THE DESIGN AND IMPLEMENTATION OF QNEMU: AN INTERACTIVE QUEUEING NETWORK ANALYSIS PACKAGE

by

Neil Ray Ladyman

A thesis presented in partial fulfulment of the

requirements for the degree of

Master of Science in Computer Science

at

Massey University

January 1984

ABSTRACT

In this thesis a number of design objectives are discussed for the generation of interactive packages. Two of the most important are considered to be portability and user friendliness. Unfortunately these can at times produce conflicting design aims.

The objectives identified are incorporated into the design and implementation of a menu driven package that analyses networks of queues.

A brief description of the way the package was implemented is included, as is a discussion on the validation of the package so far carried out.

The primary use of the package is seen to be in the performance evaluation of computer systems, but it need not be restricted to this class of application.

ACKNOWLEDGEMENTS

In presenting this thesis I would like to take this opportunity to express my thanks to the following people:

Firstly my supervisor, Tom Docker, for his guidance, and criticism throughout all stages of the work reported in this thesis.

To the numerous members of both the Computer Science Department and the Computer Centre at Massey University for their interest shown and support given especially in the final stages. Particular thanks to Walt Abell whose constructive criticism was invaluable in shaping this thesis.

To my many friends whose encouragement provided much incentive towards the completion of this thesis.

Finally, to both my parents, but especially Joy Ladyman who assisted in the typing of this thesis.

Table of Contents

Figures and Tables

page

0 INTRODUCTION

The widespread development of interactive systems has brought with it a requirement for much improved interfaces which are not based on the concept of the card based interfaces of the earlier systems. The most widespread form that this improved interface takes currently is the menu. However menu driven interfaces have problems in satisfying users with varying ranges of skill and experience. In addition many menu driven systems contain a complex tree of menus which can cause the user to become disorientated.

Users without a knowledge of interactive computer systems can be easily put off using a system if it is difficult to understand and use. Our main interest lies within the area of statistics (analytic modelling in particular), where many of the packages, such as Minitab and SPSS, are little more than interactive batch entry, punched card image systems. What is needed, is software which provides users with an interface to translate their requirements into a simple set of commands that produce clear results.

We begin in Chapter One with a general discussion of desirable design goals for interactive packages, with particular reference to those packages that require complicated input and output.

Following on from this discussion, a description is given in Chapters Two and Three of an analytic queueing network modelling package (QNEMU) that was designed using the design goals of Chapter One.

An important aspect of any modelling package is its validation, and so Chapter Four describes the validation exercises so far carried out on the QNEMU package.

Chapter Five gives some indication of further areas of research related to the QNEMU package.

The body of this thesis that relates specifically to the QNEMU package subsumes a knowledge of modelling, and performance modelling in particular using analytic queueing theory techniques. Those readers who require a "refresher" of performance evaluation techniques should read Appendix H; of particular relevance are sections H.7 and H.8.

1 DESIGN of an INTERACTIVE PACKAGE

1.1 Introduction

There is a need for user friendly interactive interfaces in a wide number of applications. This need is particularly true in the area of statistical analysis where there are a number of interactive packages available, but the level at which the software "fits the user" is very poor (such as SPSS, MINITAB and GASP IV). Quite often packages of this type tend to look very much like a remote entry card based system, and provide a FORTRAN-orientated interface, both of which are difficult for non-computing users to understand or use effectively. Three papers which provide useful reading on the design of interactive systems are [Ledg 80, Ling 80, Pars 79].

Two important factors in the design of an interactive package are:

- Portability
- User friendliness

The level of portability of a software package gives an indication of the ease with which that package can be transported between (dissimilar) systems. As an example, a package written in ANSI standard Cobol is likely to be more portable than the same package written incorporating manufacturer-defined extensions to the ANSI

standard. In its turn the non-standard Cobol is likely to be more portable than the same package written in IBM360, or any other, Assembler.

The level of user friendliness of a package gives an indication of how well the package "fits the user". The closer the package is to the way the user wants it the more user friendly it is.

The level of portability of a package is in principle an objective measure, compared to the level of user friendliness, which is largely a subjective measure strongly related to the user's own ability compared to the level of user the package is designed for. These two design goals can at times compete, as improving the portability may well mean omitting a feature that may be very useful, such as reverse video, but which is system specific.

1.2 Portability

The rapidly increasing costs associated with software development and maintenance are enough in themselves to suggest that serious consideration should be given to the portability of any software package. As suggested above, a major factor in the portability of a package is the choice of language (subset) used.

There are a number of levels of software portability, with one of the first levels being the transfer of software between computer systems of the same manufacturer. This can be divided further, as there can be different hardware and different system software on the systems which impact heavily on the portability of software packages. This is the case on the two Prime systems at Massey University, where some software is only available on one of the systems as the two systems run under different releases of the operating system. The problem of portability becomes much worse as software is transferred between the systems of different manufacturers.

One of the readily identifiable problem areas with the transferring of software between two systems is the manner in which the I/O is handled on each system. Any differences can largely be overcome by using only the standard I/O routines of a language such as FORTRAN 77, and resisting the temptation to use system dependent I/O routines. A very good review of guidelines for portable programming is found in [Wall 82]. A useful guide to writing portable FORTRAN 77 is [Larm 81].

With a view to its portability, the decision was made early on in the development of the QNEMU package [Oldf 82] to use FORTRAN 77 as the implementation language.

1.3 User Friendly

The user friendliness of a package is an abstract term which can best be judged by the use of the system, or at least the observations of the users of the package. One useful guide to the user friendliness of a system is how much the system is actually used. Unfortunately this can only be judged once the package has been completed, and at best indicates whether the package requires immediate modification.

Any computer package should be designed with the user as the most important person to satisfy. Therefore it is important to identify the users and their particular requirements at the time a system is being designed. This may be difficult if the package is new, but a number of design iterations where modifications are made from user feedback should be made [Norm 83].

The latest in user friendliness of systems seems to be in the areas of direct manipulation and the provision of windows acting as scratchpads for data. Direct manipulation is available on a very limited range of current computer systems, with windows being more common. Unfortunately the portability to date of such features is very low due to their hardware dependence [Schn 82].

The next level down in user friendliness is in intelligent, interactive, menu-driven packages, where the user is guided through the package with a minimum number of keystrokes and varying levels of menus [Brow 82].

The levels lower than menus include the batch style interactive input. In packages of this type, the input data are entered on each run and cannot be altered without reentering the complete set of input [Ling 80]. Most statistical packages, including Minitab and SPSS, have this level of interface. Some have a provision for saving data, but the interface is still structured at a fairly low level.

Below this are the batch systems and the actual programming languages used on the computer [Sand 78].

1.3.1 Menu Driven

A menu driven system can become annoying for an experienced user if the menus cannot be reduced in some manner to remove the bulk of the information. This is particularly noticable on a system with slow screen refresh where the menus appear to take an unreasonable time to be displayed.

In addition to varying the level of the prompt menu, the ability to enter a number of commands in a single input line, without the prompt menu for each entry being displayed to the user, is a useful feature, and tends to lead to more efficient processing.

1.3.2 Help Facilities

In addition to the package being menu driven, the user should also have available a comprehensive help facility. There are a number of levels at which this can be provided.

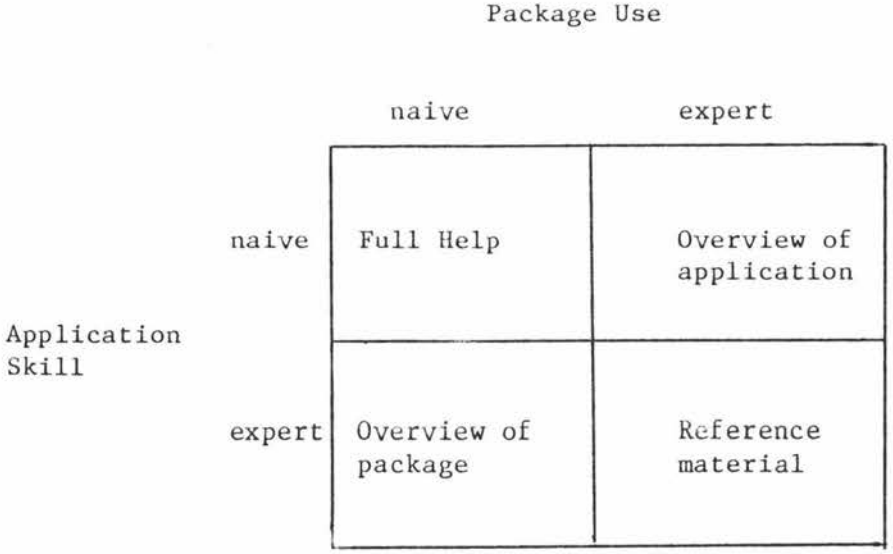
There should be off-line user manuals with detailed reference material on both the general operations of the software and the commands. Having an off-line facility is important as it permits the user to access the relevant information when a terminal is not available. This feature is particularly pertinent in an University environment where resources such as terminals are in heavy demand. However, if it is the only source of reference, it may mean that the user has the inconvenience of carrying the documentation about at all times.

The major problem with off-line facilities is that of keeping the material up-to-date. This can be made easier if the help information is maintained on-line and is updated whenever a change is made to the software, or when the help information is improved. To facilitate this, help information should be read into the software package as data, rather than being compiled as part of the package.

Various levels of online help facility should be provided as users vary in the amount of help they require. It is difficult to split users into distinct groups as both the level of application skill

and level of the awareness of the specific package influence the help level required by the user (See figure 1.1).

figure 1.1 Different Types of Package User



A general overview of the package and brief description of the commands would be useful to the inexperienced user. As the user becomes more skilled in the use of the package, a help facility would become more of a reference tool, therefore a detailed description of the commands would also be useful. Depending on the range of users, there may well be a requirement for detailed information on the application being available in the help facility as well.

The ability to tailor the help facility for particular installations or groups of users is considered a very useful feature.

1.3.3 Commands

Interactive command languages should be designed with the two objectives of ease-of-use and clarity of meaning in mind. The user should be able to express his requirements in a natural manner which bears a close relationship to the way the requirements would be specified if a computer was not being used. The language should be such that it can be seen what the meaning of a command is without further documentation. The command language and the menus are complementary, each should be consistent and reflect the structure of the other.

To help make a command language more acceptable to different users, the language should be designed to accept (user-defined) synonyms for the commands. (E.g. LIST or PRINT.)

The ease-of-use of a command language can be improved by allowing the use of both abbreviations and common mis-spellings of the keywords in the commands. Once a user is familiar with a system, abbreviations can allow the number of keystrokes to be kept to a minimum [Trot 78]. Consistency in abbreviating is important so

that a user can easily determine an abbreviation without context sensitive restrictions. For example, if an abbreviation can be used to obtain help information throughout a menu-driven package, it should not be a different abbreviation within different menus.

The use of free format commands, where the order of symbols is not significant, can also be seen to improve the ease-of-use of an interface. For example, "Print fredfile" or alternatively "fredfile Print". There are a number of commands where the order of the symbols contains information and cannot be entered in free format. For example the numerical operation subtract, where "3 - 5" and "5 - 3" do not represent the same operation.

