Projekt Paketsortieranlage (RTAI)

Echtzeitsysteme
sechstes Semester Technische Informatik
an der Hochschule für Technik in Ulm

Eduard Heidt und Jan Helber

Diese Dokumentation beschreibt die Implementierung einer Paketsortieranlage unter dem Echtzeitbetriebssystem RTAI

Powered by LaTeX
Generiert am 29. Juni 2007 um 18:17

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

1 Einleitung

Im Fach Echtzeitsysteme des sechsten Semesters der Fachhochschule Ulm bearbeiteten wir das Projekt Paketsortierung. Die Software soll die Steuerung der gegebenen Komponenten so übernehmen, dass korrekt eingelesene Pakete an der richtigen Stelle ausgeworfen werden und fehlerhaft gescannte Pakete nicht ausgeworfen werden sondern am Ende vom Band herunter fallen.

2 Aufbau der Anlage

Es existieren folgende Komponenten:

- Strichcodeleser
- 2 Bänder
- 3 Auswurfstationen
- 5 Lichtschranken
- Schieber

Für die Ansteuerung dieser Komponenten (mit Ausnahme des Strichcodeleser) ist eine Schnittstellenkarte in den Rechner eingebaut. Der Strichcodeleser ist an der seriellen Schnittstelle angeschlossen.

2.1 Strichcodeleser

Der Strichcodeleser erlaubt die Erfassung einer Adresse zur Bestimmung der Auswurfposition. Diese Komponente muss getriggert werden. Die serielle Schnittstelle zur Abtastung des Strichcodes muss mit den üblichen Mitteln einer Hochsprache angesprochen werden (nicht auf Registerebene). Das Lesen des Strichcodes ist zeitunkritisch und erfolgt außerhalb des Echtzeitbereiches.

2.2 Bänder

Das erste Förderband transportiert die unbekannten Pakete zum Strichcodeleser, während das zweite Förderband die Pakete zu den verschiedenen Auswurfstationen bewegt. Beide Förderbänder können nicht rückwärts bewegt werden, sondern lediglich ein- und ausgeschaltet werden. Daher ist es notwendig, dass die Anwendung die Auswurfstationen und Bänder in Echtzeit ansteuert.

Abbildung 1: Die Paketsortieranlage

2.3 Auswurfstationen & Schieber

Die Auswurfstationen sowie der Schieber werden pneumatisch betrieben. Der Schieber[1] erlaubt die Separierung von Paketen.

2.4 Lichtschranken

Um die aktuelle Position der Pakete zuverlässig verfolgen zu können, müssen 5 eingebaute Lichtschranken (zwischen den verschiedenen Auswurfstationen) in definierten zeitlichen Abständen abgefragt werden.

3 Aufgabenstellung

Für die Paketsortierungsanlage ist eine Steuerungs-Software zu realisieren, welche Pakete an den korrekten Auswurfpositionen auswirft.

Außerdem ist darauf zu achten, dass das nächste Paket durch den Schieber entsperrt wird, sobald die erste Auswurfsposition wieder frei ist. Es soll also insbesondere ein Paket nicht erst dann eingelassen werden, wenn das vorherige die Anlage bereits verlassen hat.

[1] Der Schieber wird in einigen unserer Grafiken und Diagramme auch Sperre genannt.

Die Verwendung eines Echtzeitsystems ist unbedingt notwendig, da die Förderbänder nicht rückwärts bewegt werden können und der Zeitpunkt für den Auswurf des Paketes nicht verpasst werden darf. Außerdem muss für alle Ereignisse die parallel[2] auftreten können garantiert werden, dass diese zu einem bestimmten Termin abgearbeitet sind.

Das Scheduling der Echtzeitkritschen Tasks soll über die Erweiterung RTAI[3] oder durch Jamaica[4] erfolgen. Im folgenden werden die einzelnen Tasks beschrieben.

4 Realisierung

Wir realisierten das Projekt in RTAI, da wir uns erstens mit dieser Art der Programmierung besser auskannten und zweitens mit eclipse und Jamaica jede Menge Probleme hatten. In eclipse durfte man z.B. an bestimmten Stellen nicht auf die rechte Maustaste kommen, da es ansonsten abstürzte.

