To: Distribution

From: Melanie Weaver

Date: 12/21/78

Subject: Transaction Processing, first release

This MTB describes the transaction processing facility that is planned for Multics release 7.0. However, TP will probably not be ready when the release is shipped because it is dependent on some features of vfile_ whose implementation has been delayed. The bulk of the MTB is a draft version of the TP manual. In addition, there is a brief discussion of some features not to be included in this release, two include files describing internal tables for those who are interested at that level, and a description of a design point that still needs to be resolved.

Comments and questions should be sent to:
Melanie Weaver
Honeywell / CISL
575 Technology Sq.
Cambridge, MA 02139

617-492-9312 or HVN 261-9312

or

Weaver.Multics on MIT or Phoenix

JECT:

Description of the Multics Transaction Processing Subsystem for Performing Data Base Related Operations

ECIAL INSTRUCTIONS:

This manual is a preliminary edition which describes a basic transaction processing capability for use on a Multics system.

TWARE SUPPORTED:

Multics Software Release 7.0

E:

January 1979

ER NUMBER:

CC96, Rev. 0

PREFACE

Transaction processing in the Multics system is performed by the subsystem described in this manual. By employing the transaction processing (TP) subsystem, the individual user can process transactions against a data base of any size by invoking a set of readily understood and easily applied commands defined by the site. This manual describes the Multics TP subsystem, describes the administrative commands and their usage, and furnishes the practical details of subsystem operation.

Although many TP users will not need to avail themselves of the full power of the Multics system, additional information regarding Multics software concepts and organization as well as specific usage of Multics commands and subroutines can be obtained from the volumes of the _Multics Programmers' Manual_ (MPM). These volumes are:

MPM Reference Guide, Order No. AG91

MPM Commands and Active Functions, Order No. AG92

MPM Subroutines, Order No. AG93

MPM Subsystem Writers' Guide, Order No. AK92

MPM Peripheral Input/Output, Order No. AX49

SECTION 1

INTRODUCTION

MULTICS TRANSACTION PROCESSING

The Multics transaction processing (TP) subsystem provides an environment for users who want to perform a well-defined set of data base related operations without taking advantage of the full range of Multics' time-sharing capabilities. In this manual, a user is defined to be a person who enters transactions. Thus an application programmer (TPR writer), while a Multics user, is not a TP user. Many aspects of the subsystem are table-driven, and modules can be modified, so that features can easily be added, changed or deleted. Several independent TP subsystems may run concurrently on a Multics system.

This manual describes how the Multics TP subsystem is organized, how to run it, how to meter it, and how to write application programs for it. However, both administrators and application programmers still need to consult the MPM, as well as other appropriate manuals. Specific TP user requests are site-dependent and so are not included.

LIST OF FEATURES

Features available in the system include the following:

- locking of individual data base records to allow concurrent data base access

- checkpoint and rollback facilities which allow handling of both concurrency control and restoration after an error or other interruption

- availability of most Multics languages for application programs

- modular construction to facilitate tailoring of the system

- separation of terminal and data processing to allow better tuning

- a priority mechanism for commands
- easily established parallel environments for testing application programs

OVERVIEW OF STRUCTURE

For the purpose of discussion within this manual, a transaction is defined as the processing of a single user command, from receipt of the input message by the TP subsystem to the completion of execution. A transaction processing routine (TPR) is an application program, usually one that references a large data base.

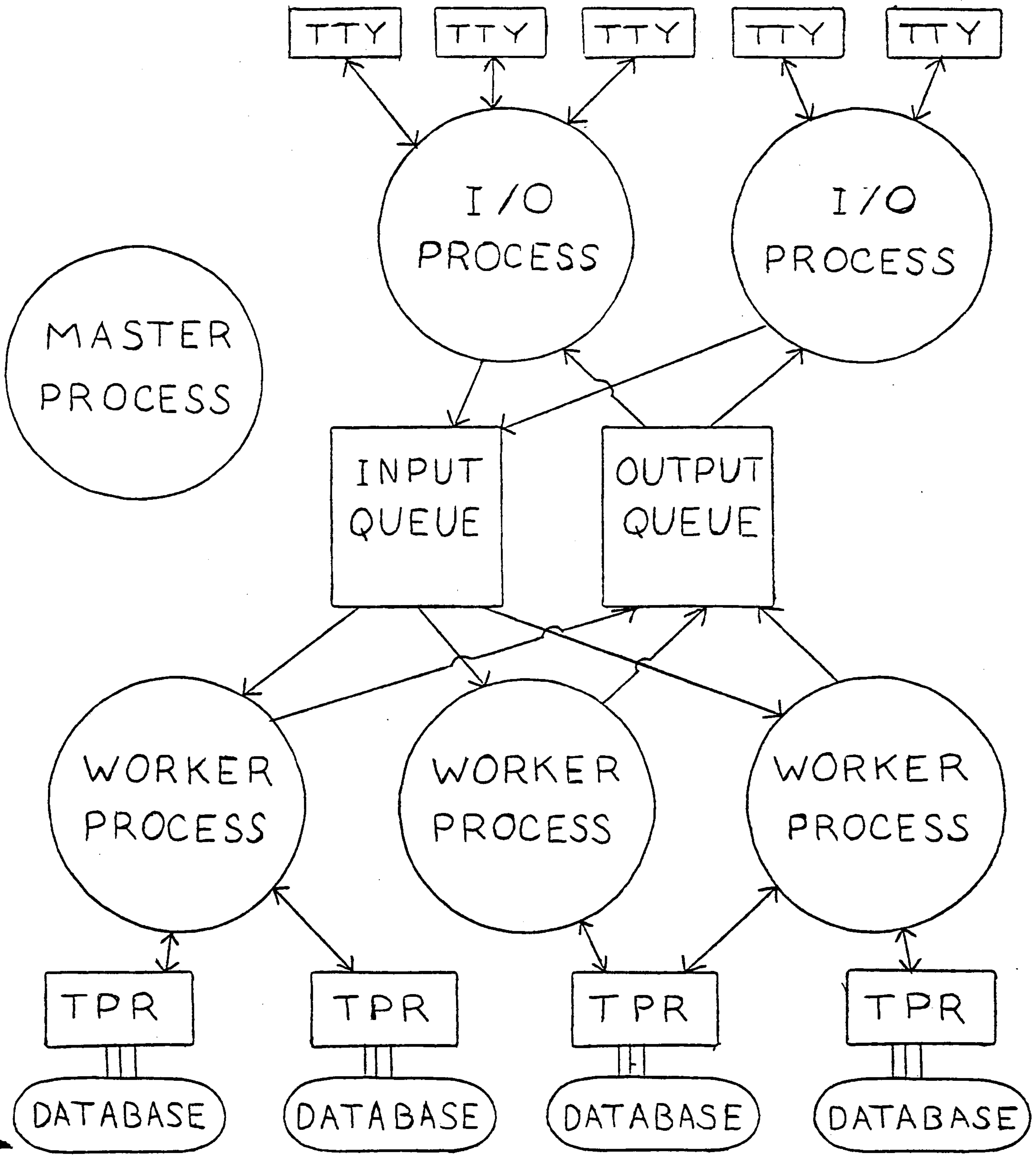
The Multics TP subsystem is organized to make efficient use of resources by taking advantage of the special characteristics of TP. Typically, many users are all performing similar operations to a fixed set of data bases, using a closed environment of their own rather than the standard Multics command interface. Allocating a process per user would entail much duplicated address space overhead and would be difficult to schedule efficiently, so a different process structure is used. Regular Multics processes are used, but they are divided into three types, according to function. The first type, called an I/O process, manages all input/output from/to terminals. The second type, called a worker process, performs the actual application execution. The third type, called the master process, of which there is only one, is used by the TP administrator to coordinate and communicate with the others. The I/O and worker processes communicate with each other via input (commands to be processed) and output (from TPRs) queues as well as through a global table. Figure 1-1 shows a simplified diagram of the relationships of the various processes. These processes are described in more detail below.