1.3.4 Responses

Responses to a user can be identified at three distinct levels: error response, warning response, and status response. In essence, they all serve to provide the user with retrospective information on the user input and the operations of a package. The most important is the error response which alerts the user to the fact that an invalid event has occurred. Hopefully the user can overcome the error using one or more of the information available in the response, menu or help facility. An audio response as well as a visual response can be useful: for example a beep at the terminal, which can indicate to the user that an attempt to input

non-valid data has been made. If an error has not occurred there are two further levels of response which can be identified.

There are cases where the user may well have entered a command which is valid but is quite possibly not consistent with the state of the package. For example a warning may be given if the user attempts to overwrite an existing file with a new file. The warning does not require an audio response as this should be reserved for error responses. If a signal such as an audio response is used too frequently it will lose its significance to the user.

The third type of response, and arguably the least important, is the status response, which indicates to the user the current status in the execution of a package.

Every attempt should be made to stop the user receiving meaningless or wrong responses.

1.3.5 Verification and Validation checks

All user input should be verified which involves both checking of the input symbols and checking of the the input structures. The checking is to determine whether meaningful results can be produced by the semantic routines before entry to them. Without checks inconsistencies in the results and wastage of resources are likely to occur. The output should also have consistency checks, either inbuilt into the package or at least provided for the users to check if they feel the results may be incorrect.

In general the package should carry out automatic checking wherever it is required and not rely on the user, who may have little or no knowledge of the checking procedures.

1.3.5.1 Scanning Checks

All possible input (including control keys), should be verified by the scanner so that invalid characters are detected and an error message printed. There will be some keys (control p or break key) which will still terminate the program abnormally at the user's request, but all other characters should be verified. If a manual or program defines the valid character set, the scanner should ensure that no invalid characters get past it. This is especially important when the user defines names with

one or more unprintable characters. If not detected as invalid by the scanner the user will have difficulty in determining what the name is at a later stage.

1.3.5.2 Semantic Checks

Semantic checks are important in most applications to ensure that the input data and the structures they describe are meaningful. Correct data means that the algorithms in the semantic routines will be able to carry out their required functions. Every possibility of abnormal termination, such as divide by zero, should be checked for by the software and not by the user, and a suitable response returned to the user. Any attempt by the user to delete non-existent entities or create duplicate entries should be checked by the software.

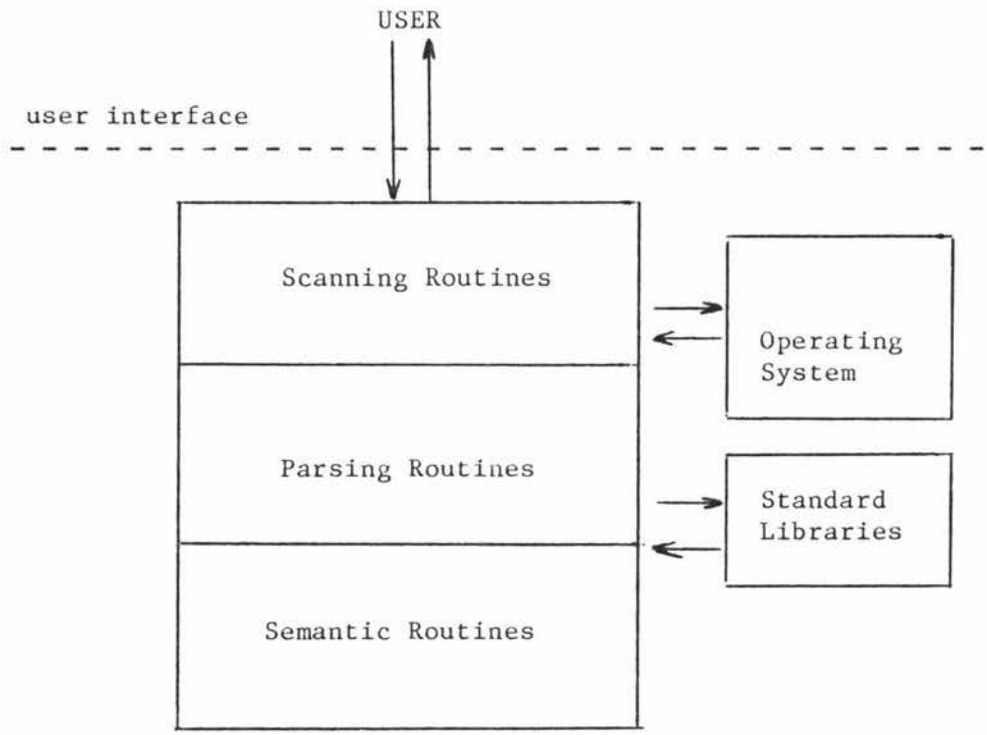
1.3.5.3 User Checks

There are some situations where the software cannot easily provide checks, or where the checks are only required a small number of times. In these cases it may be more appropriate to provide well documented user checks.

1.4 Program Structure

It is suggested that interactive packages can be well structured by using many of the techniques developed for compiler construction. The essential program components of an interactive package are scanning, parsing and semantic layers, which are equivalent to the layers found in an interactive compiler or interpreter. The user interface comprises of both input and output, with the input originating from the user and the output originating from all three layers of the package (see figure 1.2).

figure 1.2 General Structure of an Interactive Package



2 DESCRIPTION of the QNEMU PACKAGE

2.1 Introduction

QNEMU is a package designed for the analysis of networks of queues. QNEMU (Queueing Network Evaluator of Massey University) which is designed to be used interactively from terminals, incorporates the design goals identified in Chapter 1. The first implementation of the system is on the Prime 750 system at Massey University.

In QNEMU users are able to create permanent files, each of which is capable of holding a library of queueing networks, and are able to edit, save and analyze the networks held in these libraries.

The first design goal, portability, has been met in part by the use of standard FORTRAN 77 I/O routines for both disk I/O and terminal I/O. This means that the I/O may not be as efficient or as user friendly as it could have been if specific system dependent routines of the Prime system had been used, but hopefully it makes the package relatively portable in this area. The portability design goal has also meant restrictions in such areas as the provision of archiving of networks which would use system dependent routines.

In accordance with the second design goal, user friendliness, the package is designed to provide both a menu-driven interface, with

different levels of menu details, and a wide range of help facilities. In addition to the different levels of menus, limited menu by-pass is provided so that the user may skip the menus displayed at each interaction. The commands have also been designed to include synonyms and common mis-spellings, with the facility to completely tailor the commands for each installation without the need to recompile the program. This applies to the help facility and menu banners as well. As an example, a German equivalent for all the commands can replace the current English version.

2.2 QNEMU Overview

QNEMU is designed to be as terminal independent as possible. The package is designed around a 24 line 80 column screen, but a command exists by which the user can specify the number of lines on a screen. In QNEMU terminolgy the information displayed on a screen is called a page. Screens with less than 80 columns are not suitable for the running of QNEMU.

The process of starting QNEMU, manipulating networks and exiting is called a session. A session is started as soon as the following PRIMOS command is entered:

```
RUN QNEMU
```

A QNEMU session is made up of three distinct phases, namely

- Initialisation
- Analysis
- Termination

In the initialisation phase, the variables in static common are automatically set up. During this phase a series of messages is printed to the terminal to indicate the progress of the initialisation (see figure 2.1).

figure 2.1 Initialisation messages

```
|-| |-| |-| |-| |-| |-| |-| |-| |-| |-| |-| |-|
|-|                                             |-|
|-| Please be patient, QNEMU is initialising    |-|
|-|                                             |-|
|-| |-| |-| |-| |-| |-| |-| |-| |-| |-| |-| |-|

                    T E N

                   N I N E

                  E I G H T

                  S E V E N

                    S I X

                   F I V E

                   F O U R

                  T H R E E

                    T W O

                    O N E

  * * * * * * B L A S T   O F F * * * * * *
```

Once this phase is completed the analysis phase commences. The dialogue between the user and QNEMU begins with QNEMU requesting from the user the file name of the library file to use during the session.

The second phase continues until the user terminates the session with an EXIT command. The termination phase follows, and is similar to the initialisation phase in that it requires no input from the user.

2.2.1 Commands

The QNEMU commands have been designed with the features described in section 1.3.3, except that there is no provision for free format commands. QNEMU commands begin with a keyword, usually a verb, and may be followed by a number of options that modify the effect of the keyword. The keywords are chosen to represent the action required as closely as possible, and where there are a number of common synonyms for the same action each synonym is provided. Abbreviations are also provided to reduce the number of keystrokes and hopefully improve the ease of use. In using the full keyword the clarity of a command may well be increased for the inexperienced user.

In most menus the * character can be used to indicate a default command or option. The default option for each menu is indicated in the menu by either a brief line of explanation (see figure 2.7) or as an * beside the default option in the menu (see figure 2.6).

The QNEMU commands can be divided into the following categories:

Session Command

EXIT

Current Network Commands

CREATE GET SAVE RECOVER REMOVE

Change Commands

EDIT DELETE INSERT

Library Network Commands

DIRECTORY REMOVE RETRIEVE SAVE

Analyze Command

ANALYZE

Display Command

LIST

Information Command

HELP

The syntax of the QNEMU commands can be found in APPENDIX A.

A string of commands may be entered on a single line and can be separated by semicolons, commas or blanks. Example:

```
GET prime750 ; EDIT SDT cpu * 3.0 Q , ANALYZE ALL 20
```

Some commands must appear as the last command on a line, as their operation leads to the display of information on the screen. The user must respond in a specified way to indicate that the currently

displayed information is no longer wanted. Three such commands are LIST, DELETE and HELP, which all have display menus of information for the user to view. Examples:

```
LIST NAME ALL
```

```
HELP SYN
```

All the commands and the command options can be entered in lowercase or uppercase. Example:

```
get prime750  LisT **
```

All commands except EXIT allow abbreviations, synonyms and common mis-spellings. These cannot be used as names of nodes or networks (see Appendix D). Example:

```
EDIT SDT cpu * QUIT     or          E SDT cpu * q
```

2.2.2 Help Facilities

The design of QNEMU incorporates a comprehensive help facility, so that the user has on-line access to a range of information at any time while running the package. The design of the on-line help facility is such that the text in the help file can be altered without recompiling the program. In addition to the on-line information a hard-copy user manual is also available [QNEM 84].

The on-line HELP command allows the user to select from a range of help information made up of:

- QNEMU overview;
- quick index of the commands;
- detailed reference information on the commands;
- list of the reserved words (abbreviations, synonyms and common mis-spellings);
- BNF description of the command syntax.

In addition, at any point when entering a command, the user can key in an HSYM (i.e. H) and the help information for that command will be displayed to the user. The help information of all the commands are accessible at this point, by using the display subcommands to shift the display window.

2.2.3 Networks

A network is classified either as a library network or as the current network. There can only be one current network but there may be a number of saved networks in a library file. The library file can be archived or copied using the operating system commands available at the installation the package is running on. Within the library file the saved files can be backed up by saving the network under a different network name in the library file.

Every network has a unique name within a library file which is case independent; that is, a name in lowercase is the same as a network name in uppercase or any other combination of upper and lowercase characters. This reduces the confusion over upper and lowercase strings having different meanings when most users tend to regard them as interchangeable.

Examples of unique valid network names.