Außerdem verwendeten wir in unserem Programm den One-Shot-Mode da die verschiedenen Tasks mit unterschiedlichen Perioden liefen und die Ansteuerung des Strichcodelesers sehr genau erfolgen musste, um einen reibungslosen Ablauf zu gewährleisten.

[2] Z.B.: Paket1 passiert Lichtschranke4, während Paket2 Lichtschranke3 und Paket3 Lichtschranke2 passiert

[3] RTAI (Real Time Application Interface) ist eine Erweiterung von Linux zu einem Echtzeitbetriebssystem. [Wikipedia RTAI]

[4] JamaicaVM ermöglicht die deterministische Ausführung von Java Applikationen bei zugleich niedrigem Speicherbedarf und hoher Laufzeitleistung. [Aicas]

5 Implementierung

Die Paketsortieranlage wird mit 3 Tasks implementiert. Zwei Tasks (der Band-Task und der Scan-Task) sind periodisch, der dritte Task (Auswerfen der Pakete) wird vom Band-Task getriggert. In der Abbildung 5 sieht man den Logischen Aufbau der Paketsortieranlage, in der Abbildung 5 die Sequenzdiagrame der Tasks. Im folgenden Unterabschnitten werden die Tasks näher berschrieben.

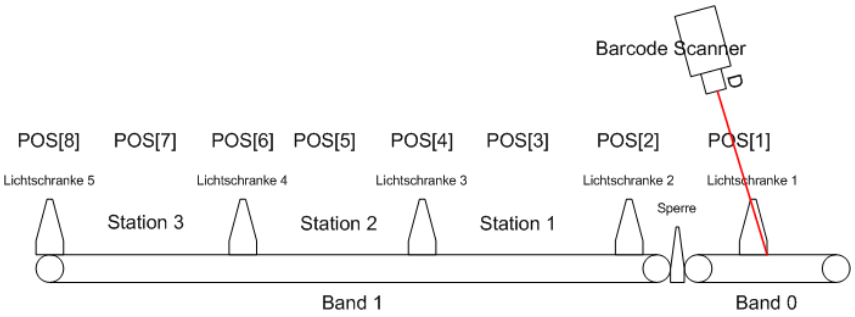

Abbildung 2: Logischer Aufbau

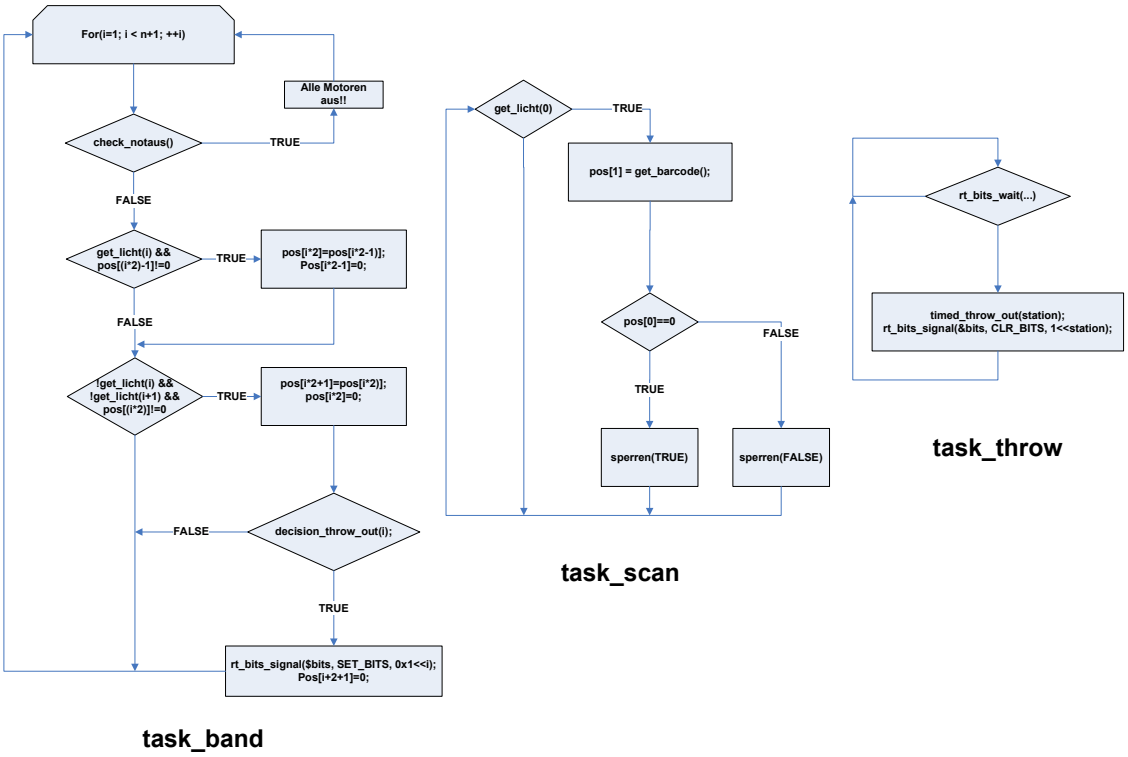

Abbildung 3: Tasks

5.1 Band-Task

Der Band-Task ist für das Band 1 zuständig. Primäraufgabe ist die Pakete anhand der Lichtschranken-Signale zwischen den Positionen zu verschieben (umzukopieren). Ist ein Paket auf einer Auswurf-Position, so muss entschieden werden ob es ausgeworfen werden soll. Bei Positiver Entscheidung wird über *rt_ bit_ signal()* ein Auswurf getriggert, welcher durch den Auswurf-Task durchgeführt wird. Um genau zu sehen was in diesem Task passiert siehe Sourcecode im Listing 1.

5.1.1 Semaphoren

Bei unserem Programm muss gewährleistet sein, dass verschiedene Tasks welche die selbe Ressource benötigen, nicht gleichzeitig auf diese zugreifen. Daher wurde der Zugriff auf diese Resource in unserem Programm durch eine Semaphore geschützt. Auch wenn ein Task die Variable ändert und alle anderen nur lesend darauf zu greifen, kann dies zu unvorhersehbaren Ergebnissen führen, wenn der Wechsel zwischen den Tasks an einer ungünstigen Stelle geschieht. Mehr dazu unter http://de.wikipedia.org/wiki/Kritischer_Abschnitt

