D A S T E I L N E H M E R S Y S T E M M O T U S

E i n l e i t u n g :

MOTUS (MASTER oriented timesharing user system) ist ein Bildschirm-Dialog-System, das am Zentrum für Datenverarbeitung der Justus Liebig-Universität Gießen für die Rechenanlage CD3300 vom Verfasser entwickelt und implementiert wurde.
Die Arbeiten hierzu begannen im September 1973. Die erste Version von MOTUS wurde Oktober 74 freigegeben. Anfängliche Schwierigkeiten wurden beseitigt und seit November 74 läuft die zweite Version des Systems sehr zufriedenstellend täglich 5 bis 8 Stunden und bedient dabei 3 bis 5 im Hause befindliche (lokale) Bildschirmgeräte. Mit der Ende Januar 75 verabschiedeten dritten MOTUS-Version können auch außer Haus aufgestellte Geräte (remote displays) bedient werden. Das geschieht mit Unterstützung der von CDC (Control Data Corporation) entwickelten Kommunikationssoftware MCS3 (Message Control System). Bei der geplanten und zum großen Teil schon verwirklichten Gießener Konfiguration werden dann bis zu 18 Bildschirmgeräte unter MOTUS bedienbar sein.
Ursprünglich sollte das System die Möglichkeit eines vollständigen Remote-Job-Entry haben, das heißt es sollte dazu dienen, über Bildschirme einerseits Dateien (insbesondere Jobs) aufzubauen und zu verändern und andererseits Jobs in die Warteschlange für die Stapelverarbeitung des Rechners einzureihen.
Im Laufe der Entwicklung des Systems hatte sich gezeigt, daß mit einem geringen Mehraufwand auch in höheren Sprachen geschriebene Dialogprogramme unter der Kontrolle von MOTUS möglich sind. Dazu ist lediglich ein entsprechendes Interface zu einem Standardpaket von Macros für jede Programmiersprache erforderlich. Auch können betreffende Dialogprogramme reentrant geschrieben werden, sodaß sie in der Lage sind, von mehreren Bildschirmgeräten kommende Aktivitäten aufzunehmen und zu verarbeiten.

KONZEPT DES TEILNEHMERSYSTEMS

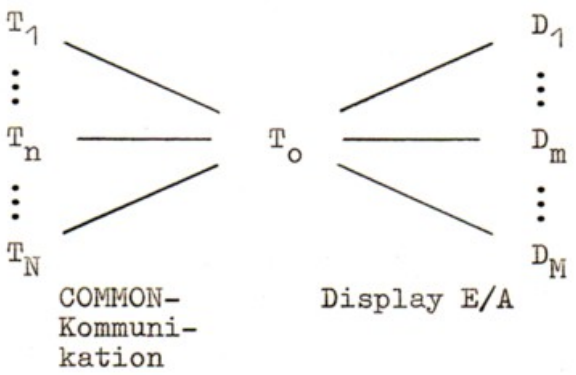
Bei der Entwicklung von MOTUS wurde im wesentlichen Wert auf die folgenden Punkte gelegt:

- einfache Erweiterbarkeit des Systems
- kurze Antwortzeiten
- benutzerfreundlich
- geringe Maschinenanforderungen

Vorausgesetzt wurde ferner ein Multiprogramming Betriebssystem (an der CD 3300 MASTER), in dem mehrere Programmeinheiten (Tasks) quasi gleichzeitig bearbeitet werden können. Es ergab sich dann das folgende Konzept.

A u f b a u :

MOTUS besteht aus einer Grundtask T_o und mehreren Arbeitstasks T_1 - T_N. Die gesamte Bildschirm Ein- und Ausgabe, die zentrale Verwaltung und einige weniger aufwendige Befehle führt die Grundtask durch. Alle anderen Aktivitäten, die viel Zeit oder Kernspeicher benötigen, werden von der jeweils zuständigen Arbeitstask übernommen. Solche befinden sich nur im Kernspeicher, wenn sie gebraucht werden und verlassen ihn nach einer gewissen inaktiven Zeitspanne. Wegen der Bildschirm-E/A verbleibt T_o dagegen dauernd im Kernspeicher.
Zur Verständigung der Tasks T_o, T_1, ... ,T_N untereinander gibt es einen gemeinsamen Kernspeicherbereich (COMMON), auf den sie alle Zugriff haben. Sollen insgesamt M Displays D_m bedienbar sein, so ergibt sich die folgende Konstellation:



Wird an einem der Displays D_m eine Zeichenkette abgeschickt, so aktiviert T_o die zuständige Arbeitstask T_n, die dann den im COMMON stehenden String entsprechend verarbeitet und verändert. Bei Beendigung der Arbeit übergibt T_n die Kontrolle für D_m an die Grundtask T_o und damit gelangt die modifizierte Zeichenkette als Antwort auf den Bildschirm D_m. T_o ist jetzt wieder bereit, weitere Eingaben von D_m entgegenzunehmen.
Unter den T_1 - T_N gibt es zwei Arten von Tasks. Das sind die sogenannten impliziten und expliziten Arbeitstasks. Implizite Tasks verarbeiten nur spezielle Zeichenketten (Befehle) der Gestalt 'Befehlskopf','Befehlsrumpf' .
Sie werden aktiviert, wenn einer ihrer Befehle eingegeben wurde. Eine explizite Task T_n dagegen wird explizit vom Teilnehmer an D_m durch einen T_n-spezifischen Einschaltbefehl aktiviert und bleibt solange mit ihm verbunden, bis er den Ausschaltbefehl RETURN gegeben hat. Während dieser Zeitspanne sind an D_m nur Eingaben möglich, die T_n verarbeiten kann.