Networklongnameis.upto>26 SHORT-network

Example of names for the same network.

NETWORKONE networkone NetwOrkoNe

Examples of invalid network names.

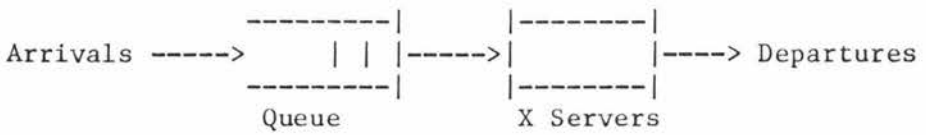
3Network KILLER;THREE ONE NETWORK

2.2.4 Current Network

There is only ever one current network and its name is the name of the network referenced by the last CREATE, GET or SAVE command. Any changes made to the current network have no effect on a network of the same name in the library file should one exist, unless the changed version is saved using the SAVE command.

Each node in the current network is made up of a queue and one or more servers (see figure 2.2). The input parameters for each node in the network are the queueing discipline, the service rate of each server, and the routing probabilities to the other nodes in the network. In figure 2.2 "Arrivals" represents the arrivals from the other nodes and "Departures" represent the departures to the other nodes.

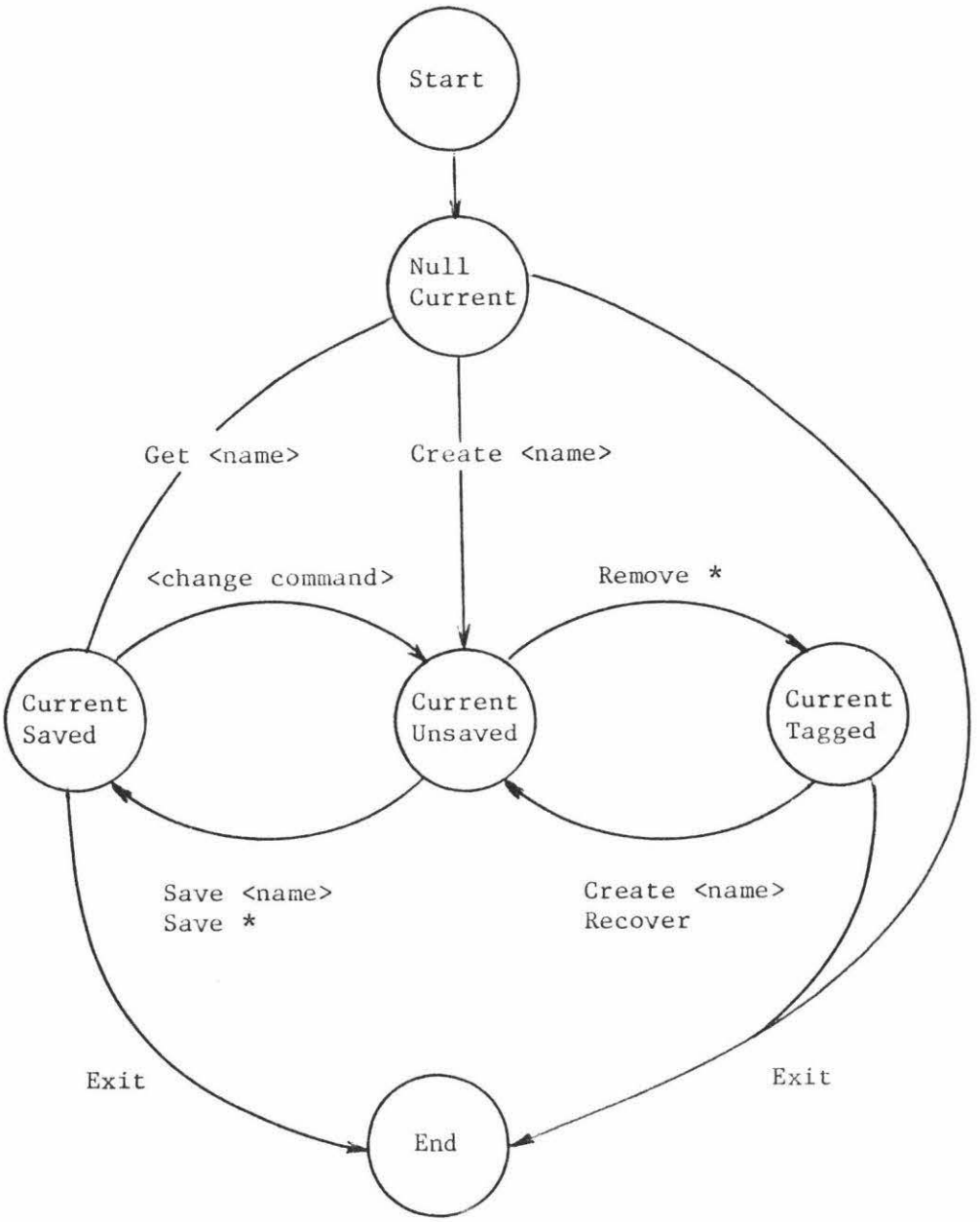
figure 2.2 A Typical Node in a Queueing Network



The calculated performance statistics are the mean server utilisation, mean queue length, mean wait time and mean node throughput at each node in the network for the specified workloads. The average workload can be a real or an integer value.

Each node in the current network can be referenced by its unique node name, which is stored, accessed and displayed in lowercase.

figure 2.3 FSM for the current network commands



Note 1: Change command includes the QNEMU commands EDIT, DELETE, and INSERT.

2: Commands not shown cause no state change.

3: <name> is a valid network name.

4: The asterisk (*) is context sensitive but means the current network in both the above cases.

2.2.5 Library Networks

When the current network is saved with the SAVE command it becomes a library network in the library file. It remains in the library file, under the saved name, until the network is removed with the REMOVE command. The contents of a library network can be changed after the network is made the current network with the GET command, then the network is both a saved library network and the current network. The library network is not affected by the changes made to the current network until the SAVE command is used again. After a SAVE the old library version is replaced with the new version, unless a new network name is used as the save option. The following examples show the case of saving under the same name and under a new name, respectively.

Example 1

```
GET oldnetwork
EDIT rpb ...
 ...
SAVE *
```

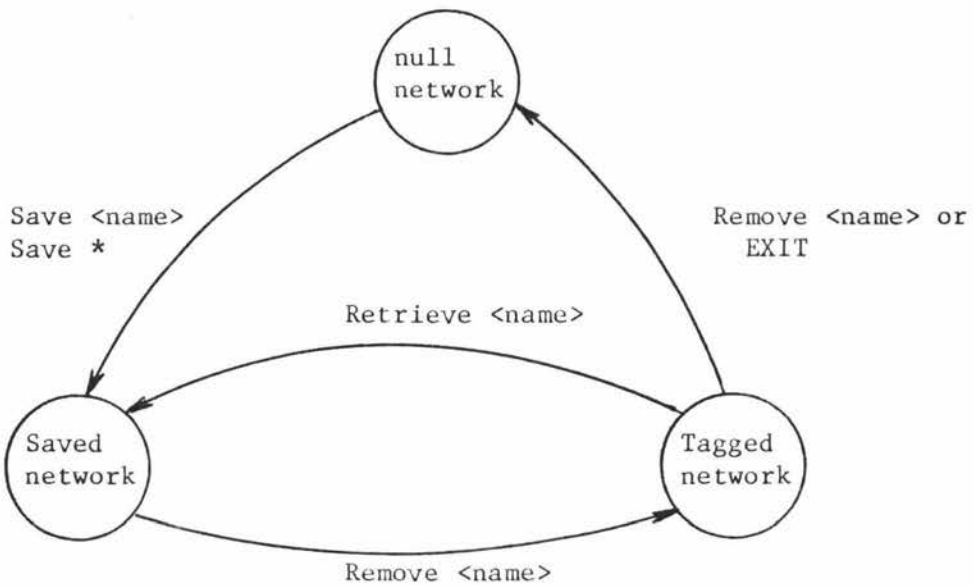
Example 2

```
GET oldnetwork
EDIT rpb ...
 ...
SAVE newnetwork
```

In the second example a new network name is used, so the old library version of the network is left unchanged and a new network is saved in the library file.

The networks in the library file can be listed using the DIRECTORY command. The date each network was last saved, and whether or not each network is tagged for removal, is also listed.

figure 2.4 FSM for a saved network in the library file



Note that "null network" signifies that a saved version of the named network does not exist, and that a current network of a different name may well exist.

2.2.6 QNEMU Menus

There are two types of menus used in the QNEMU package. Respectively, they carry out the two distinct tasks of prompting the user for input, and displaying information held by QNEMU for the user to view on the terminal.

2.2.6.1 Prompt Menus

Prompt menus exist to indicate what type of input is required next. They contain a brief description of all the options available, plus helpful guidelines for the user. Abbreviations of the options and commands are also contained in the menu. There are two levels of prompt menus, aimed at the inexperienced user and the expert user respectively. The inexperienced user menu level prompts the user with full menus (see for example figures 2.6 and 2.7), while the more experienced user can restrict the display to the response and prompt lines (see figure 2.8). The response line is described in section 3.4, and the prompt line is a summary of the full menu and normally contains enough information for the more experienced user. To switch off the full menu the user enters the command SET MENLV with option BRIEF. Option FULL turns the full menu display back on.

The five regions in a prompt menu are <heading>, <prompt information>, <response>, <prompt> and <user input>, (see figure 2.5 for the general layout and figures 2.6 and 2.7 for actual examples).

figure 2.5 General Layout of a Prompt Menu

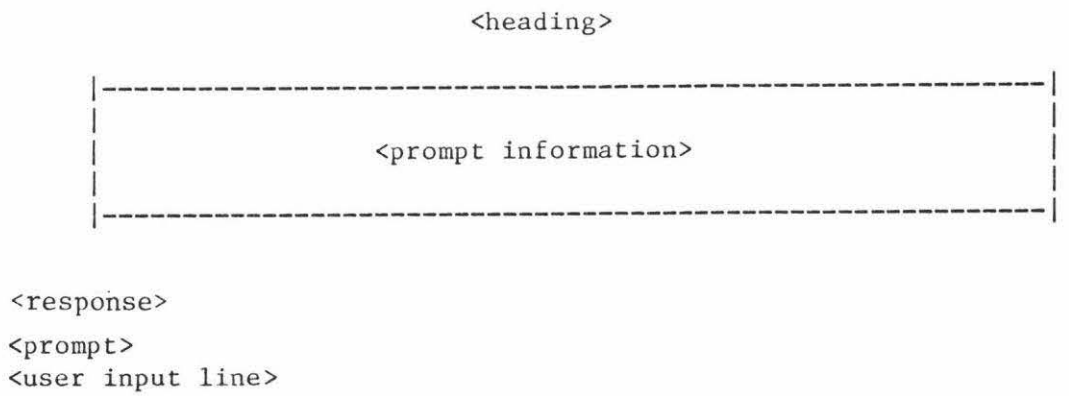


figure 2.6 Full Prompt Menu for the QNEMU Command

```
                Queueing Network Evaluator of Massey University.

|-------------------------------------------------------------|
|                                                             |
|   Command      Mnemonic              Command      Mnemonic  |
|   --------     ----------            ---------    ----------|
|   ANALYZE         A                    COPY          CO     |
|   CREATE          C                    DELETE        DEL    |
|   DIRECTORY       D                    EDIT          E      |
|   EXIT            EXIT               * HELP          H      |
|   LIST            L                    GET           G      |
|   RECOVER         REC                  INSERT        I      |
|   REMOVE          REM                  RETRIEVE      RET    |
|   SAVE            S                    SET           SET    |
|                                                             |
|     Current Network:          a null network.               |
|                                                             |
|-------------------------------------------------------------|

Status: thesis.ch4.dam file of networks has been read.

Key in ... QNEMU COMMAND,( e.g. HELP ) ... or M
```

figure 2.7 Full Prompt Menu for the Queueing Discipline

```
                Queueing Network Evaluator of Massey University.

|-------------------------------------------------------------|
|                                                             |
| Specification by the user of the QUEUEING DISCIPLINE at     |
| the node named diskone                                      |
|                                                             |
|  Queueing discipline                          Mnemonic      |
| ---------------------                        ----------     |
|   First come first served                       FCFS        |
|   Infinite server                               IS          |
|   Last come first served pre-emptive resume     LCFS        |
|   Processor sharing                             PS          |
|                                                             |
| * leaves unaltered                                          |
|                                                             |
|   L   :lists the current service details of the node.       |
|   Q   :leaves the entering of the service details.          |
|-------------------------------------------------------------|
Status: Editing the service details of the nodes.

Key in ... the QUEUEING DISCIPLINE ... L  ... * ... M ... or Q
```

figure 2.8 Brief Prompt Menu for the Queueing Discipline

Status: Editing the service details of the nodes.