The master process begins a TP session by initializing the master table, which contains information about all the processes. Any data base modifications that were incomplete because of the crash are automatically deleted as they are encountered in normal processing. Then the master process activates all the other processes. The master process is responsible for coordinating TP shutdown, ensuring that all transactions in progress are completed.

An I/O process handles input/output for a group of terminals. The lines handled may be dedicated or attached to the I/O process by the dial facility. The process also queues and schedules the input messages. Some commands, particularly those that require little processing and do not involve data base references, may actually be executed by the I/O process.

A worker process executes one transaction at a time, queuing output messages that are printed on the user's terminal as soon as they can be scheduled. Databases are referenced through the virtual file manager, vfile_, in a mode that separates before and after images so that changes can be reversed if a transaction is interrupted because of an error or system crash. Also, if a transaction fails because of a concurrent update to the data base by another process, the changes are rolled back and the transaction is re-executed. The process' executive program establishes a condition handler that in some cases handles the condition and restarts the transaction but in most cases informs the user, terminates the transaction, and rolls back the data base changes. All databases to be referenced by the process are attached and opened at the beginning of the process, so that overhead is eliminated from individual transactions.

STRUCTURE OF MULTICS TP MONITOR



SECURITY FEATURES

Security

Each I/O and worker process may have its own process group ID, so that Multics access control may be used to restrict access to certain data bases to specific workers. Individual users, however, do not have a Multics process group ID while using the TP system; in fact, they need not be registered Multics users at all. They only need to be registered as TP users. Because the set of commands they can invoke is so restricted, i.e., generally not including editors or compilers, etc., the individual TP users should not be able to perform malicious acts such as destroying subsystem data bases.

Terminal I/O -- User's View

A user may wait for the complete output corresponding to a transaction before typing the next transaction. However, the TP subsystem does not enforce serial execution. If the user enters requests without waiting for previous ones to complete, the following types of behavior may occur. The output from an immediate command usually appears before output from any other pending transaction. This is because immediate commands are executed as soon as they are read rather than queued for a worker process. (See the description of the command table source language in Section 4 for more information about immediate commands.) The output from a regular transaction might not be contiguous and might appear before the output from an earlier transaction. Aside from the fact that some commands may have higher priority than others, this effect can occur because the TP subsystem is capable of executing more than one transaction from a single user simultaneously. Although execution of transactions is begun in order first of priority and then of the time entered, a later transaction may finish before an earlier transaction. In addition, the output messages are queued for processing as soon as they are generated. An output message is defined as the result of a single output statement.

Since output messages from one transaction may be interspersed with output messages from other transactions, each output message is labelled with the transaction ID. The ID is first printed when a transaction is queued.

SECTION 2

RUNNING AND OPERATION

BACKGROUND INFORMATION

A TP subsystem is defined by a directory and the control segments contained in it. This section briefly lists and describes the control segments.

The TP master table, tp_master_table_, contains most of the dynamic runtime information used by the TP processes including output queue controls, information about each terminal, and an entry for each process containing interprocess communication information, etc.

The command table, tp_command_table_, is a list of the valid command names for the TP subsystem and their associated attributes. This is described in detail in Section 4, "Tables".

The database definition exec_com, tp_init_data_base.ec, contains commands to open data bases. There is a definition for each way each data base might be opened.

The operator table, tp_operator_table_, is a list of names of valid terminal users and their enciphered passwords.

The start_tp exec_com initializes a TP session and starts up the I/O and worker processes via Multics enter_abs_request commands.

A worker process startup exec_com invokes the data base definition exec_com for each data base to be used by the worker and then turns the process into a worker.

An io process startup exec_com turns the process into an I/O process, passing it the tty channel names.

The TP control file, tp.tcf, contains information indicating whether data base operations have been checkpointed.

The TP input queue, tpin_, contains, for each nonimmediate transaction, its input command line, meters, and information about its output destination and current state.

The TP output queue, tpout_, contains the transaction output messages to be displayed on the terminals.

HOW_TO_SET_UP_A_TP_SYSTEM

The master, I/O and worker processes must be registered by the project administrator. If a process group ID is given the multip attribute, many processes with that ID may be logged in concurrently. However, a worker process should not have the same ID as an I/O process, and worker processes with different absin files, i.e., that use different data bases, must have different IDs. Each I/O and worker process is given a name to identify it within TP that is independent of the process group ID. The master process may not have the same ID as either an I/O or worker process, but it may be the TP administrator's personal process.

Setting up a TP system primarily involves creating a special directory with the appropriate control segments. The control segments need not be created in any special order.

The command table must be produced and installed. All the TPR's and immediate commands listed must exist and the worker processes must have re access to them, although they need not reside in the TP directory. All user documentation about the available commands must be prepared at the site, since it applies only to a specific TP subsystem.

The operator table must be produced using the program tp_user$add and the I/O processes given r access to it.

The start_tp exec_com and the data base definition exec_com must be produced; see below for details.

Absin files must be created for each worker and I/O process. An absin file can be used by more than one process, but there must at least be a separate name for each process to avoid interference among the absout files. These are discussed in more detail below. All databases that are to be keyed sequential files attached through vfile_ must exist as multi-segment files before being used under TP. If a non-MDBM database is not already initialized, the TP administrator can create it with the following command line:

```
          ec  tp_pre_create  file_name
```

A file must be accessible to each worker process that will reference it. In the release described by this manual, all worker absin files must be the same, since any transaction can be executed by any worker.

The master table and TP control file are created by the TP system.

The input and output queues must be created by the TP administrator before the TP system is brought up for the first time by using tp_pre_create.ec.

Initializing the TP Session

When the TP directory and necessary files exist, and when the processes have been registered, the TP system may be initialized. This is done through the master process, which controls the session. To begin with, the master process is just an ordinary Multics process (or daemon) with the master process group ID. Executing the start_tp command causes it to become a TP master process and initializes some system data bases. The I/O and worker processes can be logged in either by enter_abs_request commands from the master process or as daemons from the Multics initializer. In either case, the ear or daemon login commands are normally in an exec_com, since the number at startup time is usually constant and there are several of them. The processes may be brought up in any order.

If absentee is used, the master process must have e access on the proxy ACS. The ear commands have the following calling sequence:

```
     ear absin_name -foreground -proxy Person_id.Project_id -ag
     proc_name
          tp_dirname
```

where:

proc_name
          is the symbolic name of the I/O or worker process

tp_dirname
is the pathname of the TP directory

Initializing a Worker Process

A worker process is initialized by its absin file if it is absentee or by an exec_com if it is a daemon. This absin file/exec_com attaches the transaction control file, opens and readies all the data bases the worker will use so that TPR's do not have the overhead of opening and closing files, and invokes tp_run_worker, which actually turns the process into a worker. Arguments to the exec_com are: a name to identify the worker process and the pathname of the TP directory. The format is given below.

```
change_wdir  &2
tp_init_tcf &2
ec tp_init_data_base file_id1 submodel_path1
                                  /* open MRDS data bases */
     .
     .
     .
ec tp_init_data_base file_idn submodel_pathn
ec tp_init_data_base file_idi file_pathnamei switch_namei
                                         /* vfile_ data bases */
     .
     .
     .
tp_run_worker  &1  &2
```

See the discussion in Section 3 on data base definition for a description of tp_init_data_base.ec. See the command descriptions in Section 3 of this manual.

Initializing an IO Process

A process becomes an IO process by executing a tp_start_io command, specifying the devices to be managed by the process. This command is in the absin file if the process is absentee and in an exec_com if the process is a daemon. In either case, the format of the file is the same. Arguments to the file are: a name to identify the I/O process and the pathname of the TP directory. The format is given below:

```
change_wdir  &2
tp_start_io  &1  &2  channel_name1 ...
```

See the command description for tp_start_io.