K o m m u n i k a t i o n :

Der gemeinsame Kernspeicherbereich enthält für jedes D_m einen Kommunikationsbereich K_m und einen E/A-Puffer P_m. Die P_m werden dynamisch verwaltet und enthalten jeweils den Display-E/A-Puffer und einen Platten-E/A-Kernspeicherbereich. Die wesentlich kleineren M Bereiche K_m liegen fest im COMMON. Jeder enthält eine Variable V_m , durch die die zentrale Kommunikation gesteuert wird. V_m zeigt jeweils den gegenwärtigen Zustand des Displays D_m an. Sowohl die Grundtask T_o, als auch alle Arbeitstasks müssen ständig über die V_m (m=1,...,M) informiert sein. Das geschieht in jeder Task über eine sogenannte Kommunikationsschleife, in der sie auf seine V_m-Werte abfragt. V_m kann sein:

2n+1 T_o hat vom Display D_m entweder einen Befehl erhalten, den die implizite Task T_n ausführen kann oder den Einschaltbefehl der expliziten Task T_n. Ist T_n bereits im Kernspeicher, so erkennt sie in der Kommunikationsschleife den sie zur Arbeit auffordernden Wert 2n+1, andernfalls wird T_n noch vorher von T_o gerufen.

2n Die explizite Task T_n ist mit dem Display D_m verbunden und bleibt zuständig. Für implizite Tasks ist dieser Wert bedeutungslos.

1 T_o erwartet vom Display D_m den LI-Befehl (den eine Sitzung einleitenden Befehl).

+0 Das letzte Kommando von D_m wurde von einer impliziten Task ausgeführt. T_o schrieb die Antwort auf D_m und las dann einen neuen Befehl, von dem T_o jetzt feststellt, welche implizite Task ihn ausführen kann bzw. ob es sich um einen Einschaltbefehl einer expliziten Task handelt.

-0 Die letzte Aktivität für D_m wurde von einer expliziten Task ausgeführt. T_o schrieb die Antwort, las einen neuen Bildschirminhalt und setzt nun V_m=2n.

-1 K_m ist frei für die Zuordnung eines neuen D_m.

-2 E/A-Anforderung einer impliziten Task.

-3 E/A-Anforderung einer expliziten Task.

-4 Der Teilnehmer an D_m hat den Ausschaltbefehl RETURN einer expliziten Task gegeben.

-5 Der Teilnehmer an D_m war zu lange inaktiv mit einer expliziten Task verbunden. T_o hat deswegen den Ausschaltbefehl RETURN simuliert.

G r u n d t a s k :

Das in Fig. 1 vereinfacht dargestellte Flußdiagramm der Grundtask T_o zeigt die Kommunikationsschleife mit den Abfragen auf die für T_o wichtigen V_m-Werte und die anschließenden Aktionen. V_m=-4 und V_m=-5 wurden der Übersicht wegen zusammengefaßt. V_m=-1 muß unberücksichtigt bleiben, da dem K_m kein D_m zugeordnet ist. Nur wenn sich ein neuer Teilnehmer anmeldet, werden solche toten Kommunikationsbereiche (K_m mit V_m=-1) in der Initialisierung für D_m wieder benutzt, das heißt sie bekommen einen Puffer P_m zugeordnet und den Wert 1 für V_m.
Die eigentliche Aktivierung der Arbeitstasks geschieht hinter den ja-Ausgängen der Abfragen V_m=-0 und V_m=+0. Im ersten Falle mußte die vorangegangene Aktivität für D_m von einer expliziten im zweiten Falle von einer impliziten Task gekommen sein. Während einerseits die Anweisung V_m:=2n eine explizite Task veranlaßt die neue von D_m kommende Eingabe zu verarbeiten, wird