Key in ... the QUEUEING DISCIPLINE ... L ... * ... M ... or Q

Prompt Menu By-pass

Prompt menus can be by-passed by the user typing in multiple options and commands on a single input line. The prompt menus are by-passed until either an error occurs (in which case the user must enter corrected input), or a display menu occurs (which requires the user to enter an acknowledgement once the information has been viewed). When either of these conditions occurs the rest of the input line is discarded.

2.2.6.2 Display Menus

The display menus supply information to the user on, among other things, the following:

- the names of the networks held in the library file using the DIRECTORY command
- help information from the HELP command
- the information from the LIST command

The general layout of the display menus is given in figures 2.9 and 2.10, with an example of the first layout in figure 2.11 and examples of the second layout in figures 2.12, 2.13 and 2.14.

The total number of lines that can be displayed is called a page. A system default exists for the page size, but this can be altered by the user via the SET command. If, for example a page size of 24 is used, the maximum physical size of the <screen of Text Information> field in figures 2.9 and 2.10 is 17 lines.

If the information can be displayed on a single page, it is displayed with the single line prompt for the next input required, as figure 2.11 shows.

figure 2.9 General Layout of Single Terminal Screen Display Menus

```
|-------------------------------------------------------------------------|
|                          <Heading information>                          |
|-------------------------------------------------------------------------|
|                                                                         |
|                       <Screen of Text Information>                      |
|                                                                         |
|-------------------------- t h e  e n d --------------------------------|
                     <QNEMU and display guideline>

<prompt>
<user input>
```

figure 2.10 General Layout of a Display Menu which is More than a Single Page

```
|-------------------------------------------------------------------------|
|                          <Heading information>                          |
|-------------------------------------------------------------------------|
|                                                                         |
|                       <Screen of Text Information>                      |
|                                                                         |
|-------------------------------------------------------------------------|
                     <QNEMU and display guideline>
Key in DISPLAY COMMAND,(i.e.H,B,P,T,N,D <int>,R,L <str>,Q to leave).
<user input>
```

figure 2.11 Display Menu of the Directory Command

```
|-------------------------------------------------------------------------|
| Saved network name        Date saved dd/mm/yy         Remove tag status |
|-------------------------------------------------------------------------|
|  IBMPC                        23/11/83                                  |
|  NET1                         11/12/83                                  |
|  PR1ME550                      9/10/83                                  |
|  NET1.BACKUP                  11/12/83          ** tagged for removal ** |
|-------------------------- t h e  e n d --------------------------------|
 QNEMU  fil:Xqnemu.dam              com:directory  lines    1-   4 of    4

Key in ... M to get the QNEMU menu ... or the QNEMU COMMAND.
```

If the amount of information cannot fit on a single page then the user may give one of a number of display subcommands which allows any portion of the information to be displayed. Typically a user would use the subcommands to "scroll" through the information, as the three figures 2.12, 2.13 and 2.14 show. The subcommands are:

B :The bottom command displays the bottom page of information.

F :The from <signed integer> command displays the page starting <signed integer> number of lines from the current line.

E.g. F -5 will start from 5 lines above the current line and F 5 will start from 5 lines below the current line.

H :The help command displays a page of display command descriptions.

L :The locate <string> command displays the first page containing <string>. The search starts from the current displayed line. Embedded blanks and case of the characters are significant.

E.g. L node name

N :The next command displays the next page of information, as does a <return>.

P :The previous command displays the previous page of information.

Q :The quit command leaves the display subcommand loop.

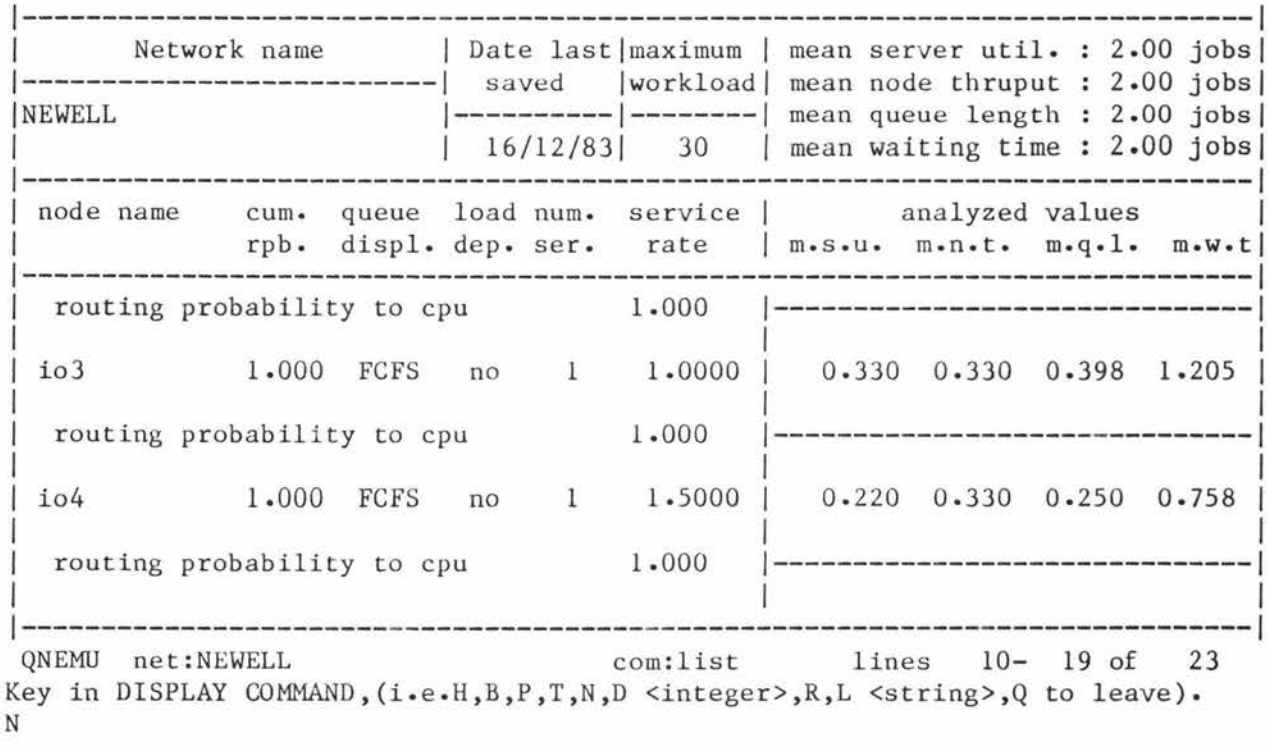
R :The repeat command carries out the last locate from the current top line, plus one.

T :The top command displays the first page of information.

figure 2.12 First Display Menu of the LIST of a 23 Line Network

```
|-------------------------------------------------------------------------|
|       Network name        | Date last|maximum | mean server util. : 2.00 jobs|
|---------------------------|  saved   |workload| mean node thruput : 2.00 jobs|
|NEWELL                     |----------|--------| mean queue length : 2.00 jobs|
|                           |  16/12/83|   30   | mean waiting time : 2.00 jobs|
|-------------------------------------------------------------------------|
| node name    cum.  queue  load num.  service |        analyzed values      |
|              rpb.  displ. dep. ser.   rate   | m.s.u.  m.n.t.  m.q.l.  m.w.t|
|-------------------------------------------------------------------------|
| cpu          1.000   PS    no    1    2.4390 |  0.589  1.436  0.804  0.560 |
|                                               |                             |
|  routing probability to io2          0.340    |-----------------------------|
|  routing probability to io3          0.230    |-----------------------------|
|  routing probability to io4          0.230    |-----------------------------|
|  routing probability to io5          0.200    |-----------------------------|
|                                               |                             |
| io2          1.000  FCFS   no    1    1.5000 |  0.325  0.488  0.391  0.802 |
|                                               |                             |
|  routing probability to cpu          1.000    |-----------------------------|
|-------------------------------------------------------------------------|
 QNEMU  net:NEWELL                    com:list       lines   1-  10 of   23
Key in DISPLAY COMMAND,(i.e.H,B,P,T,N,D <integer>,R,L <string>,Q to leave).
N
```

figure 2.13 Second Display Menu of the LIST of a 23 Line Network



```
|-------------------------------------------------------------------------|
|       Network name        | Date last|maximum | mean server util. : 2.00 jobs|
|---------------------------|  saved   |workload| mean node thruput : 2.00 jobs|
|NEWELL                     |----------|--------| mean queue length : 2.00 jobs|
|                           |  16/12/83|   30   | mean waiting time : 2.00 jobs|
|-------------------------------------------------------------------------|
| node name    cum.  queue  load num.  service |        analyzed values      |
|              rpb.  displ. dep. ser.   rate    | m.s.u.  m.n.t.  m.q.l.  m.w.t|
|-------------------------------------------------------------------------|
|  routing probability to cpu          1.000   |-----------------------------|
|                                              |                             |
| io3          1.000  FCFS   no    1    1.0000 |  0.330  0.330  0.398  1.205 |
|                                              |                             |
|  routing probability to cpu          1.000   |-----------------------------|
|                                              |                             |
| io4          1.000  FCFS   no    1    1.5000 |  0.220  0.330  0.250  0.758 |
|                                              |                             |
|  routing probability to cpu          1.000   |-----------------------------|
|                                              |                             |
|-------------------------------------------------------------------------|
 QNEMU  net:NEWELL                    com:list       lines  10-  19 of  23
Key in DISPLAY COMMAND,(i.e.H,B,P,T,N,D <integer>,R,L <string>,Q to leave).
N
```

figure 2.14 Third Display Menu of the LIST of a 23 Line Network

```
|-------------------------------------------------------------------------|
|       Network name        | Date last|maximum | mean server util. : 2.00 jobs|
|---------------------------|  saved   |workload| mean node thruput : 2.00 jobs|
|NEWELL                     |----------|--------| mean queue length : 2.00 jobs|
|                           |  16/12/83|   30   | mean waiting time : 2.00 jobs|
|-------------------------------------------------------------------------|
| node name    cum.  queue  load num.  service |        analyzed values      |
|              rpb.  displ. dep. ser.   rate    | m.s.u.  m.n.t.  m.q.l.  m.w.t|
|-------------------------------------------------------------------------|
|  routing probability to cpu          1.000   |-----------------------------|
|                                              |                             |
| io4          1.000  FCFS   no    1   1.5000  |  0.220  0.330  0.250  0.758 |
|                                              |                             |
|  routing probability to cpu          1.000   |-----------------------------|
|                                              |                             |
| io5          1.000  FCFS   no    1   2.0000  |  0.144  0.287  0.156  0.545 |
|                                              |                             |
|  routing probability to cpu          1.000   |-----------------------------|
|                                              |                             |
|------------------------- t h e  e n d ---------------------------------|
 QNEMU  net:NEWELL                  com:list       lines  14-  23 of  23
Key in DISPLAY COMMAND,(i.e.H,B,P,T,N,D <integer>,R,L <string>,Q to leave).
Q
```

2.2.7 QNEMU Responses

There are three levels at which QNEMU responds to the user: status messages, warnings, and errors (listed in increasing order of severity).