Shutting Down

A normal shutdown is performed by issuing the command stop_tp in the master process; the rest is done automatically in two stages. In the first stage, the I/O processes stop accepting input and the worker processes either finish the current transaction and log out (if -immediate was specified) or finish all pending transactions before logging out (default). In the second stage, the I/O processes finish processing all accumulated output and log out. Thus after a normal shutdown, all the output for completed transactions has been processed. If -immediate was specified, there will probably be some transactions left in the queue to be processed during the next session.

DIAL AND TERMINAL USAGE

Overview of TP Terminal Management

In Multics TP, each I/O process manages a group of terminals. It is not possible for a particular terminal to be used by more than one process at a time. An I/O process handles all the input and output for its terminals, including operator sign-ons. Terminals are connected to an I/O process either through the dial facility or through specified slave channels. Since the I/O process controls the terminals, it also controls the users' access to the TP system; a TP user need not be a registered Multics user.

Use of the Dial Facility

The Multics dial facility allows several terminals to be attached to the same process. The process first arranges with the answering service (which usually handles dialups) to accept dialups when a particular dial ID is specified. Then, instead of using the login command, which results in a separate process, but only for a registered user, a person types dial and then the dial ID. In addition, if the dial ID is not registered, the person must type the Person_id.Project_id of the corresponding I/O process. See the MAM, Communications for instructions on registering dial IDs. In order for an I/O process to accept dials, the tp_start_io command must be given the control argument -dial dial_id or -registered_dial dial_id. See the description for tp_start_io in this manual.

The dial facility should be enabled when there are nonslave channels that are to use the TP system, i.e. when some lines need to be able to access both the TP system and the regular Multics system.

Use of Assigned Terminals

An I/O process may manage channels that are specified when the command tp_start_io is invoked. These channels must be slave channels, i.e., they must be used by some process other than the answering service and cannot be used to login to Multics. See the MAM, Communications for a description of the channel table, which describes each channel, and for instructions on turning channels into slaves.

RUNNING TP IN TEST ENVIRONMENTS

Setting Up A Parallel Test Environment

Parallel TP systems can be established for testing. Setting one up requires creating a directory and the necessary TP control segments, just as for a real TP system. This includes using pre_create.ec for the input and output queues. If data bases are used in common with a TP system, the TP system's transaction control file must also be used, via a link in the test directory.

Test Mode

A user may use the TP system in test mode, which will process transactions and produce output, but will not actually change any data bases. To enter this mode, type the command "enter_test_mode". To return to normal TP processing, type "exit_test_mode".

Establishing Test Processes

Once a test TP environment has been set up, it is necessary to create test TP processes. One way to do this is to establish, within a project, special person IDs to be used in the test environment. These can be logged in as foreground absentee jobs, daemons or interactively. Actually, any processes with access to the test directory can be used; a process becomes a TP process by executing one of the commands tp_start_io or tp_run_worker. The important point is that start_tp must be executed first, and that the names given to the worker processes correspond to entries in the worker table. Any worker process that is originally an interactive process, rather than absentee or a daemon, will tie up a terminal that it won't use. If an I/O process uses the dial facility, its person.project ID as well as its dial ID must be known to all potential users.

If the queues are to be cleared before testing begins, the existing ones must be deleted and new ones created using pre_create.ec before typing start_tp.

It is possible for one process to handle different functions, but then there are restrictions. If a single process is used for both master and I/O functions, the command table must contain any desired master commands as immediate commands. The master and a worker process are not particularly compatible, but may be used together if one is willing to forego using any master commands, since master process output would not work and the worker process ignores terminal input. If all three types share a process, the TP commands must be in an exec_com and tp_start_io must be executed before tp_run_worker on the same command line.

SECTION 3

TP COMMANDS, SUBROUTINES, AND ACTIVE FUNCIONS

The commands, subroutines, and active functions available to the user for transaction processing are described in this section. They are listed in alphabetical order within this section, and are presented in a format consistent with that described in the *Multics Programmers' Manual* (MPM).

Name: start_tp

The start_tp command starts a TP session by initializing the TP master table. It must be invoked in the process that is to become the master process.

Usage

```
     start_tp  tp_dir_name
```

where tp_dir_name is an argument that is required and is the name of the directory that contains the control segments for the TP subsystem to be initialized.

Notes

The subsystem data bases, transaction control file and input queue must all be consistent, both internally and with respect to each other, before this command is invoked.

This command does not start up other TP processes.

Name: stop_tp

The stop_tp master process command shuts down a TP session. The sequence of operation is to first tell the I/O processes to stop accepting input, then to tell the worker processes to log out either when they have finished their current transactions or when they run out of work, and then to tell the I/O processes to log out after all output has been printed.

Usage

```
      stop_tp {-control_arg}
```

where control_arg is -immediate, -im to specify the worker processes should log out immediately after finishing their current transactions. The default is for the worker process to log out when there are no more transactions to execute.

Name: tp_cancel

The tp_cancel command removes a transaction request from the TP input queue to prevent it from being processed. It can be used either in the master process or in an I/O process, but if used in the latter, it must be included in the command table and the user can cancel only those requests that he/she entered.

Usage

```
     tp_cancel transaction_num1 ... transaction_numn
```

where transaction_numi is the identifying number of the transaction to be cancelled.

Name: tp_change_priority

The tp_change_priority command changes the priority of the specified transaction. It may be used only in the master process.

Usage

```
tp_change_priority transaction_num new_priority
```

where:

1. transaction_num
   is the identifying number of the transaction whose priority is to be changed.

2. new_priority
   is the new priority of the transaction.

Name: tp_cvsct

The tp_cvcst command is the TP command table compiler, converting a command table in source language to a binary form usable by the TP subsystem.

Usage

```
     tp_cvsct source_name
```

where source_name is the pathname of the command table source segment. No suffix is assumed.

Notes

The binary command table is created in the working directory; its entryname is the same as that of the source segment with the addition of the suffix .tpbct.

Name: tp_display_current_xcns, tp_dcx

The tp_display_current_xcns command displays information about the transactions currently being executed, viz., worker process name, transaction number, TPR name and user name. This command may only be used in the master process.

Usage

```
     tp_display_current_xcns
```

Name: to_get_xcn_status

The to_get_xcn_status command returns information about a specified transaction. If the transaction has not yet been processed, its position in the queue and the time it was submitted will be displayed. If it is currently being processed, the time it was submitted, the time it was started and the command name are displayed. If it has finished, except possibly for output, the information printed includes the time it was finished, whether there were any errors, the command name, the time submitted, real and cpu time used, page faults, and total real time (not including output processing).

Usage

```
     to_get_xcn_status num1 {num2 ...} {-brief}
```

where:

1. numi
   is the number of the desired transaction.

2. -brief, -bf
   causes only the state of the transaction to be printed.

Notes

When this command is used in the master process, any transaction may be specified. When used in an I/O process, information is returned only about a user's own transactions.

Name: tp_init_tcf

The tp_init_tcf command attaches and opens the TP subsystem's transaction control file (TCF). It must be called in a worker process before any databases are opened, because the TCF switch must be specified at attach time for any file that is to be used in transaction mode (i.e. covered by the checkpoint mechanism). The TCF's name must be tp.tcf and the I/O switch used is tp.tcf_.

Usage

```
     tp_init_tcf
```

Name: tp_list_pending_requests, tp_lpr

The tp_list_pending_requests command lists the transactions that have not yet been executed. It can be invoked either in the master process, where it will list all requests, or in an I/O process, where it will list only the user's own requests.

Usage

```
tp_list_pending_requests {-control_args}
```

where control_args may be chosen from the following list:

-totals, -tt
: prints only the number of transactions at each priority.

-long, -lg
: prints the time submitted, the user name and the user's channel ID in addition to the transaction number.

-priority n
: prints information only about transactions at the specified priority.

Name: tp_meters

The tp_meters command displays in the master process metering information derived from a TP input queue. This includes:

- the total cpu time and page faults for TPR execution
- the number of successful completions
- the number of errors
- the number of checkpoint failures
- the largest, smallest and average number of submissions per hour both total and per I/O process
- the largest, smallest and average time between submission and processing
- the largest, smallest and average time spent in execution, both total and per worker

  The last category includes all transactions for which any execution took place during the specified time period.