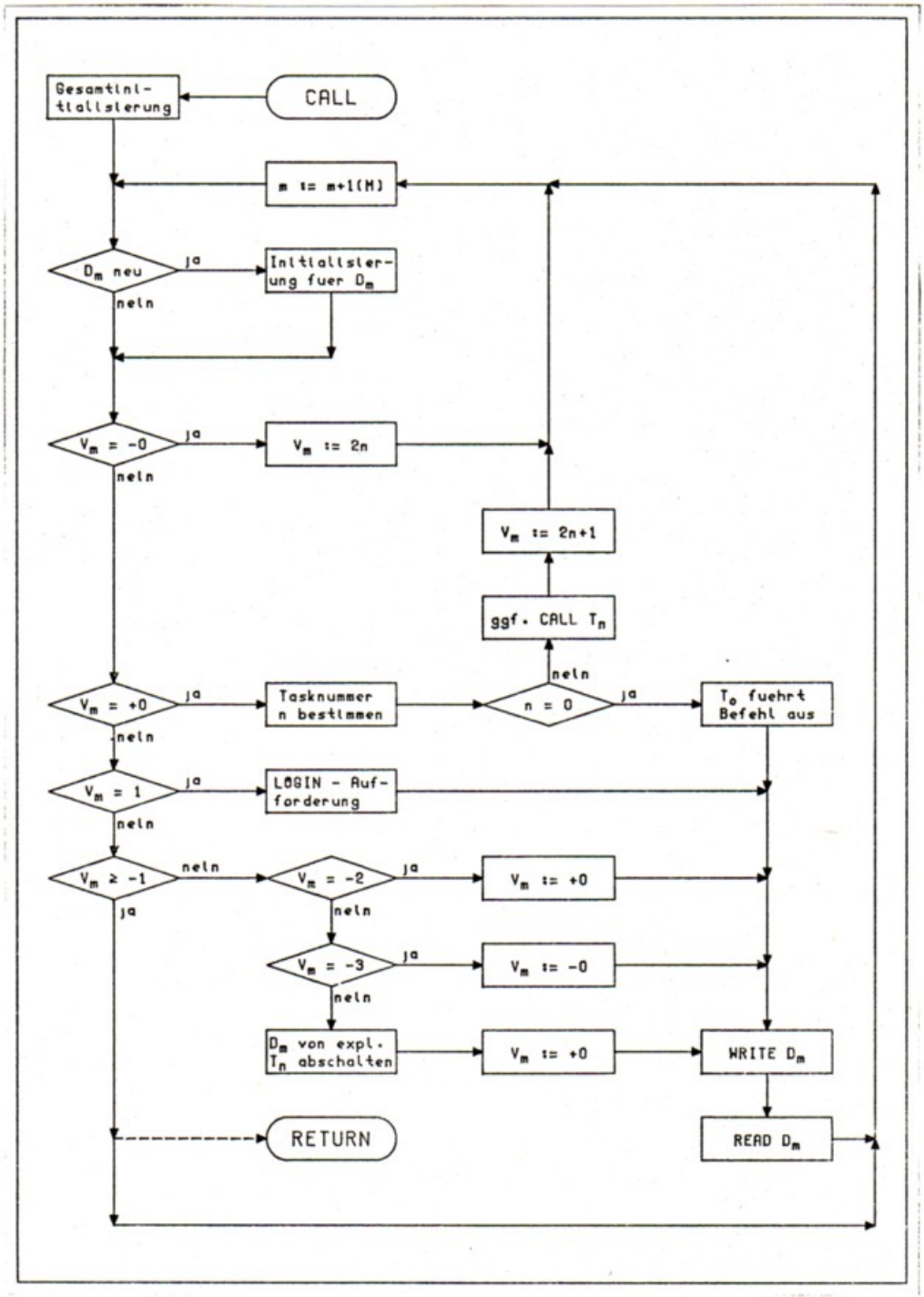


Fig. 1: Kommunikationsteil der Grundtask T_o in vereinfachter Darstellungsweise.

andererseits eine implizite Task nach Ermittlung der Tasknummer $n \neq 0$ und der Instruktionsnummer aus der Befehlstabelle durch V_m:=2n+1 aktiviert, das betreffende Kommando abzuarbeiten. Ausnahme hierbei sind die Einschaltbefehle für die verschiedenen expliziten Tasks.

Ist n=0, so handelt es sich um einen Befehl, den T_o selbst ausführen kann. Beispiele dafür sind die Kommandos LI und LO. Mit LI meldet sich ein weiterer Teilnehmer an, mit LO verabschiedet sich ein Benutzer von MOTUS. Die Anzahl der von T_o versorgten Teilnehmer wird in der COMMON-Größe AD_o geführt. Ist AD_o=0, so kann T_o alle präsenten Arbeitstasks zum Verlassen des Kernspeichers auffordern und schließlich selbst verschwinden (return).

I m p l i z i t e A r b e i t s t a s k :

Bei einer impliziten Arbeitstask T_n muß die Kommunikationsschleife den V_m-Wert 2n+1 abfragen (siehe Fig. 2). Nachdem T_n von T_o gerufen wurde, beginnt sie sofort alle V_m (m=1,...,M) auf den Wert 2n+1 hin zu prüfen. Hat T_n ein V_m=2n+1 gefunden, so führt sie denjenigen Befehl aus, dessen Nummer von T_o über K_m an T_n übergeben wurde. Anschließend wird V_m=-2 gesetzt und die Kommunikationsschleife läuft weiter ab. Jedesmal wenn sie alle V_m (m=1,...,M) einmal erfolglos (ohne einen Wert 2n+1) durchsucht hat (leere Runde), geht T_n für eine gewisse setzbare Zeit $TURNTIME_n$ (zur Zeit 400 msec) in Pause. Anschließend wird eine für T_n bereit gestellte COMMON-Größe $TIME_n$ erhöht, welche die hintereinander vorkommenden leeren Runden zählt. Übersteigt $TIME_n$ eine gewisse setzbare Grenze $TIMELIM_n$ und ist die im COMMON geführte Anzahl AD_n derjenigen Displays gleich Null, für die T_n noch arbeiten muß, dann fordert die Grundtask T_n zum Verlassen des Kernspeichers auf, indem sie AD_n=-1 setzt. Dabei kann der Kernspeicherbedarf KE_n von T_n zurückgegeben werden zu Gunsten anderer Arbeitstasks oder zu Gunsten weiterer Teilnehmer (P_m-Bereiche).

E x p l i z i t e A r b e i t s t a s k :

Eine analoge Situation finden wir bei den expliziten Arbeitstasks vor (siehe Fig. 3). Hier müssen in der Kommunikationsschleife zwei V_m-Werte abgefragt werden. Wird ein V_m=2n+1 gefunden, so hat sich der Teilnehmer an D_m durch den Einschaltbefehl mit T_n verbunden, was ihm auch nun von T_n bestätigt wird. V_m=2n bedeutet, daß eine weitere Eingabe von D_m angekommen ist und nun von T_n verarbeitet werden soll. Danach wird