2.2.7.1 Status

A status response is purely to inform the user of the current command being executed, or the last activity which QNEMU carried out. Below are listed example status responses.

Status: The altered current network is tagged for removal.

Status: Analyzed the statistics of the current network.

Status: NetworkOS360 has been tagged for removal.

2.2.7.2 Warning

A warning response is used to indicate to the user that the syntax of the input is correct, but that QNEMU is unable to carry out the desired activity. For example, if the user attempts to list the directory of saved networks in the library file and none exist, the following warning is displayed:

Warning: There are no saved networks in the library file.

Another example would be if there are no nodes in the current network and the user attempts to edit the node names, or other node information, then the following warning is issued to the

user:

Warning: There are no nodes in the network to edit.

2.2.7.3 Error

An error response indicates that an invalid command or option has been entered. This means the input syntax is invalid and should be retyped. There are also a number of system errors which can occur which are not the fault of the user. The error responses also cause an audible beep at the terminal to remind the user to re-enter the input. Below are listed two of the most frequently met error responses!

When an invalid QNEMU command is entered the response is:

* Error: an invalid QNEMU command was entered.

When the user attempts to exit QNEMU after the current network has been changed but without saving or removing the current network, the response is:

* Error: please Save or Remove the current network.

3 IMPLEMENTATION

3.1 Introduction

QNEMU has been implemented with the portability of the package as a prime goal. One of the early decisions made before this thesis was started was in the use of FORTRAN 77 as the implementation language of the underlying computational algorithms [Oldf 82]. Another earlier decision was to use the NAG library [NAG 83] to provide two of the subroutines used in the calculation of the normalisation constant, G. Both these decisions were based on the general wide availability of both FORTRAN 77 and the NAG library. It would be a relatively simple task to replace the NAG library routines by "in-built" routines if this was required to improve portability.

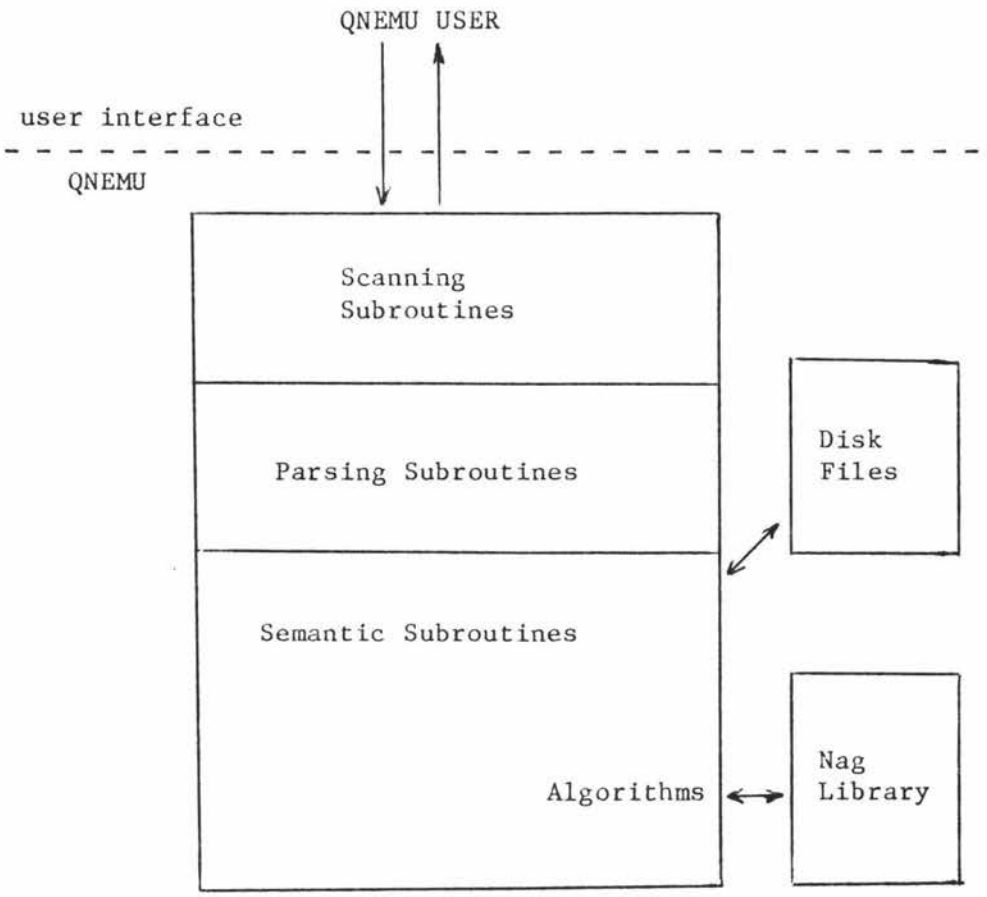
The majority of the QNEMU package consists of the interface and network handling routines, with the algorithms making up only one fifth of the code. In all, QNEMU comprises approximately 5000 lines of source code. Structured programming techniques were followed throughout the coding and separate subroutines were used whenever the clarity of the code would be improved by so doing.

The subroutines in QNEMU can be divided into three groups which reflect the progress of a QNEMU session, namely:

- initialisation subroutine
- interface subroutines
- termination subroutine

The implementation description in the following sections follows this ordering, with the details of the interface implementation being further divided into a number of sub-sections that reflect the structure described in figure 3.1.

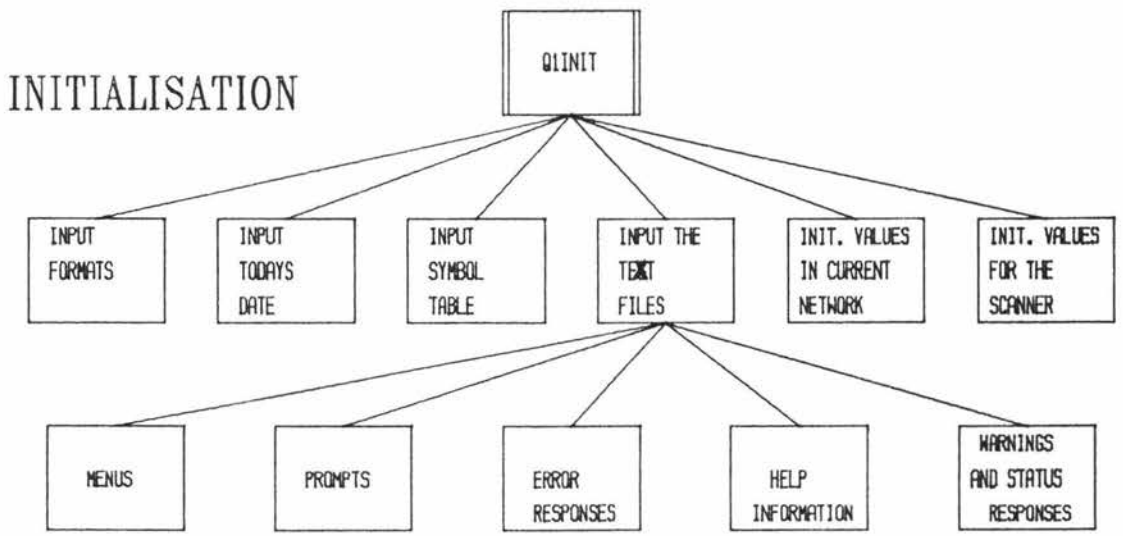
figure 3.1 Structure of the QNEMU Interface Software



3.2 Initialisation

Initialisation takes place in the subroutine Qlinit and involves a number of steps, as shown in the structure diagram in figure 3.2.

figure 3.2 Structure Diagram of the Subroutine Qlinit



3.2.1 I/O Formats

The FORTRAN I/O formats are stored in a text file and read into QNEMU on each run. This enables alterations to the formats to be made very simply without the need to recompile the program. An example use of the formats is:

```
write (termun, fmt(19)) prompt
```

where fmt(19) = '(A79)'.

3.2.2 Date

The date in QNEMU is obtained from a FORTRAN 77 extension provided in Salford F77 [SALF 83]. To keep QNEMU as portable as possible, code is also present to allow the date to be entered by the user. This requires recompiling the subroutine Qldate with a call to the subroutine that requests the user to enter the date.

3.2.3 Command Keywords and their Options

The command keywords and the command options are held as an ordered list within QNEMU, and are read in from a disk file during initialisation. Figure 3.3 describes a number of the entries in the current reserved word disk file.

figure 3.3 Section of the Disk File containing the Symbol Table

Reserved Word	Symbol Value
...	
BRIEF	68
C	44
CA	16
CAL	16
CALC	16
CALCULATE	16
CO	53
COM	45
COMMAND	45
COMMANDS	45
COPY	53
...	

To add a reserved word, all that is required is the insertion of a new line in the file in the correct position (ordered alphabetically), with the symbol value of the keyword that it represents. Following this 1 must be added to the count of lines in the text file (contained in the first record of the file) for each reserved word added. For example, to add EVALUATE as a synonym for CALCULATE, the line

EVALUATE 16

would be inserted in alphabetical within the file, and the count increased. EVALUATE and CALCULATE are accepted as synonyms as they have the same symbol value.

3.2.4 Text files

All the menus, prompts, responses, help information and menu information are held in sequential access disk files and read into data structures held in FORTRAN common blocks. The decision to hold these details in internal storage was taken to provide a faster response time, compared to reading sections from a disk file as required. This may not be much faster when using a virtual memory system, due to the effect of paging, and if the system has a lack of internal storage then QNEMU could be modified to read in sections of the text files as required. As for command keywords, using text files allows modifications to be easily made without the need to recompile any source code.

The menu text file and help text file differ from the other text files, as they both have entries which are made up of more than a single line of text. In the menu text file there are a number of multiple line prompt menus, and in the help text file there are groups of text lines which make up the different ranges of help

information available to the user. All the text files contain the number of text lines within the file as a three digit number in the first record of the file. Figure 3.4 describes the structure of the D-MENU.F77 file, which is similar to the D-HELP.F77 file described in figure 3.5.

