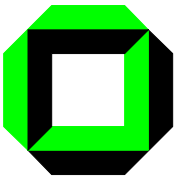

Universität Karlsruhe
Institut für Mikrorechner und Automation
Prof. Dr. rer. nat. U. Brinkschulte
Dipl. Inform. Holger Vogelsang

Haid- und Neu-Str. 7
76131 Karlsruhe
Tel. 0721 / 608 - 3170
Fax. 0721 / 661732
vogelsang@ira.uka.de

Auftragskonzept

Dokumentation Version 2.0
Holger Vogelsang, Markus Voss, Stefan Breker
16. Juli 1996

Inhalt

Die vorliegende Dokumentation beschreibt die durch das Auftragskonzept definierten Operationen zur Organisation von parallelen Prozessen (Dienstprozessen) und deren Verwendung bei einer Programmierung unter. Zur Motivation der Operationen und zur formalen Definition der Semantik wird auf das interne Papier [Voß: Ein Auftragssystem] verwiesen.

Umgebung

Ist das Auftragskonzept auf einem PC installiert, so hat dieser die Unterverzeichnis \sys_base\ak und \sys_base\kernel. Das Verzeichnis \sys_base\ak beinhaltet alle systemunabhängigen Module und \sys_base\kernel umfaßt alle systemabhängige Module. Im Programm zu inkludieren ist die Schnittstelle ak_func.h.

Parametrisierung

Zum run-time Test der Programme ist eine Möglichkeit der Aus- und Einschaltung von Terminalausgaben fast aller Systemroutinen realisiert. Diese wird durch die folgenden Parameter gesteuert:

DEBUG	Bei TRUE werden Informationen durch Systemroutinen des Prozeßsystems ausgegeben.
DEBUG_A	Bei TRUE werden Informationen zur Auftragsverwaltung ausgegeben.
DEBUG_NR	Bei TRUE werden Informationen zur Nachrichtenverwaltung ausgegeben.
DEBUG_AB	Bei TRUE werden Informationen zur Ablagenverwaltung ausgegeben.
DEBUG_N	Bei TRUE wird jede Prozeßumschaltung angezeigt.
DEBUG_TR	Bei TRUE wird jede Aktion des Transportprozesses angegeben.

NAME

Initialisiere_System - System einrichten

SYNOPSIS

#include "ak_func.h"

void Initialisiere_System(char*HPN, Boole Network, Boole Lightweight)

BESCHREIBUNG

Diese Operation muß im Anwenderprogramm zu Beginn einmal aufgerufen werden. Mit ihr werden Prozeß- und Auftragssystem initialisiert. Im einzelnen werden folgende Schritte durchgeführt:

- Das laufende Programm wird als erster Prozeß unter dem in **HPN** definierten Namen eingetragen.
- Das (lokale) Ablagensystem wird initialisiert.

Abhängig von den bool'schen Parametern werden folgende weitere Schritte durchgeführt:

- Wenn **Network** = TRUE gilt, wird ein „Transportprozeß“ unter dem Namen „TRANS“ erzeugt. Dieser realisiert die Kommunikation mehrerer Prozesse eines verteilten Systems.

- Wenn **Lightweigt** = TRUE gilt, werden alle Prozesse als Threads eingerichtet, sonst werden die Prozesse unter UNIX als „normale“ Prozesse gestartet.

BEISPIEL

```
:
#include "ak_func.h"
:
main()
{
        :
        Initialisiere_System("HP",TRUE,TRUE);
        :
}
```

Alle Prozesse werden als Threads eingerichtet und nach dem Aufruf existieren die Prozesse „HP“, „TRANS“. Alle Prozesse sind „nicht wartend“. „HP“ ist „aktiv“.

