Wort PAUSE anzeigt. Durch Betätigung der Break-Taste wird nun eine Tastatureingabe simuliert. Es wird dadurch ein Interrupt ausgelöst. Nachdem Paula in den Wartezustand gegangen ist, kann man im Feld Input–Buffer den Interruptcode 50 (hier für Tastatur-Interrupt) eingeben und mit „Give receipt“bestätigen. Die Übernahme einer Zahl von der Tastatur wird durch Eintragung des Wertes in das Feld Input–Buffer simuliert. Mit anschließendem Anklicken von „Give receipt“wird der Wert dann von Klax übernommen und an die Schale übergeben. Die Zahl wird in Form einer Zeichenkette eingegeben und vom Simulator in einen Binärwert überführt, um den Verwalter von dieser Aufgabe zu entlasten. Die Schale gibt anschließend den eingelesenen Wert mit dem Befehl `gibAus` an den Simulator weiter. Wenn es sich bei diesem Wert um den ASCII–Code 30 eines nicht druckbaren Zeichens handelt, liest der Simulator die Code–Datei des Programmes KundeA in den Hauptspeicher der Neumannmaschine auf die feste Adresse 1999999 ein. Der Simulator erwartet normalerweise nur ASCII–Codes von druckbaren Zeichen. Die Zahl 30 kann als interner Name der Code–Datei des Programmes KundeA interpretiert werden.

Diese Vereinfachung wurde gewählt, um den Verwalter vom Laden der Code–Datei des Programmes KundeA zu befreien und nur die Tastatureingaben im Fokus der Arbeit zu belassen. Der Kunde wird gestartet, sagt HALLO und wartet auf Tastatureingaben. Wenn keine vorliegen, wird erneut der Leerlaufprozeß aktiv und sagt PAUSE. Durch das gleiche Prozedere wie bei der Schale: Break, Input-Buffer(50), Give-receipt, Input-Buffer(10), Give-receipt würde der Kunde dann z.B. die Zahl 10 von der simulierten Tastatur einlesen können. Da der Kunde auch noch eine zweite Zahl zur Durchführung einer Addition benötigt, ist dann auf die gleiche Art ud Weise eine zweite Zahl einzulesen. Der Kunde gibt die eingelesenen Zahlen wieder aus und berechnet deren Summe und gibt auch diese Summe wieder aus. Wie schon oben erwähnt, werden die Ausgaben vom Simulator abgefangen und landen in seinem Ausgabefenster. Dazu müssen ggf. Binärwerte zuvor in eine Reihe von ASCII-Codes umgewandelt werden, wozu die Methode `F2` des Verwalters als Systemaufruf genutzt wird. Danach meldet sich wieder die Schale mit dem Prompt `KLAX>`.

Der folgende Schnappschuß entstand nachdem die Schale aktiviert wurde, den Text `KLAX>` angezeigt hat und eine Zahl einlesen wollte. Weil keine Eingabedaten verfügbar waren, hat Klax den Leerlaufprozeß gestartet.



Literatur

[1] **Röck**, Hans (2002): Arbeitsmaterial Betriebssysteme, Universität Rostock, Wintersemester 2002.

[2] **Wirth**, Niklaus/**Gutknecht**, Jürg (1993): Project Oberon, Addison-Wesley 1993, ISBN 0-201-54428-8.

[3] **Tanenbaum**, Andrew S. (1990): Betriebssysteme Teil 1, Carl Hanser Verlag 1990, ISBN 3-446-15268-7.

[4] **Mumm**, Harald (2003): Entwurf und Implementierung einer objektorientierten Programmiersprache für die Paula-Virtuelle-Maschine, Hochschule Wismar, Fachbereich Wirtschaft 2003, WDP – Wismarer Diskussionspapiere Heft 8/2003, ISBN 3-910102-32-8.

Autorenangaben

Prof. Dr. rer. nat. Harald Mumm
Fachbereich Wirtschaft
Hochschule Wismar
Philipp-Müller-Straße 14
Postfach 12 10
D – 23966 Wismar
Telefon: ++49 / (0)3841 / 753 450
Fax: ++49 / (0)3841 / 753 131
E-mail: harald.mumm@wi.hs-wismar.de

WDP - Wismarer Diskussionspapiere / Wismar Discussion Papers

Heft 01/2003	Jost W. Kramer: Fortschrittsfähigkeit gefragt: Haben die Kreditgenossenschaften als Genossenschaften eine Zukunft?
Heft 02/2003	Julia Neumann-Szyszka: Einsatzmöglichkeiten der Balanced Scorecard in mittelständischen (Fertigungs-)Unternehmen
Heft 03/2003	Melanie Pippig: Möglichkeiten und Grenzen der Messung von Kundenzufriedenheit in einem Krankenhaus
Heft 04/2003	Jost W. Kramer: Entwicklung und Perspektiven der produktivgenossenschaftlichen Unternehmensform
Heft 05/2003	Jost W. Kramer: Produktivgenossenschaften als Instrument der Arbeitsmarktpolitik. Anmerkungen zum Berliner Förderungskonzept
Heft 06/2003	Herbert Neunteufel/Gottfried Rössel/Uwe Sassenberg: Das Marketingniveau in der Kunststoffbranche Westmecklenburgs
Heft 07/2003	Uwe Lämmel: Data-Mining mittels künstlicher neuronaler Netze
Heft 08/2003	Harald Mumm: Entwurf und Implementierung einer objektorientierten Programmiersprache für die Paula-Virtuelle-Maschine
Heft 09/2003	Jost W. Kramer: Optimaler Wettbewerb – Überlegungen zur Dimensionierung von Konkurrenz
Heft 10/2003	Jost W. Kramer: The Allocation of Property Rights within Registered Co-operatives in Germany
Heft 11/2003	Dietrich Nöthens/Ulrike Mauritz: IT-Sicherheit an der Hochschule Wismar
Heft 12/2003	Stefan Wissuwa: Data Mining und XML. Modularisierung und Automatisierung von Verarbeitungsschritten
Heft 13/2003	Bodo Wiegand-Hoffmeister: Optimierung der Sozialstaatlichkeit durch Grundrechtsschutz – Analyse neuerer Tendenzen der Rechtsprechung des Bundesverfassungsgerichts zu sozialen Implikationen der Grundrechte -
Heft 14/2003	Todor Nenov Todorov: Wirtschaftswachstum und Effektivität der Industrieunternehmen beim Übergang zu einer Marktwirtschaft in Bulgarien
Heft 15/2003	Robert Schediwy: Wien – Wismar – Weltkulturerbe. Grundlagen, Probleme und Perspektiven
Heft 16/2003	Jost W. Kramer: Trends und Tendenzen der Genossenschaftsentwicklung in Deutschland
Heft 01/2004	Uwe Lämmel: Der moderne Frege
Heft 02/2004	Harald Mumm: Die Wirkungsweise von Betriebssystemen am Beispiel der Tastatur-Eingabe