Listing 1: Band-Task

```
1  void task_band(long n)
2  {
3   int i = 0;
4   while(1)
5   {
6    for(i=1; i<n+1; i++)
7    {
8     if(check_notaus(i))
9      continue;
10    // Wenn lichtschranke dann Packet in ächste Postion und Throw down
11    if(get_licht(i) && pos[(i*2)-1] != 0)
12    {
13     rt_sem_wait(&positions);
14     pos[i*2]=pos[(i*2)-1];
15     pos[i*2-1]=0;
16     rt_sem_signal(&positions);
17    }
18    // Wenn nichts auf Lichtschranken dann Packen in ächste postion
19    if(!get_licht(i) && !get_licht(i+1) && pos[(i*2)] != 0)
20    {
21     rt_sem_wait(&positions);
22     pos[(i*2)+1]=pos[i*2];
23     pos[i*2]=0;
24     rt_sem_signal(&positions);
25     print_pos();
26
27     if(decision_throw_out(i)) // Entscheiden ob ßRausschmeien oder nicht
28     {
29      rt_bits_signal(&bits, SET_BITS, 0x1<<i);
30      rt_sem_wait(&positions);
31      pos[(i*2)+1] = 0; // falls ja Pos = 0;
32      rt_sem_signal(&positions);
33     }
34    }
35    band_motor(0, !get_licht(0));
36   }
37   rt_task_wait_period();
38  }
39 }
```

5.2 Scan-Task

Der Scan-Task ist für das Erfassen von Paketen auf Band 0 zuständig. Ist die Lichtschranke 0 aktiv so wird der Scanner getriggert. Solange kein Barcode erfasst ist, ist die Sperre aktiv. Bei Positivem Scanvorgang wird die Sperre deaktiviert, und das Paket wird auf das Hauptband transportiert. Um genau zu sehen was in diesem Task passiert siehe Sourcecode im Listing 2.

Listing 2: Scan-Task

```
void task_scan(long n)
{
 while(1)
 {
  if(get_licht(0))
  {
   rt_sem_wait(&positions);
   pos[1] = get_barcode();

   if(pos[1]!=0)   // Wenn nichts auf Position dann Sperren
    sperren(FALSE);
   else
    sperren(TRUE);

   rt_sem_signal(&positions);

  }
  else
   barcode_flag = 0;

  rt_task_wait_period();
 }
}
```

5.3 Auswurf-Task

Der Auswurf-Task wird durch den Band-Task getriggert. Wird eine Auswurf getriggert so läuft eine Auswurf-Sequenz ab. Die Auswurf-Sequenz (timed_throw_out()) besteht aus dem Aktivieren des pneumatischen Schiebers, einer Wartezeit bis der Auswerfer wirklich oben ist damit das Paket ausgeworfen wird. Um genau zu sehen was in diesem Task passiert siehe Sourcecode im Listing 3.