NAME

Erzeuge_Prozeß - Prozeßerzeugung (Dienst-Einrichtung)

SYNOPSIS

#include "ak_func.h"

P_Nummer Erzeuge_Prozess(void(*Proc)(), char *name, int Stack)

BESCHREIBUNG

Mit Aufruf von **Erzeuge_Prozess(Proc,Name,Stack)** wird unter dem Klarnamen **Name** ein „Light-Weight-Process“, d.h. ein eigenständiger Kontrollfluß (Thread) erzeugt, der mit dem ersten Befehl der Prozedur **Proc** beginnt. Es wird also quasi „aus einer Prozedur ein Prozeß gemacht“. Dabei ist es möglich, die gleiche Art von Kontrollfluß, d.h. die gleiche Prozedur, mehreren Prozessen zuzuweisen.

Ein auf diese Art erzeugter Prozeß existiert zuerst in einem nicht wartenden, nicht aktiven Zustand. (Die Kontrolle geht nicht direkt an den neu erzeugten Prozeß über).

Die Größe des Prozeßstacks kann mit **Stack** spezifiziert werden. Als aktueller Parameter ist hier die Auswahl von **AUTOSTACK** möglich.

BEISPIEL

"Speisende Philosophen"

```
#include "ak_func.h"

philosoph()
{
        :
}

main()
{
        :
        Initialisiere_System("HP",FALSE,TRUE);
        :
        Erzeuge_Prozess(philospoh,"P1",AUTOSTACK);
        Erzeuge_Prozess(philosoph,"P2",AUTOSTACK);
        Erzeuge_Prozess(philosoph,"P3",AUTOSTACK);
        Erzeuge_Prozess(philosoph,"P4",AUTOSTACK);
        Erzeuge_Prozess(philosoph,"P5",AUTOSTACK);
        :
}
```

NAME

Terminiere_Prozeß - Prozeßterminierung (Dienst-Terminierung)

SYNOPSIS

#include "ak_func.h"

Terminiere_Prozess(P_Nummer p_no)

BESCHREIBUNG

Mit Aufruf von **Terminiere_Prozess(P_Nummer)** wird der angegebene Prozeß beendet.

NAME

Erteile_Auftrag, Erwarte_Fertig, Erwarte_Auftrag, Melde_Fertig - Die zentralen Operationen der Auftragsschicht.

SYNOPSIS

#include "ak_func.h"

Fehlercode Erteile_Auftrag(Auftrag A)
Fehlercode Erwarte_Fertig(Auftrag A, Ergebnis E)
Fehlercode Erwarte_Auftrag(Auftrag A)
Fehlercode Melde_Fertig(Auftrag A, Ergebnis E)

BESCHREIBUNG

Diese Operationen sind zentral für das gesamte Konzept. Bei ihrer Benutzung ist die folgende Logik zu berücksichtigen:

Mit **Auftrags- bzw. Ergebnisformular** wird eine Variable vom Typ Auftrag bzw. Ergebnis bezeichnet. Ein Formular ist wie ein Betriebsmittel zu betrachten. Auftragnehmer und -geber arbeiten mit verschieden Kopienen eines Auftrags und dessen Inhalts

Der Typ Auftrag hat folgendes Aussehen:

```
typedef struct {
        P_Name      Auftraggeber,
                    Auftragnehmer;
        N_Typ       Typ;
        A_Zustand   Zustand;
        V_Nummer    Vorgang;
        A_Typ       ATyp;
        I_Typ       Inhalt;
}       Auftrag;
```

Die Operationen haben die folgende Semantik:

Ein Aufruf **Erteile_Auftrag(A)**, wobei **A** ein Auftragsformular ist, bewirkt logisch, daß das Formular in die Ablage des Prozesses mit der Kennzeichnung A.Auftragnehmer abgelegt wird. De Facto wird es nur dann in die Ablage eingefügt, wenn der Auftragnehmer lokal ist (A.Auftragnehmer.Rechner ist der lokale Host). Im anderen Fall wird der Auftrag in die Ablage des Transporteurs abgelegt. Wartet der Auftragnehmer auf diesen Auftrag (er hat dann zuvor **Erwarte_Auftrag** aufgerufen, s. unten), so wird dieser deblockiert. Zudem wird die Schedulingroutine **NEXT()** aufgerufen (s. dort).