Usage

```
tp_meters -from time1 -to time2 {q_name}
```

where:

1. -from time1
   specifies the beginning of the time period; time1 is a string that is passed to convert_date_to_binary_.

2. -to time2
   specifies the end of the time period; time2 is also passed to convert_date_to_binary_.

3. q_name
   is the name of the TP input queue to be metered. If omitted, the current queue is used.

Name: tp_pre_create.ec

The tp_pre_create exec_com creates an empty indexed file as a multisegment file. It should be used before the first reference to an indexed file if the file does not exist or has been truncated. In particular, this should be used on the TP input queue.

Usage

```
     ec tp_pre_create file_name
```

where file name is the name of the file to be initialized.

Name: tp_reset_xcn_num

The tp_reset_xcn_num command changes the current transaction number (one less than the number to be assigned to the next transaction request) to a specified value. If the value is to be decreased, the input queue must be empty (see the descriptions of the stop_tp and tp_shrink_q commands).

Usage

```
     tp_reset_xcn_num {value}
```

where value is the new current transaction number. The default is zero.

Name: tp_run_worker

The tp_run_worker command turns a process into a TP worker process and then stays around as the worker executive, processing transactions.

Usage

```
tp_run_worker worker_name tp_dir_name
```

where:

1. worker_name
   is the name of the worker process as used within the TP subsystem.

2. tp_dir_name
   is the name of the directory containing the control segments for the TP subsystem.

Name: tp_shrink_q

The tp_shrink_q command removes from the TP input queue records concerning transactions that were processed before a specified time. The records may be either deleted or copied onto tapes or into a sequential file. The command should be used outside a TP subsystem.

Usage

```
tp_shrink_q q_name {-control_args}
```

where:

1. q_name
   is the pathname of the TP input queue from which records are to be removed.

2. control_args
   may be chosen from the following list:

   -time t, -tm t
   removes records of only those transactions that completed before time t. The default is the current time.

   -delete, -dl
   deletes records without copying them.

   -tape
   copies the records onto tape; the user of the command is asked for the tape numbers. This is the default.

   -file filename
   copies the records into the specified sequential file.

   -all, -a
   removes all eligible records. This is the default.

   -successful
   removes only the records concerning transactions that completed successfully.

   -error
   removes only the records concerning transactions that aborted because of an error.

-checkpoint_failure
    removes only the records concerning transactions that could not be checkpointed.

Notes

The control arguments -delete, -tape and -file are mutually exclusive.

This command can be executed while the TP subsystem is running. It can also be used in several processes simultaneously, for example, to put records of successful transactions on tape and records of checkpoint failure transactions into a file.

This command should not be used exclusively to keep the input queue trimmed. Occasionally the input queue should be completely cleared out so that transaction numbers can be reused.

Name: tp_start_io

The tp_start_io command turns a process into a TP I/O process, adding an entry in the master table. It attaches any specified channels and can set itself up as a dial responder. Its purpose is to initialize support for each channel, not to do actual input or output.

Usage

```
     tp_start_io io_proc_name tp_dir_name {channel1 ... channeln}
          {-control_args}
```

where:

1. io_proc_name
   is the name of the I/O process as used within the TP subsystem.

2. tp_dir_name
   is the name of the directory containing the control segments for the TP subsystem.

3. channeli
   is the name of a slave channel. The name must be defined in the system channel definition table.

4. control_args
   may be chosen from the following list:

   -dial dial_name
   establishes a dial responder for users who use the dial facility with the dial_name.

   -registered_dial dial_name
   similar to -dial but allows users to omit specifying the process group ID when using the dial facility.

   -switch_tty_to_tp
   when tp_start_io is invoked from an interactive process, this control argument switches the handling of terminal I/O from Multics command level to the TP subsystem. Useful for debugging.

Name: tp_user

The tp_user command contains interfaces for managing the TP person name table.

Entry: tp_user$add

The add entrypoint adds users and their passwords to the TP person name table. These user names are not related to any user names registered on the general Multics system.

Usage

```
tp_user$add pnt_name {control_args}
```

where:

1. pnt_name
   is the pathname of the TP subsystem's person name table.

2. control_args
   may be chosen from the following list:

   -file_input file_name, -fi file_name
   specifies that the input is to come from the segment whose pathname is file_name; in this case, each line consists of a single user name and, optionally, a password. If the password is omitted, the caller is prompted for it. The default is to prompt the caller for each user name. The password may be given on the same line; but if it is not, the caller is prompted for it also. To exit from the command, type "write" instead of a user name; this performs the actual update. On the next line, type "quit". If quit is given without write, the caller is questioned.

   -brief, -bf
   specifies that the introductory message about how to use the command is not to be printed. It is ignored in file_input mode.

Entry: tp_user$change

The change entrypoint changes a person's password.

Usage

    tp_user$change pnt_name {control_args}

where the arguments are the same as in the add entrypoint.

Entry: tp_user$delete

The delete entrypoint deregisters users from the TP subsystem.

Usage

    tp_user$delete pnt_name {control_args}

where the arguments are the same as in the add entrypoint except that no passwords are involved.

Name: tp_who

The tp_who command prints the names of the current users of the specified TP subsystem. It may be used in either the master process or an I/O process.

Usage

```
tp_who {Person_id1 ... Person_idn} {-control_args}
```

where:

1. Person_idi
   is the name of a user as registered in the TP subsystem.

2. control_args
   may be chosen from the following list:

   -long
   prints the channel IDs as well as the names of users. The default is just to print the names.

   -io_proc proc_name
   prints information only about users connected to the specified I/O process. proc_name is the name of the I/O process as used within the TP subsystem.

Name: transaction_call, trc

The transaction_call command performs, controls, or obtains status information for an atomic data base operation.

Usage

```
trc op_key tcf_sw {-brief, -bf} {args}
```

where:

1. op_key designates one of the following operations:

   transact, t
   executes a given command line as a transaction.

   checkpoint, c
   attempts to complete the current transaction and reset the transaction number to 0 if successful.

   rollback, r
   undoes all modifications made on behalf of the current transaction and resets the current transaction number to 0.

   assign, a
   reserves a unique transaction number for the current transaction by creating a new entry in the control file (tcf).

   status, s
   prints information about a transaction, including its number, completion status, and optionally also shows the counts of references and/or verifies them.

   number, n
   prints and optionally resets the current transaction number without altering the tcf.

2. tcf_sw
   names an I/O switch attached to the transaction control file (tcf).

3. -brief, -bf
   optionally suppresses the default message printed by this command. If status was requested, this suppresses the counting of references made by the transaction.

4. args
   vary as follows, depending on the choice of op_key:

```
transact
     command_line
checkpoint
     (no additional args)
rollback
     (no additional args)
assign
     {transaction_number}
status
     {-verify,-vf} {transaction_number}
number
     {transaction_number}
```

where:

1. command_line
   is a Multics command line that need not be enclosed in quotes unless it contains special characters.

2. transaction_number
   is a nonnegative integer that uniquely identifies a single transaction. If the assign op_key is given, then this is the new transaction number; if omitted, the next available number is automatically supplied. If the status op_key is given, then this is the number of the transaction whose status is desired; otherwise, if omitted, the current transaction is assumed. If the number op_key is given, then this is the new transaction number; if omitted, then the current number is printed and no changes are made.

3. -verify, -vf
   causes a check of all passive references made in the transaction for possible asynchronous changes. If a previously referenced item has been changed, an error message is printed, indicating that this transaction will be unsuccessful.

Notes:

A transaction number is automatically assigned upon the first reference to a data base item within a new transaction, if no transaction number has been explicitly set via the "assign" or "number" op_key. The tcf switch must be open for modification in all cases, except when one uses the "number" op_key, and if the transaction is entirely passive (i.e. does not alter the data base).

See the description of the transaction_call_ subroutine for more detailed notes.