In figure 3.4 the Display Menu Lines immediately following the first record contain the headings of the display menus. Following these are the menus, separated by special flag records containing the character ^ in the first column. All the lines of the D-MENU.F77 file, except the flag records, are read into the menlin array during initialisation. When a flag record is encountered, the current line number is stored in the menptr array, this then points at the first line of the menu in the menlin array. The pointer at the first line of the next menu, minus 1, will be the last line of the previous menu.

figure 3.4 Layout of D-MENU.F77 File

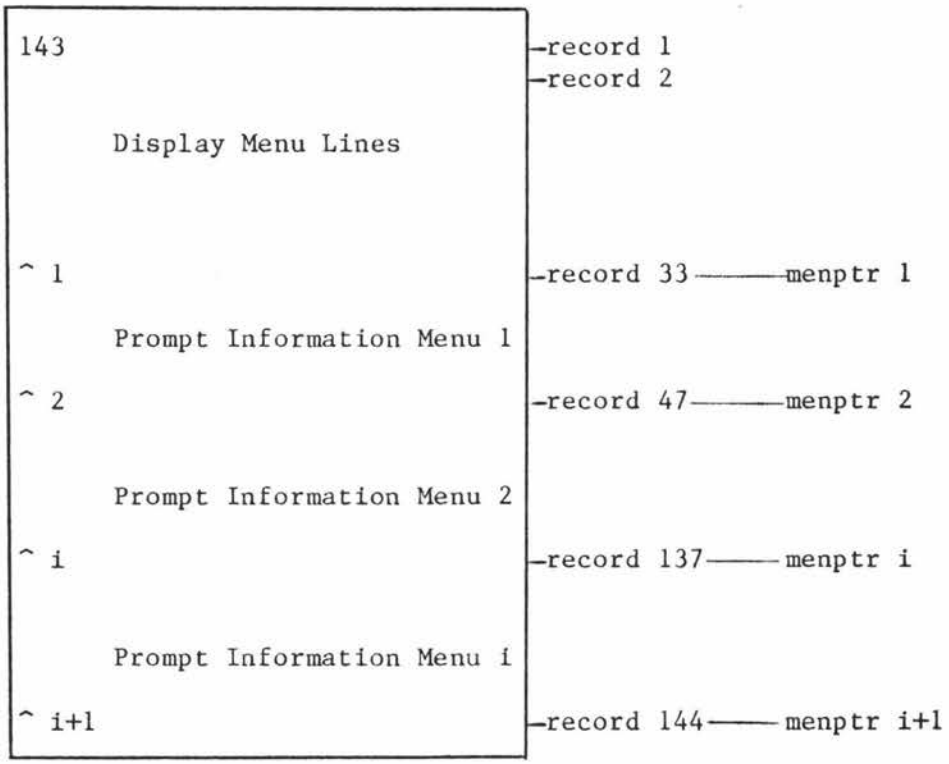
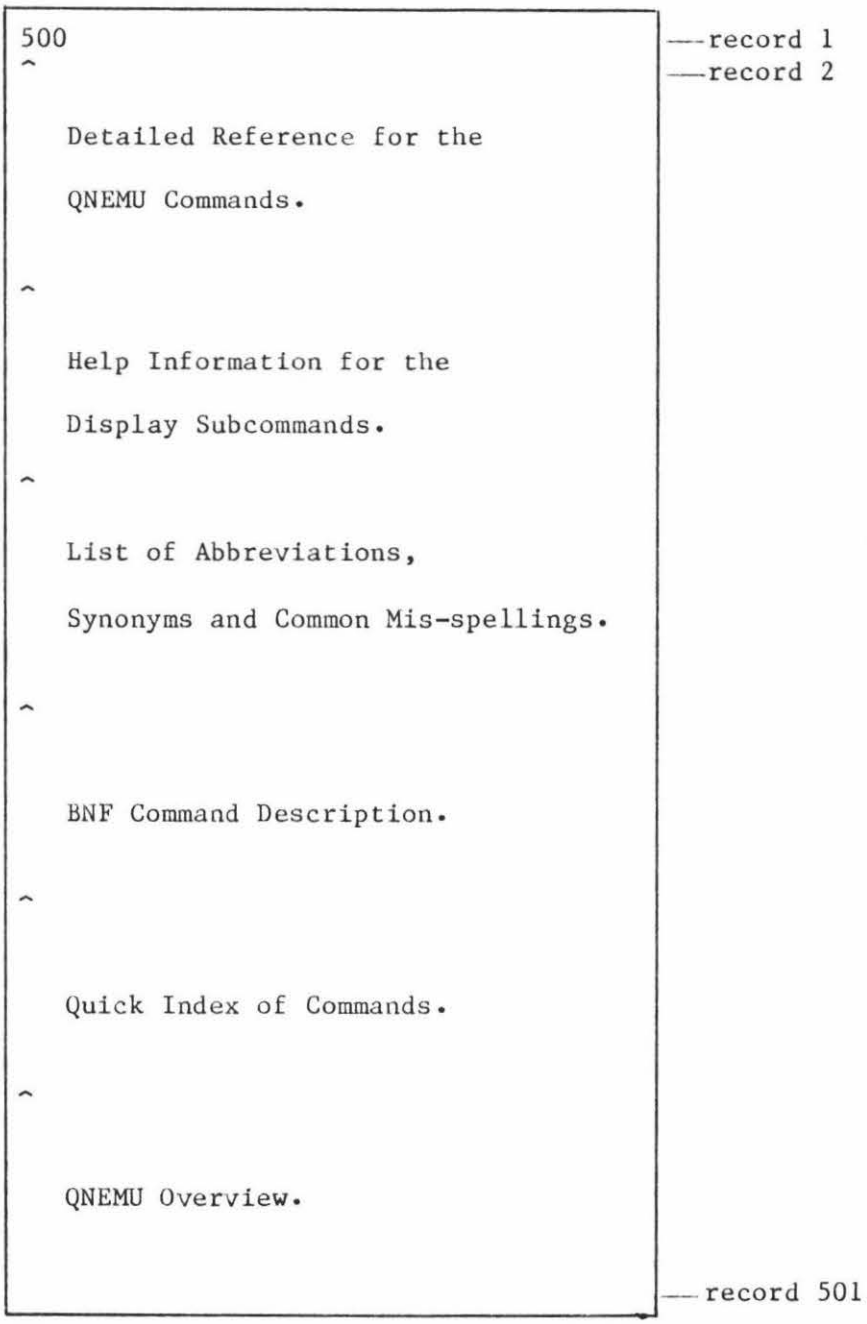


figure 3.5 Layout of D-HELP.F77 File



3.2.5 Current Network

The initialisation for the current network results in the setting up of null network values. These include a blank current network name and a zero node count. This allows the user to get a saved network or create a new network at the commencement of a new session.

3.2.6 Scanner

The defaults are set so that full prompt menus are displayed and a 24 line terminal screen is assumed.

3.3 Help

The on-line help facilities are provided as text files which are viewed using the display menus described in section 2.2.6.2. The D-HELP.F77 text file is read into an array by Qlinit (see figure 3.2).

There are two basic points at which help information can be obtained in QNEMU. The first is if the user enters the QNEMU HELP command; depending on the help option entered, the required information is loaded into the array used for the display menus and is then made available for the user to view. The second point at which help information can be obtained is during the entry of any of the QNEMU commands. When entering a QNEMU command, the user can enter the character H and the first page of the help information pertaining to that command will be displayed. This is implemented by keeping the current QNEMU command in a global variable. If an H is entered, the complete help reference information is loaded and the uppercased command located. As the first occurrence of the uppercase command is always in the first line of the help information for that command, the first page associated with that command is displayed.

Figures 3.7 and 3.8 give examples of two types of help information. The different help options are displayed in the menu given in figure 3.6.

figure 3.6 The Prompt Menu for the Help Command Option

```
                Queueing Network Evaluator of Massey University.

|-------------------------------------------------------------|
|                                                             |
| Specification of the HELP OPTION for the different text     |
| files of useful information.                                |
|                                                             |
| help option                                  mnemonic       |
|-------------                                 ----------     |
|                                                             |
| displays the BNF description                    BNF        |
| displays a full description of commands         COM        |
| displays a brief QNEMU description              QNE        |
| displays command synonyms and abbrev.           SYN        |
| displays a single line command description *    WAL        |
| displays the <string> if found by a Locate      <string>   |
|   in the full description of commands                       |
|-------------------------------------------------------------|

Status: Help command.

Key in ... the HELP OPTION, ... M ... or *
```

figure 3.7 Display Menu of The SYN Option of the HELP Command.

Qnemu Command	Synonym Commands		Abbreviations and Common Misspellings			
ADD	INSERT		AD			
ANALYZE	CALCULATE	STATISTICS	A	AN	ANAL	ANALYSE
CALCULATE	ANALYZE	STATISTICS	CA	CAL	CALC	
CREATE			C	CR	CRE	CREAT
DELETE			DEL	DELET		
DIRECTORY	LIBRARY		D	DIR		
DISPLAY	LIST	FIND	DIS	DISP		
EDIT	UPDATE		E	ED	EDI	
EXIT	STOP	TERMINATE	none			
FIND	LIST	DISPLAY	FI	FIN		
GET	LOAD		G	GE		
HELP			H	HE	HEL	
INSERT	ADD		I	IN	INS	
LIBRARY	DIRECTORY		LIB	LIBRAY		
LIST	DISPLAY	FIND	L	LI	LIS	
LOAD	GET		LO			
RECOVER			REC			
REMOVE			REM			
RETRIEVE			RET	RETREIVE		
SAVE			S	SA	SAV	
STATISTICS	ANALYZE	CALCULATE	ST	STA	STAT	STATS
STOP	TERMINATE	EXIT	none			
TERMINATE	EXIT	STOP	none			

t h e e n d

QNEMU net: com:help lines 1- 23 of 23

Key in ... M to get the QNEMU menu ... or the QNEMU COMMAND.

figure 3.8 Display Menu of the Wallpaper Option of the HELP Command

QNEMU Command	Single Line Description
ADD	inserts a new network node into the current network.
ANALYZE	calculates the performance statistics of the current network.
CALCULATE	calculates the performance statistics of the current network.
CREATE	initialises the current network to default values and names it.
DELETE	deletes nodes or the routing probabilites of nodes.
DIRECTORY	lists the directory of the networks currently saved.
DISPLAY	lists to the terminal details of the current network.
EDIT	changes the nodes and workload of the current network.
EXIT	stops execution of QNEMU.
FIND	gets the named network as the current network.
GET	gets the named network as the current network.
HELP	lists helpful information to the terminal.
INSERT	inserts a new network node into the current network.
LIBRARY	lists the directory of the networks currently saved.
LIST	lists to the terminal details of the current network.
LOAD	gets the named network as the current network.
RECOVER	recovers the tagged for removal current network.
REMOVE	tag or, if tagged, permanently removes current or named network
RETRIEVE	retrieves the tagged network in the library to untagged.
SAVE	saves the current network under the name given or its name.
STATISTICS	calculates the performance statistics of the current network.
STOP	stops execution of QNEMU.
TERMINATE	stops execution of QNEMU.

t h e e n d

QNEMU net: com:help lines 1- 23 of 23

Key in ... M to get the QNEMU menu ... or the QNEMU COMMAND.