Der Aufruf **Erwarte_Fertig(A,E)** bewirkt das Blockieren des Aufrufers (Zustand 'wartend'), bis der zuvor erteilte Auftrag **A** durch den Aufragnehmer mittels **Melde_Fertig** (s. unten) fertig gemeldet wird. Wird bei dieser Fertigmeldung ein Ergebnis übergeben, so kann dieses mit dem Ergebnisformular **E** referenziert werden. Wird kein Ergebnis erwartet, so kann **Erwarte_Fertig** mit dem Parameter **ERWARTE_KEIN_ERGEBNIS** aufgerufen werden.

Erwarte_Auftrag(A) bewirkt, daß der Aufrufer blockiert wird, bis ein Auftrag an ihn erteilt, d.h. in seine Ablage eingefügt wurde. Der Auftrag kann nach Rücksprung aus dieser Operation mit dem Auftragsformularzeiger A referenziert werden.

Der Aufruf **Melde_Fertig(A,E)** schließlich bewirkt sowohl das Deblockieren eines evtl. auf diese Fertigmeldung wartenden Auftraggebers, als auch das Ablegen eines evtl. Ergebnisses. Soll kein Ergebnis zurückgegeben werden, so kann **Melde_Fertig** mit dem Parameter **MELDE_KEIN_ERGEBNIS** aufgerufen werden. Zudem wird hier, wie auch in **Erteile_Auftrag**, die Scheduleroutine **NEXT()** aufgerufen (s. dort).

BEISPIEL

```
/* Prozess P*/
:
Auftrag     A;
Int         I;
Ergebnis    E;
float       F;
:
I = 4;
Initialisiere_Formular ( A );
Setze_Inhalt ( A , I );
Setze_Auftragnehmer ( A , "WS","i50s8" )
Erteile_Auftrag ( A );
Erwarte_Fertig ( A , E );
Lese_Inhalt ( E , F );
printf ("Wurzel  = %f\n", F);
```

```
"Wurzel-Server" auf Rechner i50s8
:
/* Prozess WS */

Auftrag     A;
int         I;
Ergebnis    E;
float       F;
:
Endlosschleife {
    Erwarte_Auftrag ( A );
    Lese_Inhalt ( A , I );
    F = sqrt ( I );
    Setze_Inhalt ( E , F );
    Melde_Fertig ( A , E );
}
```

RETURN VALUES

Ein Rückgabewert kleiner 0 (**F_KEINF**) bedeutet einen Fehler. Die Routinen **Erteile_Auftrag** und **Melde_Fertig** liefern bei fehlerhafter Ausführung die verbleibende Kapazität der betreffenden Ablage zurück.

Mögliche Fehlercodes:
F_FORMNFREI - Das als Parameter angegebene Formular ist für die Operation noch nicht freigegeben.
F_ABLVOLL - Die Ablage ist voll. Der Auftrag bzw. das Ergebnis kann nicht abgelegt werden.

NAME

Existiert_Auftrag, Ist_Fertig - Die Informationsfunktionen der Auftragsschicht.

SYNOPSIS

#include "ak_func.h"

Boole Existiert_Auftrag()
Boole Ist_Fertig(Auftrag A)

BESCHREIBUNG

Diese beiden Informationsfunktionen sind primär zur Realisierung von nicht-blockierenden Abläufen (alternativ zu den Operationen Erwarte_Auftrag und Erwarte_Fertig) gedacht.

Existiert_Auftrag() ermittelt, ob in der Ablage des Aufrufers ein Auftrag vorliegt.
Ist_Fertig(A) ermittelt, ob der Auftrag **A** 'durchgeführt' ist.

RETURN VALUES

Beide Funktionen liefern entweder TRUE, wenn das abgeprüfte Kriterium gilt, oder FALSE anderenfalls.