<u>Name</u>: transaction_call_

The transaction_call_ subroutine executes a given command line as an atomic transaction on a specified data base. Handlers are established for the cleanup and program_interrupt conditions. The cleanup handler causes the transaction to be rolled back if, for example, the user quits and releases. The program_interrupt handler permits one to rollback and reexecute the command line by typing pi from command level.

<u>Usage</u>

```
     dcl transaction_call_ entry  (ptr, fixed  bin(35),  char(*),
          fixed bin(35));

     call transaction_call_ (tcf_ptr , cur_tcode , command_line ,
          code);
```

where:

1. tcf_ptr
   points to an iocb for the transaction control file (Input).

2. cur_tcode
   is set to transaction number just completed (Output).

3. command_line
   is a Multics command line that need not be enclosed in quotes unless it contains special characters.

4. code
   is a standard system error code (Output).

<u>Transactions</u>

A transaction is a unit of processing that has the appearance of taking place as an indivisible, atomic operation. Arbitrary procedures involving any collection of vfile_ indexed files may be invoked as transactions via this subroutine.

APPEARANCE

A partially completed transaction terminates either by a successful checkpoint operation, or by a rollback. That is to say, until a checkpoint occurs, the data base appears unchanged, except within the current transaction. Any data base modifications that a transaction makes appear simultaneously, outside the transaction which makes them, when the checkpoint takes place.

PURPOSE

There are two major reasons for encapsulating a procedure as a transaction. The first is to simplify the programmer's task of handling inconsistencies that can arise from interrupted operations that are not resumed (e.g., because of a system crash or an application program error). Second, in the event that a data base is shared among independent processes, the entire burden of synchronizing file access is removed from the programmer and automatically managed by the system transaction processing facility.

TCF SWITCH

Each independent transaction server (task or process that performs transactions) requires an I/O switch that associates the transactions with a particular data base. This switch is attached by the user to a permanent transaction control file (tcf), that is used in conjunction with the collection of files that compose a single logical data base.

TRANSACTION NUMBERS

A transaction has a unique identifying number associated with its tcf switch. Initially and after a checkpoint or rollback, this number is zero, indicating that no current transaction is defined for the given tcf switch. A transaction number is assigned automatically when a data base file attached via -transaction to the tcf switch is referenced, unless a nonzero number already has been set explicitly.

REFERENCE LISTS

A temporary reference list is automatically maintained with each tcf switch. This structure, which is implemented as an indexed file without records, contains the necessary information for keeping track of passive references made during the course of each transaction, so that asynchronous changes that might invalidate the transaction can be detected. The reference list also identifies all items modified during each transaction, in order to clean up the data base at checkpoint or rollback time.

Files

DATA BASE

Any collection of vfile_ indexed files may be defined as a data base upon which to apply transactions. All that is required is that a common tcf always be used in connection with references to any file in the given data base, and that the individual data base files be attached with the -transaction option specifying a tcf switch attached to the tcf for the data base.

TRANSACTION CONTROL FILE

The tcf is a permanent indexed file containing only index entries (i.e. no records). The user is responsible for its creation, but the tcf is implicitly manipulated by vfile_ and the various transaction_call_ routines, so that no explicit user operations on this file are required. If concurrent transactions are performed on a common data base, the -share option must be given in the tcf attachment, as well as in the attachments to the data base files that are shared.

TCF ENTRIES

Keys are added to the tcf when a transaction number is assigned for a new transaction. Each key's descriptor is a flag indicating the state of logical completion of a single transaction. Thus the atomicity of a transaction is reduced to changing the flag on its tcf entry.

OPENING CONSTRAINTS

In order to use transactions, the user must first attach and open the tcf for the data base. The user is also responsible for attaching and opening all data base files to be referenced before issuing any transactions, and none of these files should be closed within a related transaction.

ABNORMAL TERMINATION

When a checkpoint is attempted, or upon referencing a data base item previously read in the same transaction, it is possible that an error resulting from an asynchronous change in another transaction may be detected. This situation makes it impossible to correctly complete the current transaction, and the transaction must be aborted. To determine whether an unexpected error was caused by an asynchronous data base change, one may use the transaction_call_$status entry with the verify option.

See the description of the vfile_ I/O module in the MPM Subroutines. For a description of the command level interfaces corresponding to the transaction_call_ entries, see the description of the transaction_call command.

Entry: transaction_call_$assign

This entry reserves a unique transaction number for the current transaction and returns the new transaction number. The tcf switch must be opened for modific~~ation~~, so that a new entry can be created.

Usage

```
     dcl transaction_call_$assign  entry  (ptr,  fixed  bin(35),
          fixed bin(35));

     call transaction_call_$assign (tcf_ptr , cur_tcode , code);
```

where:

1. tcf_ptr
   points to an iocb for the transaction control file (Input).

2. cur_tcode
   is set to the new transaction number (Output).

3. code
   is a standard system error code (Output).

Notes

A transaction number can also be assigned via the transact_$number entry. The user is not required to preassign a transaction number at all, in which case one is automatically assigned upon making the first reference to a data base item for the new transaction.

Entry: transaction_call_$checkpoint

This entry attempts to complete the current transaction on a data base associated with a given transaction control switch. The current transaction number becomes undefined if the checkpoint is successful.

Usage

```
     dcl transaction_call_$checkpoint entry (ptr, fixed  bin(35),
          fixed bin(35));

     call transaction_call_$checkpoint   (tcf_ptr,     cur_tcode,
          code);
```

where:

1. tcf_ptr
   points to an iocb for the transaction control file (Input).

2. cur_tcode
   is set to transaction number just completed (Output).

3. code
   is a standard system error code (Output).

Entry: transaction_call_$number

This entry returns and optionally resets the current transaction number for a given tcf switch. The control file itself is not referenced or altered by this operation, permitting purely passive transactions to have only read access to the tcf.

Usage

```
     dcl transaction_call_$number entry (ptr, fixed bin(35),
          fixed bin(35), fixed bin(35));

     call transaction_call_$number (tcf_ptr, cur_tcode,
          next_tcode, code);
```

where:

1. tcf_ptr
   points to an iocb for the transaction control file (Input).

2. cur_tcode
   is the current transaction number (before changing) (Output).

3. next_tcode
   is the new transaction number or zero, if no change is desired (Input).

4. code
   is a standard system error code (Output).

Notes

When a transaction is known to involve no data base alterations, this entry may be used to initialize the transaction number to a unique value, thereby avoiding the necessity of modifying the tcf in order to reserve new code. Unless the transaction number has been initialized, a tcf entry is automatically assigned on the first reference to a data base item in the current transaction; the default behavior requires that the tcf be opened for modification.

Entry: transaction_call_$rollback

This entry undoes all modifications that have been made on behalf of the current transaction in a specified data base.

Usage

```
     dcl transaction_call_$rollback  entry (ptr,  fixed  bin(35),
          fixed bin(35));
     call transaction_call_$rollback (tcf_ptr, cur_tcode, code);
```

where:

1. tcf_ptr
   points to an iocb for the transaction control file (Input).

2. cur_tcode
   is set to the transaction number just aborted (Output).

3. code
   is a standard system error code (Output).

Notes

The effect of a rollback is logically invisible outside the current transaction, except possibly in its immediate cleaning up of accumulated garbage (after images). The transaction number for a rolled-back transaction is not reused. After issuing a rollback, the caller's transaction number for the given tcf switch becomes undefined, and the data base is restored to its state following the last checkpoint.

Entry: transaction_call_$status

This entry returns various items of information about a transaction for a specified tcf switch. These include the transaction number, its completion status, and optionally counts of passive and non-passive references.