3.4 Prompt Scanner

The prompt scanner has two main functions which both deal directly with the input of the QNEMU commands. The scanner prints prompting information to the terminal and then scans the user input checking its validity.

The prompt scanner is implemented as the getsym subroutine (see figure 3.9) and in essence is the input interface between the user and QNEMU. There are two distinct levels of interaction in getsym. The first is between getsym and the user, and the second is between getsym and the high level parsing subroutines (see figure 3.10).

figure 3.9 Structure Diagram of the Prompt Scanner

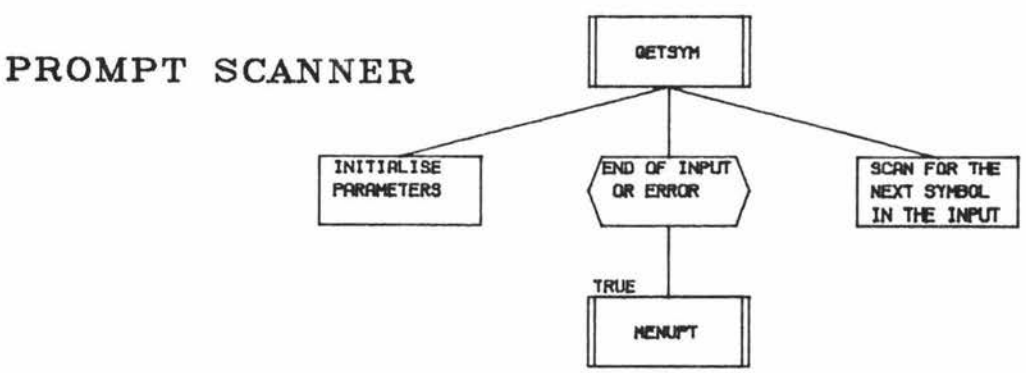
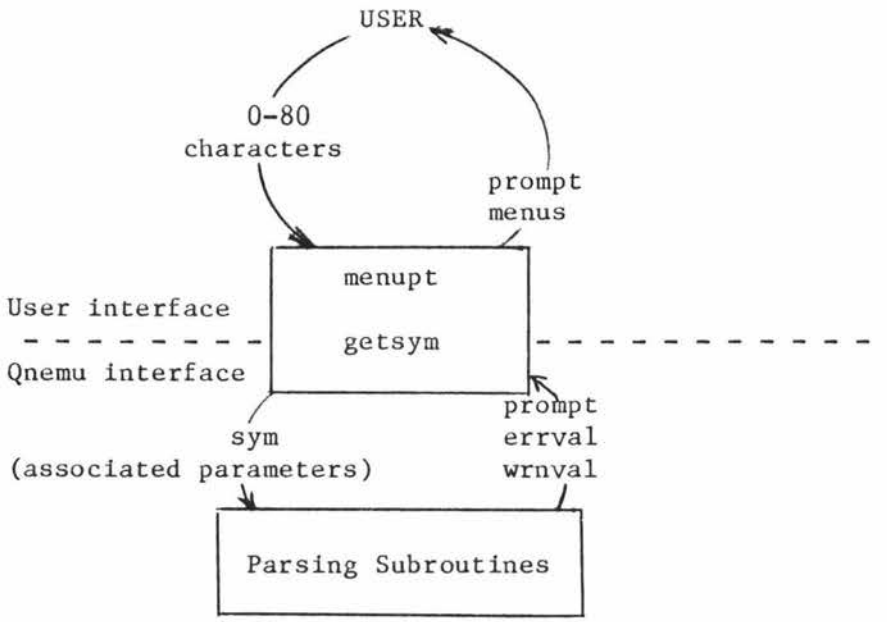


figure 3.10 Levels of Interaction in the Subroutine Getsym



A call on the subroutine menupt is made by getsym whenever the underlying subroutines require a new symbol and the current symbol stream cannot supply it. This happens if there are no more symbols in the input stream, or alternatively a symbol which cannot be parsed is currently the next in the symbol stream. The second alternative produces an error message in the response line of the prompt, and is indicated to the scanner by the parsing subroutine giving the errval parameter the value of the index into the array errlin, which contains the relevant error response. The wrnval, parameter is also used to specify the index into the array wrnlin which contains the relevant status or warning response for the current state of QNEMU.

An 80 character line is read in from the terminal. Getsym appends a semicolon to the line which the scanner will detect as an end-of line character, following which getsym scans for the first symbol in the the line.

If the first symbol found is a string, the reserved word table is searched, the list of node names in the current network is searched, and the list of network names in the directory is searched. A string can be used as both a node name and a network name but neither a node name or network name can be a reserved word. The QNEMU program can determine from the context of the command whether the string found by the scanner should be a node name or network name, and parse it accordingly.

Table 3.11 is a description of the meaning of the getsym parameters, depending on the value returned in the sym parameter.

Table 3.11 Getsym Parameter Table

Sym	Parameter	Description
intsym	intvax:	integer value
	relval:	real (integer value)
relsym	intvax:	int (real value)
	relval:	real value
strsym	intvax:	number of characters in the string
	intval:	position of node if nodfnd is true
	intva2:	position of network if netfnd is true
	nodfnd:	true if string is a node in the current network
	netfnd:	true if the string is a network in the directory
	strval:	the string
	strvau:	uppercased string
	strvax:	lowercased string

otherwise (the parameters have no meaning).

3.5 Command Language

The QNEMU command language is made up of QNEMU commands and options for specifying the commands which are collectively called the QNEMU keywords for ease of exposition. The actual syntax of the QNEMU command language can be found in Appendix A. Abbreviations for all the keywords except the keywords for finishing executing the QNEMU package are provided. An abbreviation for the EXIT command would possibly let the user exit by mistake. In most prompts an * can be entered which will default to one of the most common input options for that prompt. Most abbreviations are simply made up of the first few letters of the keyword (e.g. A, AN, ANAL for ANALYZE). Common mis-spellings or mis-keying for keywords are also used in QNEMU

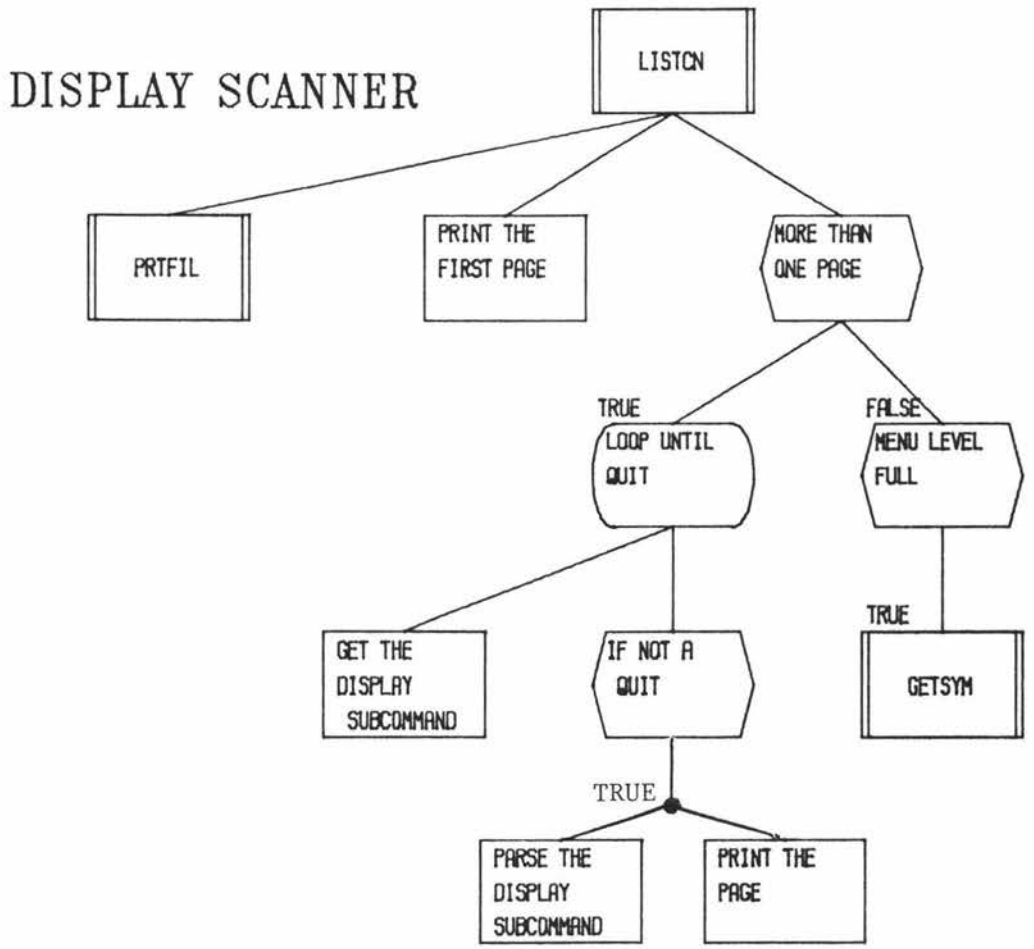
(e.g.LIST or LSIT, ANALYZE or ANALYSE) and are included in the symbol table.

3.6 Display Scanner

The display scanner, subroutine listcn, is called by a number of the semantic routines to print information to the screen.

The subroutine listcn (see figure 3.12) gets the information to be displayed by calling subroutine prtfil which passes back the information in the text file fdarry. The first page, which consists of a number of contiguous lines, is displayed to the user. If the amount of information is greater than fits in a single page, the user can enter one of a number of display subcommands which allows the user to shift the display page within the information array fdarry (see section 2.2.6.2). If the amount of information is less than fits in a page, and the menu level is full, then getsym is called with the prompt of the next symbol expected.

figure 3.12 Structure Diagram of the Subroutine Listcn



3.7 Parsing

The parsing in QNEMU is carried out by a large number of subroutines which all form a tree structure, with the main program as the root and a subroutine for each of the commands. The parsing is handled by the command subroutines and the underlying subroutines that they call. Any subroutines which call the subroutine getsym can be assumed to be carrying out some parsing within QNEMU.

The output from the scanner is a value assigned to the sym parameter and the other associated parameters (see table 3.11).

The syntax of the QNEMU command language is described as tramline diagrams in Appendix A. When parsing a command there are three symbols not contained in the syntax diagrams, but which are always valid, namely:

- crsym i.e. the carriage return character
- msym i.e. the character M
- hsym i.e. the character H

An msym will cause the full prompt menu to be printed for the current prompt. This is extremely useful when the menu level has been set to brief and the user requires some extra prompting information.

An hsym input at any stage of entering a command will display to the user the help information contained in the help file on the QNEMU

command with the first displayed page being that of the current command being parsed.

A qsym, Q, is a valid symbol at any stage once command analysis has been entered. If entered it causes the parser to exit one level of the parse. A number of commands have a number of levels and therefore up to three consecutive qsyms may be needed to reach the main QNEMU command level.