NAME

Initialisiere_Formular, Beende_Vorgang - Zusätzliche Verwaltungsfunktionen der Auftragsschicht.

SYNOPSIS

#include "ak_func.h"

void Initialisiere_Formular(Auftrag A)
void Beende_Vorgang(Ergebnis E)

BESCHREIBUNG

Vor der erstmaligen Benutzung eines Auftragsformulars empfiehlt es sich, **Initialisiere_Formular(Auftrag A)** aufzurufen. Ohne diesen Aufruf kann es in Ausnahmesituationen dazu kommen, daß **Erteile_Auftrag(A)** mit Fehler **F_FORMNFREI** abbricht.

Beende_Vorgang(E), wobei **E** ein auf Auftraggeberseite geführter Ergebnisformularzeiger ist, bewirkt, daß alle vom System dynamisch erzeugten Daten wieder aufgelöst werden. **Beende_Vorgang** sollte immer dann aufgerufen werden, wenn der Auftraggeber weder das Ergebnisformular, noch den damit verbundenen Inhalt weiter benötigt.

BEISPIEL

Die Makros werden schon im Beispiel der Verwendung der Grundoperationen mitbenutzt.

NAME

Setze_Inhalt, Lese_Inhalt, Setze_Auftragnehmer, Setze_Auftragstyp - Operationen zum Formular-Handling

SYNOPSIS

#include "ak_func.h"

void Setze_Inhalt(Auftrag A, void I)
void Setze_Inhalt(Ergebnis E, void I)

void Setze_Auftragstyp(Auftrag A, A_Typ AT)
void Setze_Auftragnehmer(Auftrag A, char *PName, char *RName)

void Lese_Inhalt(Auftrag A, void I)
void Lese_Inhalt(Ergebnis E, void I)

BESCHREIBUNG

Die vier obigen Makros erleichtern das Beschreiben bzw. Auslesen von Formularen. So findet beispielsweise die Zuordnung eines Inhalts **I** (beliebigen Typs!) zu einem Formular **A**, d.h. die Belegung der Struktur Inhalt[1] durch den Macro **Setze_Inhalt(A,I)** statt.

Mit **Setze_Auftragnehmer(A,PName,RName)** wird der Prozeß mit dem Klarnamen **PName** auf dem Rechner mit dem Klarnamen **RName** als Auftragnehmer in das Formular **A** eingetragen[2]. Als aktueller Parameter für **RName** ist **DIESER_RECHNER** möglich. Es wird dann der Name des lokalen Hosts eingetragen.

Setze_Auftragstyp(A,AT) belegt das Auftragstypenfeld im Formular mit **AT**[3].

Lese_Inhalt(A,I) kopiert den Inhalt eines Auftrags in die Variable I

BEISPIEL

Die Makros werden schon im Beispiel der Verwendung der Grundoperationen mitbenutzt.

[1] A.Inhalt.Adresse=&I
[2] A.Auftragnehmer.Prozess=Nummer_von(PName,RName)
A.Auftragnehmer.Rechner=IP_to_Long(Name_to_IP(RName))

NAME

NEXT - Schedulingroutine

SYNOPSIS

#include "ak_func.h"

void NEXT()

BESCHREIBUNG

Mit einem Aufruf der Routine **NEXT()** gibt der gerade aktive Prozeß die Kontrolle (den Prozessor) an den nächsten, nicht wartenden Prozeß ab. Die Ermittlung, welcher Prozeß 'der Nächste' ist, wird Scheduling genannt. In der derzeitigen Version 1.0 ist der nächste Prozeß immer derjenige nicht wartende Prozeß, der am längsten nicht mehr über die Kontrolle verfügte.