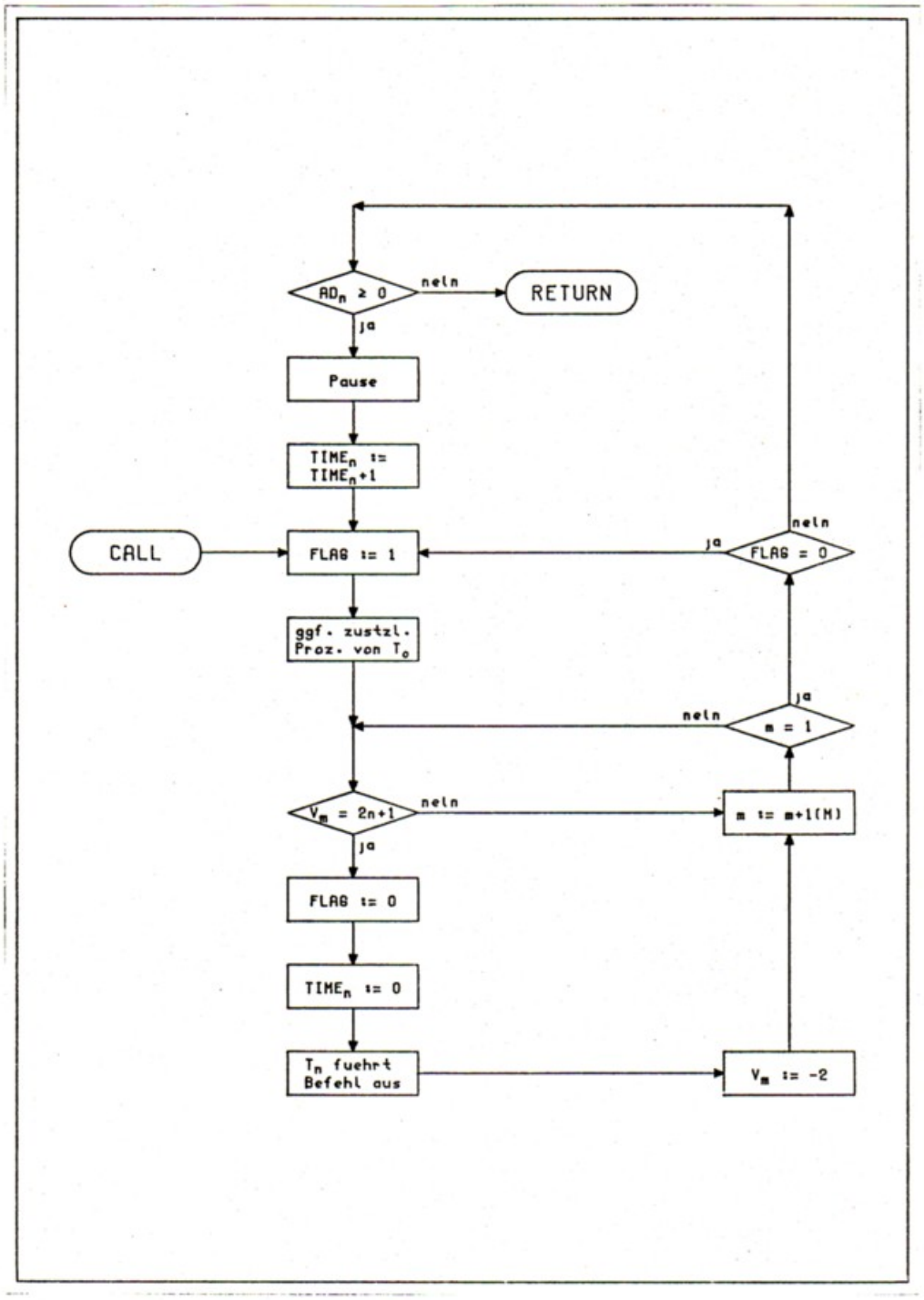


Fig. 2: Kommunikationsteil einer impliziten Arbeitstask.

V_m=-3 gesetzt. Vor der weiteren Verarbeitung wird jede Eingabe daraufhin untersucht, ob die ersten 6 Zeichen mit dem String RETURN übereinstimmen. In diesem Fall hat sich D_m von T_n ausgeschaltet. Dieser Sachverhalt wird durch Setzen von V_m=-4 an

T_o weitergegeben.