Usage

```
      dcl transaction_call_$status entry (ptr, fixed bin(35),
          bit(36) aligned, ptr, fixed bin(35));

      call transaction_call_$status (tcf_ptr, cur_tcode,
          ts_status_word, ts_infop, code);


      dcl 1 ts_info based ( ts_infop ) ,
      2 flags,
           3 verify bit ( 1 ) unal, /* causes data base items to
be checked */
           3 pad bit ( 17 ) unal,
           3 version fixed ( 17 ) unal, /* set to current version
by user -- Input */
      2 passive_refs fixed ( 34 ) , /* Output */
      2 non_passive_refs fixed ( 34 ) , /* Output */
      2 pad fixed ; /* reserved for future use */ dcl
ts_info_version_0 static internal fixed options ( constant ) init
( 0 ) ;

                 ts_status_flags

      dcl 1 ts_status_flags based ( addr ( ts_status_word ) ) ,
      2 defined bit ( 1 ) unal , /* set if transaction number
found in tcf */
      2 status fixed ( 34 ) unal; /* 0 = incomplete , 1 = done , 2
= aborted */
```

where:

1. tcf_ptr
   points to an iocb for the transaction control file (Input).

2. cur_tcode
   is the transaction number for which status information is desired, or set to 0 to specify the current transaction. If this is zero, then the returned value is the current transaction number (Input/Output).

3. ts_status_word
   contains a code defining the status of this transaction as one of the following (Output): undefined - no tcf entry exists incomplete - in progress, but not yet checkpointed done -

successfully checkpointed (can't rollback) aborted - rolled back (can't checkpoint)

4. ts_infop
   points to a structure, ts_info, in which the counts of references made by the transaction are to be returned. If null, this information is not obtained (Input).

5. ts_info.verify
   if set, causes the list of passively referenced items for this transaction to be checked for possible asynchronous changes. If a change is detected, the returned code is set to error_table_$asynch_change, indicating that this transaction is unsuccessful (Input).

6. ts_info.version
   is the version number for this info structure, which should be set to ts_info_version_0 (Input).

7. ts_info.passive_refs
   is the number of distinct items referenced passively (not modified) so far in this transaction (Output).

8. ts_info.non_passive_refs
   is the number of distinct data base items modified so far in this transaction (Output).

9. code
   is a standard system error code (Output).

SECTION 4

TABLES

COMMAND_TABLE

Summary

The list of commands that a particular site may use is contained in a binary table known as tp_command_table_, maintained by the TP administrator of the site. There is one entry in the tp_command_table_ for each command; the entry contains the name, scheduling information and other attributes of the command. The tp_command_table_ is created from an ASCII segment known as the TP source command table (TPSCT).

The TP subsystem references the tp_command_table_ when it gets a command line. In order to make a change to the tp_command_table_, the TP administrator must modify the TPSCT and convert the TPSCT into a binary copy of the tp_command_table_. The tp_command_table_ can be installed only when the TP subsystem is not running.

Description of Source Language

The source language for the command table consists of a list of keywords with values. There must first be a global section specifying default values that differ from the system-provided defaults, and then a section for each command.

The syntax of a statement is as follows:

```
<keyword>:  <parameter>;
```

GLOBAL SECTION

The global section consists of a global statement followed by statements whose values are to be applied to all commands as default values. It must appear at the beginning of the command table source segment and is terminated by the first name statement.

The global statement syntax is:

global:

There are no parameters.

The statements in this section may contain any of the keywords described under the command section with the exception of the name and pathname keywords, and are overridden by statements in the command sections that have the same keywords.

If the global section is empty, the global statement is immediately followed by a name statement.

COMMAND SECTION

There must be a section in the source command table for each user command that is to be available in the TP subsystem. Each command section begins with a name statement and must also contain a pathname statement. The other statements are optional.

The keywords are described as follows:

name: name1, name2, ... namen;
: specifies the name(s) by which a TP subsystem user can reference the command.

pathname: "command_pathname";
: command_pathname is the pathname of the command.

call_convention: n;
: the parameter is a decimal integer specifying the way the arguments are to be passed to the command. Numbers 1 through 10 are reserved for use by the standard TP software; others may be site-defined. In this release, 1 is similar to normal Multics command processing in that the arguments are all treated as character strings; however no active function or parenthesis processing is done. This is the default.

cpu_time_limit: n;
: the parameter is a decimal integer specifying the maximum amount of cpu time, in milliseconds, that the command is to use in a single transaction. If the time limit expires, the transaction is aborted. The default is 0 (no time limit).

real_time_limit: n;
: the parameter is a decimal integer specifying the maximum amount of real time, in seconds, that the command is to use in a single transaction. If the time limit expires, the transaction is aborted. The default is 0 (no time limit).

immediate: string;
: if string is yes, the command is executed immediately in the process that handles the user's I/O. The command is invoked using Multics active function conventions and must not make data base references.

  If string is no, the command is queued as a transaction. This is the default.

priority: n;
: the parameter is a decimal integer specifying the priority of the command. Of two transactions queued at the same time, the one whose command has the lowest numbered priority is processed first. The priority value may be in the range from 0 to 1000. The default is 0.

retry_limit: n;
: the parameter is a decimal integer specifying the maximum number of times a transaction is to be reexecuted if checkpoint fails. After n+1 tries, the transaction is aborted. The default is 3.

Installation

The binary command table is compiled from the source by the tp_cvsct command (see the writeup in the Command section). The binary form is created in the working directory with the entryname source_entry_name.tpbct. To make this table accessible to the TP subsystem, it must be put in the TP directory and renamed to tp_command_table_. This installation must not be done while the TP subsystem is up.

Modifying the Compiler

This section describes the steps necessary to add new items to the command table.

1. Add the new items to tpcommands.incl.pl1.

2. Add code for processing the new items to tp_cvsct.rd, which is the reduction compiler source segment. See the writeup of reduction_compiler in the Tools Manual for information about how to use reduction_compiler constructs.

3. Type:
   reduction_compiler tp_cvsct
   to get tp _cvsct.pl1, which is the actual compiler source segment.

4. Compile tp_cvsct with the PL/1 compiler to get tp_cvsct, which is the compiler for the command table source language described in this manual plus the new items.

5. Add the new items to the command table source segment and recompile it.

6. Recompile all TP subsystem programs that use tpcommands.incl.pl1, making appropriate changes to use the new items.

DATA_BASE_DEFINITION_EXEC_COM

Summary

A worker process must open and ready each data base it will use before it starts to process any transaction. This is done through exec_coms run when the worker process is initialized. The exec_coms are prepared at the site, but recommended formats are presented in this manual to show the necessary functions. Following the standard formats enable a site to take better advantage of some future enhancements. The main worker exec_coms are described in Section 2 under "Initializing a Worker Process". This section describes the exec_com that actually opens the data bases. There is a labeled part for each way data bases are used; e.g., a data base used by one worker for retrievals and by another for updates would have two different exec_com sections.

The exec_com is called with two or three arguments: a file ID, the submodel/file pathname, and the I/O switchname. The file ID is the label of the appropriate section in the exec_com and may be some variation of the submodel/file pathname. The I/O switchname is given only when the data base is used directly through vfile_ (rather than through MRDS/MIDS).

Description_of_tp_init_data_base.ec

```
&goto &1
&label file_idi                    /* format for MRDS data base */
mrds_call open &2 null
mrds_call set_ctl_file tp.tcf_ [get_mrds_dbi &2]
mrds_call ready_file [tp_get_database_index &2] file_name1
          rdy_mode1 ...
&quit
```

```
      .
      .
      .
      /* similar format for MIDS */
      &label file_idj                 /* format for vfile_ data base */
      io attach &3 vfile_ &2 attach_options
      io open &3 open_mode
      &quit
```

Note for the vfile_ section: for the file to be checkpointed, attach_options must include "-stationary -transaction tp.tcf_" and the file must be indexed.

OPERATOR_TABLE

The operator table contains an entry for each user consisting of the user name and the password in coded form. The table can be modified only by the TP administrator using the tp_user command. See the command description in Section 3 of this manual.

SECTION 5

ERROR HANDLING AND RECOVERY

CRASH RECOVERY

A system crash does not usually result in loss of data. However, all processes are destroyed and so transactions are interrupted. When the system comes up again, new processes are created and all work in progress, which was associated with the old processes, is discarded unless already checkpointed. The aborted transactions are then rescheduled. This is done automatically by the TP system. When there is loss of data, most of the input queue (primarily records of transactions that have been processed) should be intact on disk. All the records from after a specified time corresponding to the state of the data bases should be made to look as if the transactions had not been run, and then those transactions should be rerun.