Listing 3: Auswurf-Task

```
static void task_throw(long station)
{
 unsigned long resulting_mask;

 while(1)
 {
  rt_bits_wait(&bits, ALL_SET, 0x1<<station, NOP_BITS, 0, &resulting_mask);
  timed_throw_out(station);
  rt_bits_signal(&bits, CLR_BITS, 0x1<<station);
 }
}
```

5.4 Strichcodeleser

Der Barcode wird folgendermaßen gelesen, siehe Sourcecode im Listing 4.

5.4.1 Initialisierung

Um über die RS232-Schnittstelle mit dem Strichcodeleser eine Verbindung herstellen zu können, musste die Schnittstellen-Karte mit folgenden Parametern initialisiert werden:

- Baudrate: 9600 bps
- 8 Datenbits, Odd Parity, 1 Stop Bit
- FIFO Control Register
- AN: DTR, RTS, OUT2

Listing 4: Barcode Funktionen

```
1
2  long get_barcode(void)
3  {
4   int j = 0, i = 0;
5   long ret = 0;
6   if(barcode_flag!=0)
7     for(j=1; j<9; ++j)
8     {
9       ret *= 10;
10      ret += (barcode[i+j]-'0');
11    }
12  return ret;
13 }
14
15 void read_barcode(long t) {
16  //int barcode_new=0;
17  set(SCANNER, FALSE);
18  int i=0;
19  for(i=0; i<=19; i++){
20    barcode[i]='\0';
21  }
22  i=1;
23  while(1) {
24    if(inb(COM1+5) & 1) {//char erhalten?
25      ch = inb(COM1); //lesen ...
26      if(ch==13)
27      {
28        barcode[i]='\0';
29        barcode_flag=1;
30        i++;
31        set(SCANNER, TRUE);
32        rt_sleep(SCANNER_SLEEP); //ansonsten scannt er nicht
33        set(SCANNER, FALSE);
34      }
35      else if(barcode[i-1]=='\0') //letztes Zeichen nach dem RETURN weg werfen und i auf 0 setzen
36      {
37        i=0;
38      }
39      else
40      {
41        if(barcode[i]!=ch)
42        {
43          barcode[i]=ch;
```

```
    }
    i++;
  }
 }
 rt_task_wait_period();
}
}
```

5.5 Maschinensteuerung

Der folgende Sourcecode 5 zeigt von den Tasks verwendete Unterfunktionen.

Listing 5: Maschinensteuerung

```
int get(int b)
{
return inb (IO_PORT) & b;
}

void set(int b, BOOL value)
{
static int reg = 0xff;

if(value)
 reg |= b;
else
 reg &= 0xff^b;
outb(reg, IO_PORT);
}

void band_motor(int i, BOOL on)
{
switch(i)
{
 case 0: set(BAND0, !on); break;
 case 1: set(BAND1, !on); break;
}
}

void throw_out(int i, BOOL on)
{
switch(i)
{
 case 1: set(THROW1, !on); break;
 case 2: set(THROW2, !on); break;
 case 3: set(THROW3, !on); break;
}
}

void sperren(BOOL on)
{
set(SPERRE, !on);
}

void timed_throw_out(int i)
{
rt_sleep(1500000000);
throw_out(i, TRUE);
rt_sleep(2000000000);
throw_out(i, FALSE);
}

BOOL get_licht(int i)
{
return get(LICHT0<<i);
}
```

6 Inbetriebnahme

6.1 RTAI

Um echtzeitfähige Anwendungen einsetzen zu können musste zunächst RTAI installiert werden. RTAI läuft dabei in der ADEOS-Domäne. Wir haben den Linux-Kernel gemäß Labordokumentation mit RTAI gepacht und im System verankert. Zum compilieren des Kernels wurde von uns das Makefile 6 verwendet.