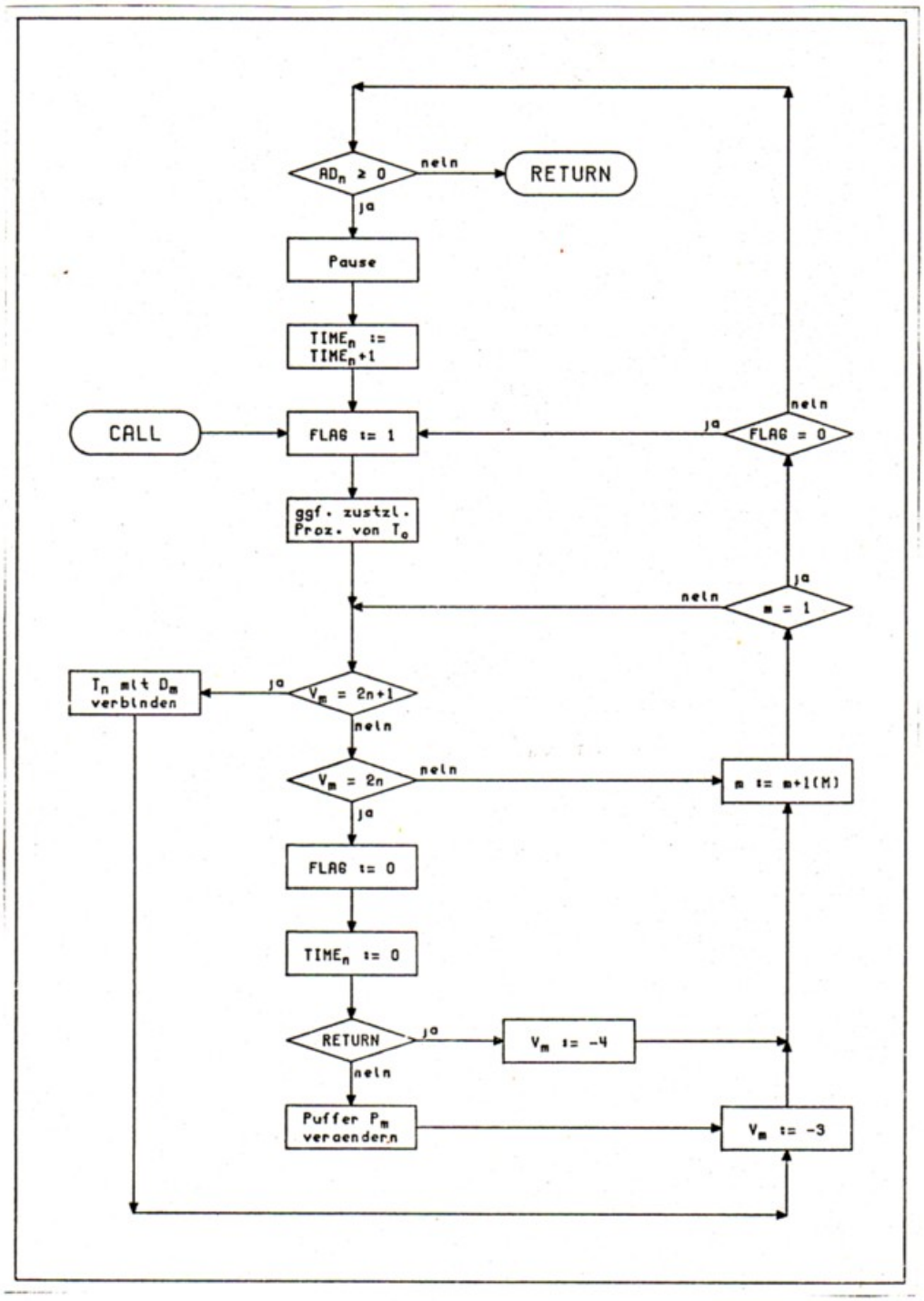


Fig. 3: Kommunikationsteil einer expliziten Arbeitstask.

P r o z e s s o r m ö g l i c h k e i t :

Sowohl implizite als auch explizite Arbeitstasks können neben der von den Teilnehmern kommenden über T_o weitergeleiteten Arbeitsaufträge auch noch sogenannte Prozesse für T_o ausführen, die davon unabhängig auftreten. Solche Prozesse sind zeit- oder kernspeicheraufwendige Arbeiten, die T_o veranlaßt aber nicht selbst ausführt, damit ihre verwaltende Tätigkeit nicht zu lange blockiert wird. Die dazu notwendige Verständigung zwischen T_o und T_n geschieht über die COMMON-Größe $MOTUSPRZ_n$. Für T_o steht kein Prozeß mehr zur Bearbeitung an, wenn $MOTUSPRZ_n=0$ ist. In der Kommunikationsschleife von T_n wird nach jeder Runde genau einmal geprüft, ob ein Prozeß für T_o durchzuführen ist oder nicht.

K e r n s p e i c h e r g e s a m t a b r e c h n u n g :

Das Teilnehmersystem MOTUS führt intern eine Abrechnung für den momentanen Kernspeicherbedarf durch, damit der anfangs mitgegebene Vorrat zu keiner Zeit überschritten wird. Sowohl der durch neu hinzukommende Teilnehmer als auch der durch präsente Arbeitstasks benötigte Kernspeicher wird aus einem "Topf" genommen und abgerechnet. Wird dabei mehr Kernspeicher gebraucht als zur Verfügung steht, so informiert MOTUS den verursachenden Teilnehmer mit einer "Engpaßmeldung" (siehe Fig. 4).
Eine Arbeitstask T_n , die einen Prozeß ausführen soll, muß aus der Kernspeicherabrechnung herausgenommen werden ($KE_n=0$). Denn für sie darf kein Kernspeicherengpaß auftreten, damit sich die Bearbeitung des Prozesses nicht verzögert. Für eine derartige Arbeitstask muß daher ständig Kernspeicher zur Verfügung stehen, damit sie jederzeit rufbar ist.

A k t i v i e r u n g s v o r b e r e i t u n g e n ,
F o l g e t a s k s :

Bevor die Grundtask die Arbeitstask T_n durch Setzen von $V_m:=2n+1$ aktivieren kann, muß sie noch zusätzliche Vorbereitungen treffen. Diese sind in Fig. 1 durch "ggf. CALL T_n" angedeutet und werden im Folgenden erläutert.
Für jede Arbeitstask T_n ist AD_n die Anzahl der noch zu bearbei-