No facilities are provided with this release for journalizing the input queue or for adjusting data bases, queues, etc. in case of loss of data.

TRANSACTION ERROR HANDLING

Aborted Transactions

The TP subsystem recovers from unexpected errors by TPRs. When a TPR gets an unrecoverable condition, the worker process executive program aborts the transaction, sending the user a message to that effect. Then the worker process rolls back any changes made and goes on to the next transaction. Recoverable conditions include command_error_ and endpage. A site can insert its own condition handler to detect what it considers to be recoverable conditions.

Sometimes a transaction cannot be checkpointed because data base records used by the transaction were modified by someone else during the transaction. When this happens, the changes made by the transaction are rolled back and the transaction is rerun.

The maximum number of rerun attempts for a particular TPR is specified in the command table.

SECTION 6

GUIDELINES FOR WRITING TPRS

DESCRIPTION OF OPERATING ENVIRONMENT

TPRs run in an essentially standard Multics environment (see the MPM Reference Guide), although there are some restrictions. Multics access control is performed with respect to the worker process, not the individual user. Input from the terminal is not supported in the release described by this manual, so a TPR cannot interact with the user. A TPR's input comes only from arguments given on the command line and from reading files. A TPR should not do things that change the environment, such as changing search rules, although run units may be used. TPRs can call other programs in the Multics storage system, as long as the worker process has access to them. Because a TPR is not always invoked on behalf of a particular user, internal static storage should not be used. Likewise, external static storage (including FORTRAN common) should be used only within a program as defined by an invocation of the TPR by the TP executive, including any subroutines it calls. Thus any external static storage must be reinitialized by a TPR each time it is invoked by the TP executive. TPRs can be recompiled whenever the TP subsystem is shut down.

HOW TO DO I/O

Terminal Output

Output that is to appear on the terminal of the user who initiated the transaction must be written on the I/O switch iox_$user_output (PL/I sysprint, FORTRAN file 0 or 6, COBOL DISPLAY statement). No way is provided for a TPR to write on any other terminal.

File I/O

In the TP subsystem, all files to be referenced through vfile_ or the MDBM must be opened at the beginning of a worker process. This reduces overhead and eliminates some cleanup problems when TPRs abort. Thus TPRs themselves do not open files. Files cannot be accessed through standard language I/O mechanisms in this situation; all use of files must be through already opened iox_ switches with predetermined names or through the MDBM.

USING MRDS/MIDS WITHIN TP

All the data bases used by a TPR are opened and readied only once in the worker process, so that a TPR should not call dsl_$open, dsl_$ready_file or dsl_$finish_file. In order to obtain the data base index to use in other MDBM operations, a TPR must call dsl_$get_dbi, giving it the same submodel pathname as would otherwise be given to dsl_$open. Its usage is described in the commands section of this manual. A TPR may call a subroutine that uses a different submodel.

If using the MDBM from a TPR written in COBOL, the following division must be included just before the identification division:

```
CONTROL DIVISION.
DEFAULT SECTION.
GENERATE AGGREGATE DESCRIPTORS.
```

Device Output

Disk storage is accessible through the Multics storage system or Multics Data Base Management (MDBM). The line printer may be used through the dprint_ subroutine, which causes the I/O daemon to print the contents of a segment on the printer. (See the dprint_ subroutine description in the MPM Subroutines.) Cards may be punched via the I/O daemon but may not be read directly. They must first be read into the Multics storage system. (See the description for ? in the MPM Commands.) Tapes may be used through the I/O modules tape_mult_, tape_ibm_ or tape_ansi_. (See the description for iox_ in the MPM Subroutines.) If a TPR attaches a tape (accomplished through the Resource Control Package), it must also detach the tape and include a cleanup handler to detach the tape if the TPR aborts.

CHECKPOINT AND ROLLBACK FEATURES

The system provides a checkpointing facility, which updates a file with all changes made by the process since the last checkpoint. Any changes made before the checkpoint are invisible to other users. At any time before the completion of the checkpoint, all changes made since the last checkpoint may be rolled back, i.e. the file looks as though nothing had happened. In order to use this facility, a file must be attached with the -transaction option and all users of the file should use the same transaction control file.

In the TP subsystem, a checkpoint is performed automatically by the TP executive at the end of each transaction applying to all files used by that transaction. It is recommended that TPRs not perform intermediate checkpoints as this may affect the TP recovery mechanism.

However, if a TPR takes a long time to execute, there is an increased probability that some of the records read would have since been changed and checkpointed by another transaction. If this is likely, the TPR should be written to include a call to transaction_call$status with the verify flag set. This indicates whether a checkpoint would succeed if issued at that point in time. If it wouldn't, there is no reason to continue further. The TPR should call tp_rollback_transaction_ (with no arguments), which will roll back the current work and reinvoke the TPR.

CONTENTS

COMMENTS ON THE TP MANUAL

One facility that is not yet discussed in the manual but will probably be included is that the I/O and worker processes can also be run as daemons.

Sometime during the next year a TP PLM should be written, since most sites will need to make some changes to suit their needs. The TP software is designed to facilitate such changes.

A NAMING PROBLEM

One item that still needs to be resolved is how the location of the TPR should be represented in the command table. Two methods are proposed: full pathnames and reference names. In either case, the name (at least in the source) will end in $entry_name when necessary to specify the particular entry point.

Some advantages of using pathnames are:
- the location of a TPR is more obvious;
- the mechanism is simpler;
- it is safer, since the exact location is specified;
- it is easier to check whether the correct version is being used.

Some advantages of using reference names are:
- search rules are used (these can be made reasonably safe by putting the list in a segment which all worker exec_coms use);
- they are more multicious;
- the command table does not need to be changed if TPRs are moved around;
- different versions of TPRs can be tested without changing the command table (the command table would be linked to from a test subsystem).

In either case, a worker process obtains a pointer to an entrypoint only once, when it first has to execute a particular TPR.

SOME FEATURES NOT INCLUDED IN THE FIRST RELEASE

Recovery from loss of data after a crash when ESD fails. The data bases are left in an inconsistent state. The backup mechanism is not adequate in this case unless the data base was backed up when it was not being used. This is an important but very difficult problem and will be tackled in the future. Currently, there is available a force write primitive whose use can make things somewhat safer. It is not used by the TP subsystem.

Journalization of the input queue. The queuing of a transaction should be acknowledged only after the system is sure the record is out on disk or tape. Then the user will know that if the transaction is acknowledged, it will always be executed. This feature will be available when the journalization option to vfile_ is implemented.

Performance testing. In order to test the TP subsystem's performance with particular TPRs in a useful way, it will be necessary to set up an internal driver mechanism. In this case, transactions are read from a file instead of from terminals which allows them to be entered at a much faster as well as more controlled rate. The details of this have not been worked out, but although it will be necessary for benchmarks, it will probably not be done for the first release.