Listing 6: Makefile

```
obj-m := paket.o

KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
EXTRA_CFLAGS := -I/usr/realtime/include -I/usr/src/linux/include -I/usr/include

default:
 $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

clean:
 rm -r .tmp_versions
 rm .`basename $(obj-m) .o`.*
 rm `basename $(obj-m) .o`.o
 rm `basename $(obj-m) .o`.ko
 rm `basename $(obj-m) .o`.mod.*
```

6.2 Automatisches compilieren und insmod der Kernelmodule

Das Shell-Skript 7 wurde von uns eingesetzt um die Kernelmodule zunächst mit dem Kommando rmmod zu deaktivieren, zu compilieren und danach per insmod wieder zu aktivieren. Es ersparte uns jede Menge Tipparbeit während der Entwicklungsphase.

Listing 7: loadmod.sh

```
while(true);
do
  rmmod DeutschePost
  rmmod rtai_wd
  rmmod rtai_sem
  rmmod rtai_bits
  rmmod rtai_sched
  rmmod rtai_hal
  dgau 0xff
  read test
  cd /home/TI6/DeutschePostRTAI/
  make
  insmod /usr/realtime/modules/rtai_hal.ko
  insmod /usr/realtime/modules/rtai_ksched.ko
  insmod /usr/realtime/modules/rtai_bits.ko
  insmod /usr/realtime/modules/rtai_sem.ko
  insmod /usr/realtime/modules/rtai_wd.ko
  insmod ./DeutschePost.ko
  read test
done
```

7 Durch dieses Projekt gelernt und Bekannte Probleme

- Auch der Watchdog-Timer von RTAI rettet das Betriebssystem nicht, wenn man nicht alle Tasks mit rt_task_delete am Ende löscht.
- Latex in Verbindung mit SVN sind unschlagbar.
- Notaus Hand in Lichtschranke benötigt Neustart
- Wenn nicht alle Tasks mit `exit_module()` beendet werden, stürzt der Rechner beim nächsten insmod des Kernelmodules ab.
- Wir wollten soweit wie möglich auf `rt_busy_sleep()` verzichten, um den Programmablauf nicht zu stören. Daher haben wir an solchen Stellen `rt_bits_wait()` verwendet und mit unserer Funktion `timed_throw_out()` die entsprechenden Bits nach einer bestimmten Zeit setzen lassen. Wir benötigten `rt_busy_sleep()` nur ein einyiges mal wegen der seriellen Schnittstelle zum Strichcodescanner. `rt_busy_sleep()` blockiert das System. Dies stört den Ablauf des Programmes nicht. Da an dieser stelle das Programm allerdings sequenziell abläuft wird auch das Suse Betriebsystem blockiert, welches die übrigen Ressourcen zum Betrieb nutzt.
- Zunächst hatten wir einen Task, für jede Station. Dieser Task wartete auf das Lichtschranken-Signal, gab das Paket an die nächste Station weiter, wartete dann eine bestimmte Zeit, damit das Paket auch wirklich an der Lichtschranke vorbei ist und warf es dann gegebenenfalls aus.
  Daher musste ein Paket erst aus einer Station entfernt sein bevor das nächste bearbeitet werden konnte. Durch aufsplitten dieser vielen Funktionen in einem Task auf mehrere Tasks konnte das Problem sauber gelöst werden. Der Abstand zwischen den Paketen auf dem Band konnte deutlich verringert werden. Besonders wichtig dabei ist, dass die Weitergabe der Pakete von einer Position zur nächsten nicht vom selben Task bearbeitet wird, wie der Auswurf.

8 Zusammenfassung

Wir konnten die Aufgabenstellung in vollem Umfang lösen. Für den reibungslosen Ablauf eines unserer Test-Programme war der Schieber gar nicht mehr notwendig, da unsere Pakete ohne probleme mit sehr geringem Abstand auf dem Band unterwegs waren. Wir starten das Transportband zum Strichcodeleser nicht erst (wie in der Aufgabenstellung gefordert) sobald das vorherige Paket die erste Auswurfstation verlassen hat, sondern schon sobald das Paket die zweite Lichtschranke passiert hat, sich also vollständig in der erste Auswurfsposition befindet.

9 Quellen

Literatur

[Wikipedia RTAI] http://de.wikipedia.org/wiki/RTAI

[Aicas] http://www.aicas.com/newsletters/newsletter_2002_2.pdf