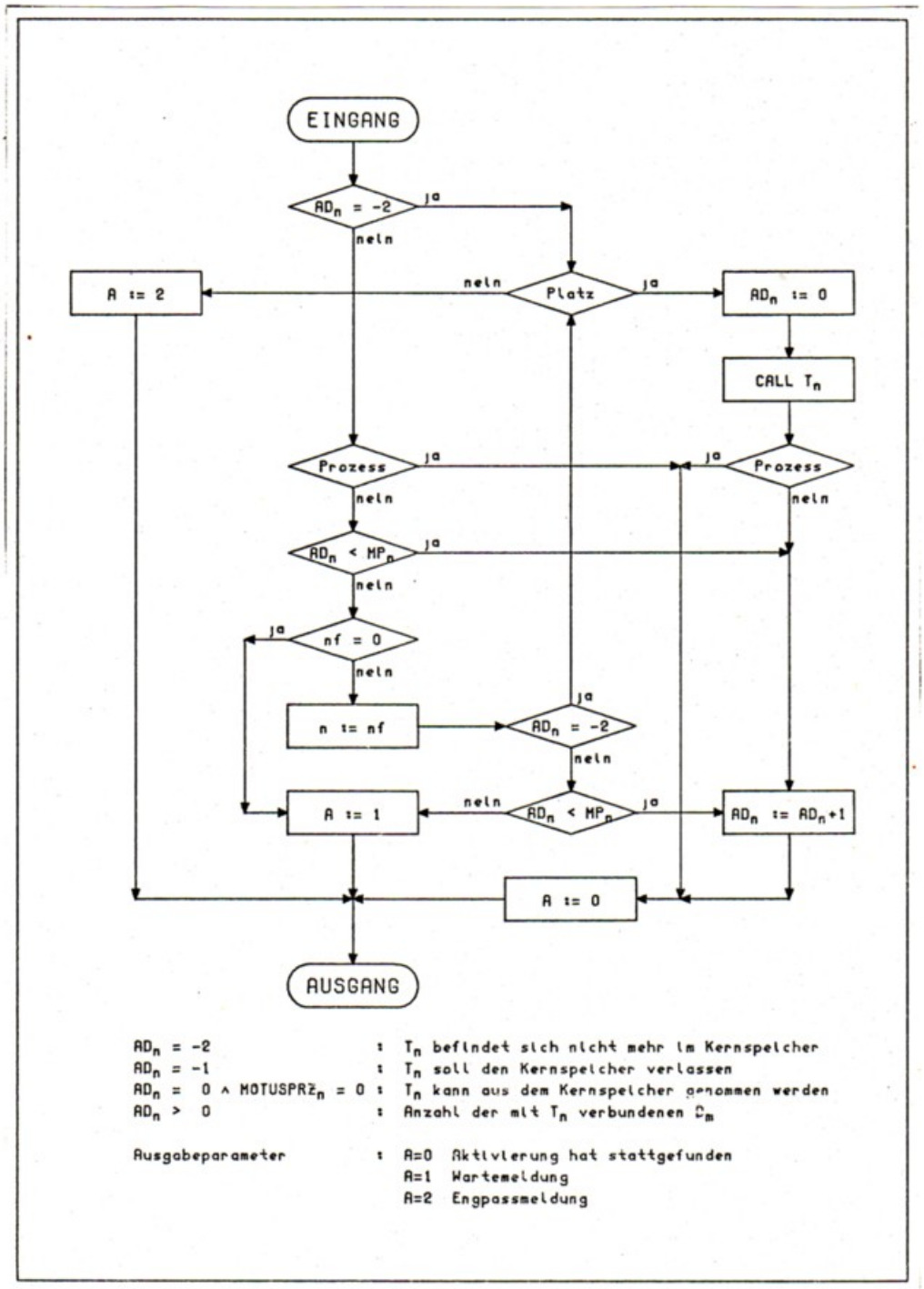


Fig. 4: Aktivierungsvorbereitungen in der Grundtask.

tenden Teilnehmeraufträge bei impliziten Tasks und die Anzahl der mit T_n verbundenen Displays bei expliziten Arbeitstasks. Übersteigt diese Größe die Maximalanzahl MP_n, für die T_n höchstens zuständig sein darf (siehe Fig. 4), so wird der verur-

sachende Teilnehmer durch eine "Wartemeldung" informiert. Ist $MP_n \geq$ der Maximalanzahl MP_o der Displays, die T_o höchstens versorgt, dann werden die Arbeitsaufträge in eine Warteschlange eingereiht und es kommt zu keiner "Wartemeldung".
Im praktischen Betrieb eines Teilnehmersystems wird es sicher vorkommen, daß die einzelnen Arbeitstasks unterschiedlich stark benutzt werden (zum Beispiel T_2 100 Aktivitäten und T_3 400 Aktivitäten pro Stunde).
Um oft benutzte Tasks zu entlasten, besteht für jede Task T_n die Möglichkeit, ihr eine sogenannte Folgetask T_{nf} zuzuordnen. T_{nf} ist eine Kopie von T_n mit anderem Namen und anderer Nummer und kann selbst keine Folgetask haben. Ebenso dürfen Tasks keine Folgetask besitzen, die für T_o Prozesse ausführen sollen. T_{nf} kommt immer dann zum Einsatz, wenn $AD_n > MP_n$ wird (siehe Fig. 4). Sie greift gewissermaßen hilfreich zu, wenn T_n besonders stark belastet ist und verkürzt damit die Wartezeiten der Teilnehmer, die T_n beauftragt haben. Eine "Wartemeldung" erscheint jetzt nur dann, wenn $AD_{nf} > MP_{nf}$ und $MP_n + MP_{nf} < MP_o$ ist.

ANSCHLUSSMÖGLICHKEIT VON ARBEITSTASKS

Wesentlichen Einfluß auf die Konzeption von MOTUS hatte die Forderung der einfachen Erweiterbarkeit des Systems durch den Anschluß weiterer Arbeitstasks genommen. Diese Möglichkeit soll im Folgenden erläutert werden.