Access control within TP. The first release does not provide such facilities as limiting the number of commands a user can execute or restricting the data bases a user can reference.

```
/* BEGIN INCLUDE FILE ... tpcommands.incl.pl1 */

dcl        1 tp_commands                    aligned based(tpcot_ptr),

           2 character_region               char(16384) aligned,

           2 buckets(0:511)                 aligned,
             3 slots(4)                     aligned,
               4 character_index            fixed bin(17)        unaligned,
               4 command_table_index        fixed bin(17)        unaligned,

           2 commands(0:2047)               aligned,

             3 command_number               fixed bin(35),
             3 command_type                 fixed bin(35),
             3 modify_time                  fixed bin(35),

             3 scheduling_info,
               4 cpu_time_limit             fixed bin(35),
               4 real_time_limit            fixed bin(35),
               4 drop_dead_time             fixed bin(35),
               4 deadline                   fixed bin(35),
               4 delay_time                 fixed bin(35),
               4 max_concurrent             fixed bin(35),
               4 expected_cpu_time          fixed bin(35),
               4 deadlock_priority          fixed bin(35),

             3 access_info,
               4 tp_level                   fixed bin(35),
               4 tp_attribute               bit(36)   aligned,
               4 pad_1                      fixed bin(35),
               4 pad_c                      fixed bin(35),

             3 control_info,
               4 failure_limit              fixed bin(35),
               4 retry_limit                fixed bin(35),
               4 admin_out_of_service       bit(1)    unaligned,
               4 failure_out_of_service     bit(1)    unaligned,
               4 test_mode                  bit(1)    unaligned,

             3 execution_info,
               4 pathname                   char(168) aligned,
               4 callname_index             fixed bin(35),
               4 callname_total             fixed bin(35),
               4 immediate                  bit(1)    unaligned,
               4 interactive                bit(1)    unaligned,
               4 conversational             bit(1)    unaligned,
               4 call_convention            fixed bin(35);

dcl        tpcot_ptr                        ptr;

/* END INCLUDE FILE ... tpcommands.incl.pl1 */
/* BEGIN INCLUDE FILE ... tp_master_table.incl.pl1 */

dcl (tpmp, wkp, iop, tp_ttep, irp) ptr;

dcl 1 tp_master_table based (tpmp) aligned,          /* Segment structure */
      2 lock bit (36),                                /* Allocation lock */
      2 master_procid bit (36),                       /* process which issued start_tp */
      2 worker_error_event fixed bin (71),            /* event-call channel for worker errors */
      2 io_error_event fixed bin (71),                /* event-call channel for io errors */
      2 logout_event fixed bin (71),                  /* event_call channel for logouts */
      2 startup_time fixed bin (71),                  /* time of start_tp */
      2 sysdir char (64),                             /* table directory */
      2 n_io_procs fixed bin,                         /* current count of io procs */
      2 n_worker_procs fixed bin,                     /* current count of workers */
      2 io_proc_head bit (18),                        /* head of ioproc chain */
      2 worker_proc_head bit (18),                    /* head of worker chain */
      2 transaction_no fixed bin (35),                /* transaction counter */
      2 pad (5) fixed bin,
      2 area area (1000);                             /* Allocation of everything else */

dcl 1 worker_entry based (wkp) aligned,               /* allocated when worker attaches */
      2 type fixed bin,
      2 next bit (18),
      2 lock bit (36),                                /* zero if worker is blocked */
      2 name char (8),                                /* name */
      2 control_state fixed bin,                      /* state of tp_run_worker */
      2 io_module_state fixed bin,                    /* state of tp_attach */
      2 have_error bit (1) aligned,                   /* error info indicator and lock */
      2 logout_sw bit (1) aligned,                    /* master -> worker to log out */
      2 delayed_logout_sw bit (1) aligned,            /* master -> worker to log out when all done */
      2 logged_out_sw bit (1) aligned,                /* worker has logged out (set by worker) */
      2 procid bit (36),
      2 wait_list,
        3 nchan fixed bin,
        3 input_event fixed bin (71),                 /* worker blocks on this */
      2 cur_xcn_info aligned,                         /* info about xcn being executed */
        3 xcn_no fixed bin (35),                      /* transaction number */
        3 tpr_name char (32),                         /* name of tpr as given by user */
        3 user_name char (32),                        /* name user logged in with */
```

```
    2 error_info,
      3 flags aligned,
        4 abort_sw bit (1) unal,                    /* abort all IP processing */
        4 queue_sw bit (1) unal,                    /* can't checkpoint queues */
        4 read_error_sw bit (1) unal,               /* can't read from input queue */
        4 logout_sw bit (1) unal,                   /* proc is logging out */
        4 pad1 bit (32) unal,
      3 command_no fixed bin,                       /* index of current command */
      3 condition char (32) var;                    /* name of condition */

dcl 1 io_entry based (iop) aligned,                 /* allocated at ioproc startup */
    2 type fixed bin,
    2 next bit (18),
    2 lock bit (36),                                /* zero if ioproc not typeing out */
    2 name char (8),
    2 writer_state fixed bin,                       /* state of tp_writer_ */
    2 procid bit (36),
    2 stop_input_sw bit (1) aligned,                /* master -> io proc to stop accepting input */
    2 logout_sw bit (1) aligned,                    /* master -> io proc to log out after finishing c
    2 logged_out_sw bit (1) aligned,                /* io proc has logged out (set by io proc) */
    2 terminal_head bit (18),                       /* relp to list of ttys */
    2 output_busy bit (36),                         /* zero if io proc is blocked for output */
    2 output_driver_event fixed bin (71),           /* event-call channel */
    2 dial_chn fixed bin (71),                      /* if a dial server */
    2 in_iocbp ptr,                                 /* local storage */
    2 out_iocbp ptr,
    2 tct_iocbp ptr;

dcl 1 tp_tte based (tp_ttep) aligned,               /* declaration of a single terminal entry */
    2 type fixed bin,
    2 next bit (18),
    2 active fixed bin,                             /* state of entry (^=0 -> active) */
    2 name char (12),                               /* channel name */
    2 user_name char (32),                          /* name user logged in with */
    2 twx fixed bin,                                /* ring 0 twx */
    2 tty_state fixed bin,                          /* channel state */
    2 tty_type char (32),                           /* terminal type */
    2 tty_id_code char (4),                         /* id code */
    2 cur_line_type fixed bin,
    2 baud_rate fixed bin,
    2 tra_vec fixed bin,                            /* transfer-vector for event call channel */
    2 event fixed binary (71),                      /* name of event call channel associated with TTY
    2 dim fixed bin,                                /* 1=hcs_$tty, 2=net_as_, 3=g115_a_ */
    2 pad (4) fixed bin,
    2 control,
      3 inhibit bit (1) unal,                       /* Inhibit output - opr is typing in */
      3 output_wait bit (1) unal,                   /* hardcore wouldn't take more output */
      3 output_pending bit (1) unal,                /* some output queued */
      3 quit bit (1) unal,                          /* opr hit QUIT */
      3 conversation bit (1) unal,                  /* TRUE if conversational xcn */
      3 unused bit (31) unal,
    2 inct fixed bin,                               /* input line count */
    2 outct fixed bin,                              /* output */
    2 user_state (4) fixed bin;                     /* storage for TPR */

dcl 1 input_record based (irp) aligned,
    2 transaction_no fixed bin (35),                /* transaction number */
    2 vfile_transaction_no fixed bin (35),          /* transaction number assigned by vfile_ */
    2 user_name char (32),                          /* name user logged in with */
    2 reply_ioproc_rel bit (18),                    /* offset of reply I/O proc entry */
    2 reply_tty_rel bit (18),                       /* offset of reply tty entry */

    2 time_entered fixed bin (71),                  /* time input queued */
    2 time_execution_begun fixed bin (71),          /* time tpr was invoked */
    2 time_execution_done fixed bin (71),           /* time tpr finished (incl retries) */
    2 execution_cpu_time fixed bin (71),            /* cpu time spent by tpr */
    2 execution_paging fixed bin (35),              /* page faults taken by tpr */
    2 worker_paging fixed bin (35),                 /* page faults taken by worker */
    2 input_paging fixed bin (35),                  /* page faults taken while queuing input */
    2 no_of_retries fixed bin,                      /* no of times tpr was invoked */
    2 completion_flags aligned,                     /* indicates how tpr finished */
      3 normal bit (1) unal,
      3 error_abort bit (1) unal,
      3 checkpoint_abort bit (1) unal,
      3 cpu_time_abort bit (1) unal,
      3 real_time_abort bit (1) unal,
    2 command_name char (32),                       /* name of tpr as given by user */
    2 command_index fixed bin,                      /* index of tpr in command table */
    2 buffer_length fixed bin,                      /* length of input buffer in chars */
    2 argument_buffer char (2048);                  /* input minus command name */

dcl (WORKER init (3), IO init (4), TERM init (5)) fixed bin static options (constant);

/* END INCLUDE FILE ... tp_master_table.incl.pl1 */
```