Diese Strategie ist für Echtzeitsysteme nicht adäquat! Hier muß die Vergabe der Kontrolle abhängig von der Auftragslage geschehen, wobei ein Charakteristikum eines Auftrags sein Termin ist. Ein internes Papier über Zeitanforderungen in formalen Modell und Scheduling im Auftragssystem ist z.Zt. in Arbeit. Die Version 2.0 des Auftragskonzepts wird die Ergebnisse berücksichtigen!

NAME

Ausgabe_Ablagensystem, Ausgabe_Ablage,Ausgabe_Ablageninhalt, Ausgabe_Ablage_Komplett, Ausgabe_Auftrag - Terminalausgaben

SYNOPSIS

#include"ak_func.h"

void Ausgabe_Ablagensystem()
void Ausgabe_Ablage()
void Ausgabe_Ablageinhalt()
void Ausgabe_Ablage_Komplett()
void Ausgabe_Auftrag(Auftrag*A)

BESCHREIBUNG

Die Operationen geben Informationen über das System zum Zeitpunkt des Aufrufs auf dem gerade aktiven Ausgabegerät aus.

BEISPIEL

Der Aufruf von Ausgabe_Ablage() liefert ein Resultat wie:

```
--------------------SYSTEM-INFORMATION-----------------
-
Ablage    :    Prozess 3
Belegt    :    3
First          :    4         Last      :    8

-------------------------------------------------------
```

Der Aufruf von Ausgabe_Ablagensystem() liefert ein Resultat wie:

```
---------------------SYSTEM-INFORMATION-----------------
-
Ablagensystem
Kapazität :    1000
Davon Belegt:  11        frei      :    989
FirstFree :    401       LastFree  :    378
```

NAME

Nummer_von, Mein_Name, Name_to_IP,IP_to_Long-Operationen der Prozeßnamensverwaltung.

SYNOPSIS

#include "ak_func.h"

P_Nummer Nummer_von(char*PName, char *RName)
P_NummerMeine_Nummer()

char* Mein_Name()
char* Name_to_IP(char*RName)
unsigned long IP_to_Long(char*IPA)

BESCHREIBUNG

Ein Aufruf von Nummer_von(PName,RName) liefert die lokale Prozeßnummer (ID) des Prozesses in Klarnamen PName auf Rechner RName zurück.

Aufrufe von Main_Name() bzw. Meine_Nummer() liefern Klarname bzw. lokale Prozeßnummer des Aufrufenden Prozesses zurück.

Name_to_IP(RName) gibt zum Klarnamen RName eines Rechners dessen Internetadresse (als String) zurück. IP_to_Long(IPA) konvertiert eine solche Internetadresse in einen Long-Integer-Wert (4 Byte). Dies entspricht dem Typ R_Name.

RETURN VALUES

Im Fehlerfall (eines nicht interpretierbaren Parameters) wird ein Wert kleiner als 0 (F_KLEINF) bzw. Nullzeiger zurückgeliefert.

NAME

Ausgabe_Aktivmeldung,Ausgabe_Aktiv,Ausgabe_Systemstatus-Terminalausgaben

SYNOPSIS

#include"ak_func.h"

void Ausgabe_Aktivmeldung()
void Ausgabe_Aktiv()
void Ausgabe_Systemstatus()

BESCHREIBUNG

Am zum Zeitpunkt des Aufrufs aktiven Ausgabegerät wird bei Aufruf folgendes ausgegeben:

Bei Ausgabe_Aktivmeldung() eine Aktivmeldung mit Klarnamen des aktiven Prozesses.

Bei Ausgabe_Aktiv() die lokale Prozeßnummer des aktiven Prozesses.

Bei Ausgabe Systemstatus() eine Information der folgenden Form:

```
----------------------SYSTEM-INFORMATION-------------------
 Name Nummer Host         Zustand Wartend Prio
      HP   1    lokal               0    0         6
 NULLP     2    lokal               0    0         4
 TRANS     3    lokal               0    1         6
 P1   4    lokal               1    0         6
------------------------------------------------------------
Aktiv:                          P1                          (4)
------------------------------------------------------------
```