F o r m a l e r A u f b a u :

Eine Arbeitstask T_n besteht aus vier Segmenten (siehe Fig. 5). Einmal die COMMON-Definition, der T_o-Anschluß, der Teil, in dem die von den Teilnehmern abgeschickten T_n-Aufträge bearbeitet werden und schließlich im letzten Abschnitt die Abwicklung der von T_o beauftragten Prozesse. Die ersten beiden Segmente treten bei allen Arbeitstasks in gleicher Weise auf und können daher standartisiert werden. Der dritte Teil hängt von der Syntax und Semantik der P_m-Inhalte ab, die von T_o an T_n übergeben werden. Um auch diesen Abschnitt zu vereinheitlichen liegt es nahe, ge-

wisse Grundprozeduren in Form von Macros bereit zu stellen (etwa solche, die Abbau und Aufbau des Puffers P_m unterstützen).

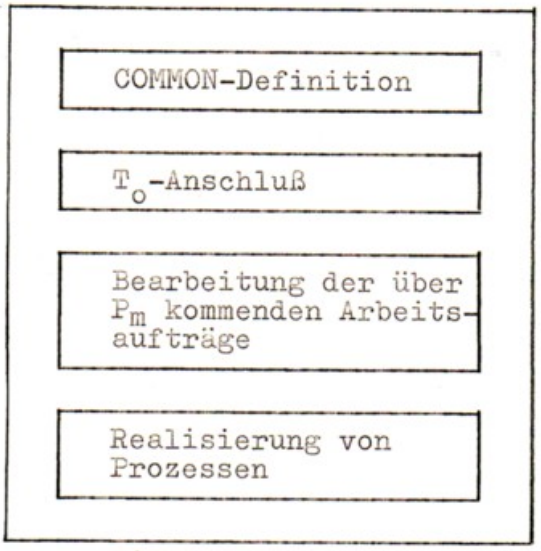


Fig. 5: Formaler Aufbau einer Arbeitstask T_n.

T_o - A n s c h l u ß :

Besonders mühsam und zeitaufwendig wäre es, wenn der T_o-Anschluß in jeder Task neu formuliert werden müßte. Es steht daher ein Macro WORKTASK zur Verfügung, das diesen Anschluß realisiert und darüber hinaus auf Wunsch auch noch Routinen generiert, die durch Macro-Aufrufe anzusprechen sind. Das Macro WORKTASK wird in der folgenden Weise aufgerufen:

WORKTASK(TN,N,INST,PROC,KIND,M1,M2,M3, ...)

TN Sprungadresse zum Beenden der Task T_n.

N Tasknummer

INST Sprungadresse zur Abarbeitung des Puffers P_m (3. Segment in Fig. 5). Bei impliziten Tasks wird dabei noch die Befehlsnummer übergeben.

PROC =NONE T_n wickelt für T_o keinen Prozeß ab.
≠NONE Name eines Unterprogramms, in dem Prozeß verwirklicht ist.

KIND =IMPL T_n ist eine implizite Arbeitstask.
≠IMPL T_n ist eine explizite Arbeitstask und KIND ist eine Sprungadresse, die zu dem Programmteil von T_n führt, in dem der Einschaltbefehl bestätigt wird.

M_i Namen von Macros, die in T_n aufgerufen werden können. Die dabei angesprungenen Unterprogramme werden generiert, je nachdem ob der betreffende Macroname genannt ist oder nicht.

M a c r o - K e r n :

Mit Aufruf von WORKTASK steht (unabhängig von den M_i) generell ein Satz von Macros zur Verfügung, die den sogenannten Macro-Kern von WORKTASK bilden. Er kann beliebig erweitert werden und besteht zur Zeit aus den folgenden Macros:

WRITREAD(CURSER,WRITLGTH,ANSWER)

Am Ende der Abarbeitung von P_m durch T_n steht dem Teilnehmer an D_m eine Antwort zu. Durch Aufruf des Macros WRITREAD kann T_o veranlaßt werden, die ersten WRITLGTH Zeichen von P_m auf D_m zu schreiben und weitere Vorbereitungen zu treffen, damit von D_m erneut eine Zeichenkette abgeschickt werden kann. Die Anfangsadresse ANSWER eines Strings, der in das Antwortfeld von P_m übertragen wird, kann dabei ebenso genannt werden wie der Parameter CURSER, der die Lage des Bildschirmzeigers beeinflußt. Durch diese Möglichkeit kann man dem Teilnehmer Positionierarbeit abnehmen (zum Beispiel beim blättern auf Dateien).

GETCHR

Dieser Macroaufruf bewirkt, daß das nächste Zeichen des Puffers P_m in ein Register (Q-Register) gelangt. Der Eingabezeiger wird dabei um eine Stelle verschoben.

PUTCHR

Der Inhalt des Q-Registers wird auf die nächste Zeichenposition von P_m gebracht. Der Ausgabezeiger erhöht sich dabei um 1.

M a c r o - R a h m e n :

Die Gesamtheit aller Macros, deren Namen als Parameter im WORKTASK-Macro genannt werden können, bilden den Macro-Rahmen von WORKTASK. Auch er kann beliebig erweitert werden. Jedes dieser Macros ruft ein Unterprogramm auf, das von WORKTASK erzeugt wird, wenn der Macroname als aktueller Parameter in der Parameterliste auftaucht. Damit wird erreicht, daß das Macro nur solche Unterprogramme erzeugt, die auch tatsächlich in der betreffenden Arbeitstask angesprungen werden.

Die folgenden Macros bilden zur Zeit den Macro-Rahmen von WORKTASK:

GETPOS(POS,KIND) M1

Durch Aufruf dieses Macros kann man sich die Position des Eingabezeigers (KIND=I) bzw. Ausgabezeigers (KIND=O) in einem Wort mit der Adresse POS besorgen. Die Zeiger können von 1 bis zur Zeichenlänge von P_m laufen. Bei impliziten Tasks verweist der Eingabezeiger anfangs auf das erste Zeichen des Befehlsrumpfs.

SETPOS(POS,KIND) M2

Hiermit läßt sich die Position des Eingabezeigers (KIND=I) bzw. Ausgabezeigers (KIND=O) beeinflussen. Die Zeiger werden auf den Wert des Wortes mit der Adresse POS gesetzt.

GETSTG(ADDR,LGTH,DLCHR1,DLCHR2) M3

Mit dem Aufruf dieses Macros werden von der Position des P_m-Eingabezeigers an höchstens LGTH viele Zeichen von P_m nach ADDR übertragen. Die Begrenzungscharakter DLCHR1 und DLCHR2 brechen die Übertragung vorzeitig ab und der Rest des Zielfeldes wird mit blanks aufgefüllt. Die Anzahl der übertragenen Zeichen steht danach im Indexregister B1 und für B1=LGTH wird das A-Register Null gesetzt. GETSTG ruft GETCHR und verändert damit den P_m-Eingabezeiger.

PUTSTG(ADDR,LGTH,DLCHR1,DLCHR2) M4

PUTSTG wirkt exakt umgekehrt wie GETSTG und verändert den P_m-Ausgabezeiger, da es PUTCHR aufruft.

GETINT(INT,DLCHR1,DLCHR2) M5

Mit dem Aufruf von GETINT wird von der Position des P_m Eingabezeigers an die nächste ganze Zahl zeichenweise eingelesen und binär dargestellt in das Wort mit der Adresse INT gebracht. Der Vorgang bricht ab, wenn die Charakter DLCHR1 oder DLCHR2 angetroffen werden. GETINT verändert den Eingabezeiger.

PUTINT(INT,LGTH) M6

Wandelt eine in dem Wort mit der Adresse INT binär dargestellte Zahl um in ihre dezimale Darstellung. Diese gelangt nach P_m an die Position, wo der P_m-Ausgabezeiger

vor Aufruf des Macros stand. PUTINT verändert den P_m-Ausgabezeiger.

Beispiele für die Verwendung der Macros:

Entsclüsseln eines Befehlsrumpfes mit einem alphanumerischen und einem numerischen Parameter in einer impliziten Task T_n:

Der P_m-Eingabezeiger verweist nach Aktivierung von T_n durch T_o bereits auf das erste Zeichen des Befehlsrumpfes AP,NP , sodaß durch

```
        GETSTG   (APADDR,8,COMMA,BLANK)
        AZJ,EQ   APTOLONG
        ISG      1,1
        UJP      APMIS
        GETINT   (NPADDR,COMMA,BLANK)
        AZJ,EQ   OVERFLOW
        AZJ,LT   SYNTERR
        LDA      DEFAULT
        ISG      1,1
        STA      NPADDR
    weitere Befehlsabwicklung
COMMA   EQU      73B
BLANK   EQU      60B
APADDR  BCD,C    8,
NPADDR  BSS      1
APTOLONG WRITREAD (2,TWOLINES,LONGAP)
LONGAP  BCD,C    20,AP-PARAM. TOO LONG
APMIS   ..........................
OVERFLOW ..........................
..................................
```

die beiden Parameter AP und NP in handlicher Form in APADDR und NPADDR zur weiteren Befehlsabarbeitung vorliegen.

Abändern eines numerischen Parameters NP innerhalb eines Befehlsrumpfes ANFANG,NP,ENDE , der von einer impliziten Arbeitstask entschlüsselt wird:

```
ANFANG entschlüsseln

    GETPOS    (POS1,I)
    GETINT    (NPADDR,COMMA,BLANK)
    AZJ,EQ    OVERFLOW
    AZJ,LT    NPSYNTAX
    ISG       1,1
    UJP       NPMIS
    GETPOS    (POS2,I)

ENDE entschlüsseln und Befehl ausführen;
jetzt NP um STEPSIZE erhöhen

    SETPOS    (POS2,I)
    GETSTG    (STRING,TWOLINES,BLANK)
    SETPOS    (POS1,0)
    ENA       STEPSIZE
    RAD       NPADDR
    PUTINT    (NPADDR)
    ENQ       COMMA
    PUTCHR
    PUTSTG    (STRING,TWOLINES,BLANK)
    WRITREAD  (3,SCRNSIZE,FINISHED)
FFINISHED BCD,C    20,FINISHED, GO ON
 NPADDR   BSS      1
 STRING   ..........................
 ..................................
```

A r b e i t s t a s k s i n h ö h e r e n P r o g r a m m i e r s p r a c h e n :

Um Arbeitstasks in höheren Programmiersprachen zu erstellen, muß grundsätzlich von der betreffenden Sprache aus die Möglichkeit vorhanden sein, den Macro-Aufruf WORKTASK zu realisieren, um die Verbindung zur Grundtask T_o herzustellen. Das könnte für FORTRAN oder ALGOL durch einen einmaligen SUBROUTINE- oder PROCEDURE-Aufruf geschehen, wodurch ein Assembler-Unterprogramm mit geeigneten Parametern aktiviert wird, in dem WORKTASK Anwendung findet. Die P_m erkennenden und aufbereitenden Basisroutinen müssen auf gleiche Weise der betreffenden Sprache zugänglich gemacht werden. Die höheren Routinen könnten dann in der betreffenden Sprache selbst formuliert werden.

Inhalt:

DAS TEILNEHMERSYSTEM M O T U S

M O T U S

Ein auf einfache Weise erweiterbares Teilnehmersystem
mit der Möglichkeit eines vollständigen Remote-Job-Entry

Entwickelt und
implementiert von:

Dieter Wolff

Zentrum für Datenverarbeitung
der Justus Liebig-Universität

63 Gießen

Mai 1975