An lsym, L, is also a valid symbol in a number of places within a parse. The lsym causes the parse to be suspended while relevant details to the current parse are displayed to the user. For example, when entering the name of a node to insert in the current network an lsym will list the nodes in the current network to the user. If an lsym is a valid entry it will be shown in the prompt.

When parsing, any invalid symbol causes an error response indicating the reason for the error.

QNEMU uses parameters declared in the common blocks to represent the keywords. For example:

Command	Parameter name	Parameter value
Analyze	asym	16
List	lsym	18

The parsing code in QNEMU can then be written as:

```
else if (sym .eq. asym) then
     . . .
else if(sym .eq. dsym) then
     . . .
else if(sym .eq. lsym) then
     . . .
```

This enables the abbreviations, synonyms and common mis-spellings to be transparent to all but the scanner.

3.8 Library File I/O

The library file I/O is handled by four separate subroutines: savecn, loadcn, getdir and savdir. The library file is a direct access file, with all I/O being unformatted read and writes for increased efficiency. The record size is 64 characters as this is the smallest integral division of the 2048 character blocks used on the Prime 750 system which fits the largest record used in QNEMU (the node header record).To obtain efficient use of the storage, the routing probabilities are stored as a sparse matrix. (See appendix B for the record formats used in the library file).

The library file can be divided into two main sections (see figure 3.13).

The Directory section consists of a single Directory Header Record, followed by a number of alphabetically ordered Directory Entry Records, one for each saved network in the library file.

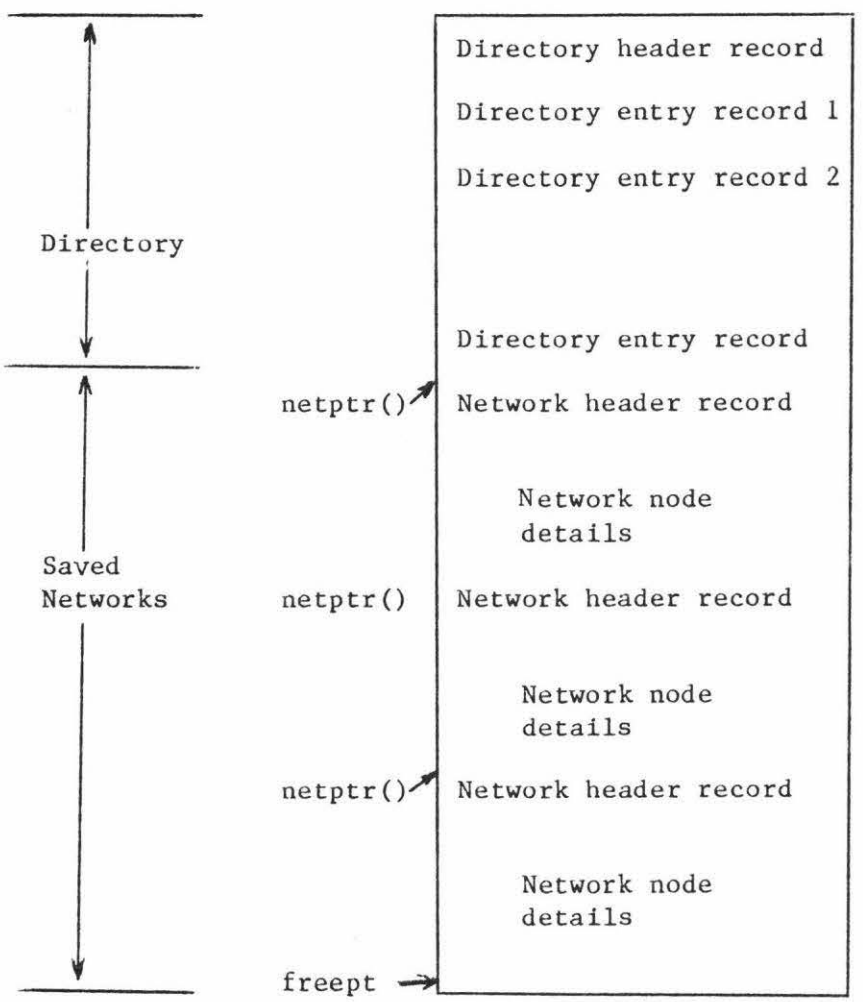
The Saved Networks section consists of the same number of saved networks as there are Directory Entry Records. The networks are found in FIFO order. Freept always points at the last used record in the library file.

Each network consists of a Network Header Record followed by a number of Node Header Records and Node RPB Records. Each node has one Header Record and, depending on the number of routing probabilities, zero or

more RPB records following the next node's header record. The next node's header record is positioned before the previous node RPB records as the number of RPB records for a node can vary from 0 upwards and the pointer to the next node header record is contained in the previous node's header record (see figure 3.14 and Appendix B).

An arbitrary restriction of 100 saved networks is placed on each library file.

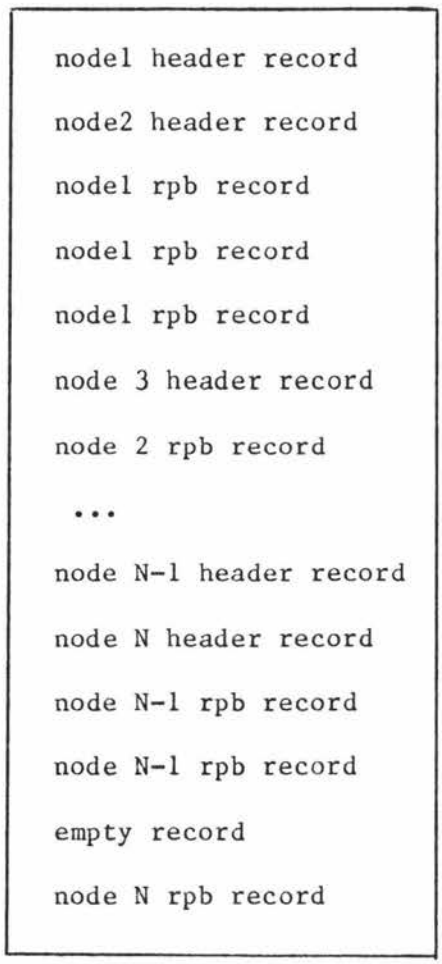
figure 3.13 Layout of the network Library File



3.9 Network Node Details

The network node details section of each saved network has the layout shown in figure 3.14 (the formats of the records can be found in Appendix B).

figure 3.14 Layout of the Node Details Section for an N Node Network



3.10 Algorithms

The algorithms used in QNEMU had been implemented by an earlier researcher [Oldf 82]. These algorithms required validation and modification for use in the final package. They were based on the algorithms described in [Brue 80]

There were a number of minor modifications made which did not impact on the results obtained, but which were necessary to simplify the interface with the rest of the package. In addition it was found that for a number of cases the algorithms produced incorrect results. Below are the corrections that were made to the algorithms to handle these incorrect results.

After testing the algorithms it was found that the mean server utilisations for load dependent nodes were in error, but the calculations of the mean node throughput produced results which were correct. The corrections made to the algorithms used the following Operational Analysis formula to calculate the mean server utilisations from the mean node throughputs [Denn 78] for load dependent nodes:

For load dependent nodes the calculation used is:

$$\text{server utilisation} = \frac{\text{mean node throughput}}{\text{number of servers x mean service rate}}$$

In addition the mean server utilisations at infinite server nodes was not provided for in the algorithms, therefore the following calculation was used for this type of node:

$$\text{server utilisation} = \frac{\text{mean mode throughput}}{\text{number of jobs in the workload} \times \text{mean service rate}}$$

4 VALIDATION of QNEMU

4.1 Introduction

The validation of QNEMU that has been carried out to date has been against published results obtained from other modelling analyses. The main sources of the published data used are [Chiu 75, Saue 81, Bask 75 and Prit 74]. In general only the server utilisations were analyzed in the published papers, but given that the theory used in the algorithms is correct the other statistics which QNEMU calculates can also be validated. The level at which roundoff errors will affect the results has not been fully determined, but further research in this area is being carried out by members of the Performance and Evaluation Group (PEG) within the Department of Computer Science at Massey University.

The results from QNEMU can be checked by a number of simple calculations, which will detect any inconsistencies in the four performance measures calculated by QNEMU (see appendix C).

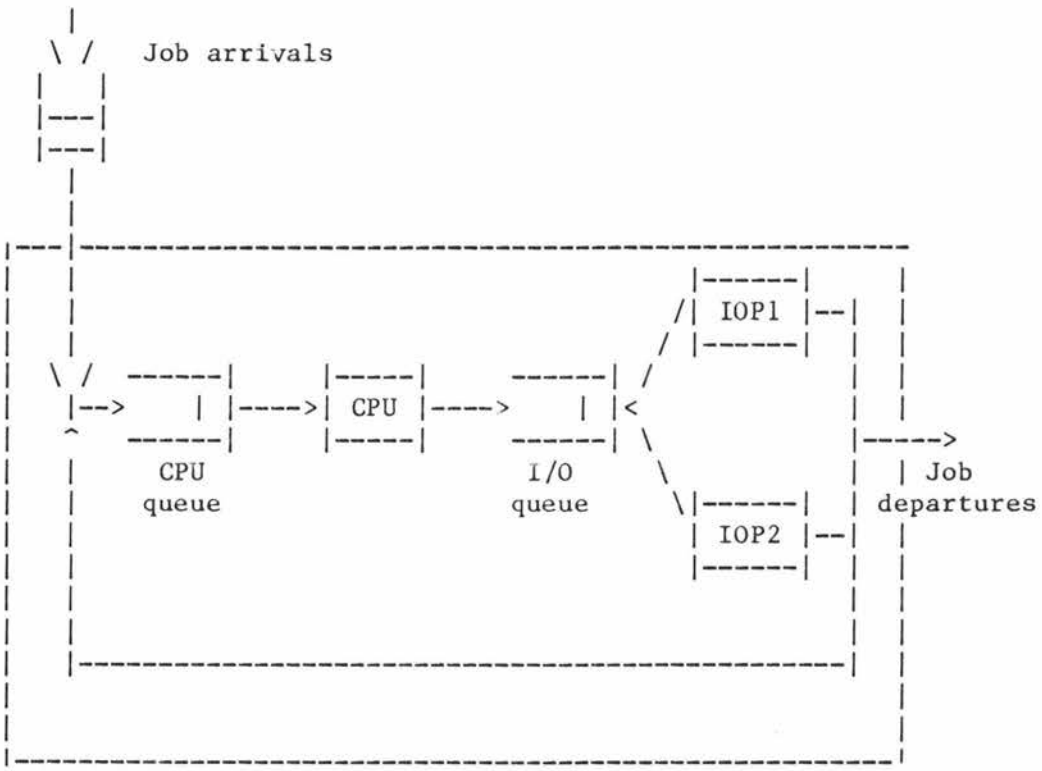
4.2 Validation Exercise 1

4.2.1 Introduction

The first validation exercise uses the reported results obtained from the monitoring of an actual system [Chiu 75]. The monitored data was used by the authors to validate two queueing models. The results in the paper have been reproduced by QNEMU to a very close degree of agreement. In addition to the paper by Chiu et al., the same system was modelled by Sauer, who obtained an additional result suitable for validation of the QNEMU package [Sauer 81].

The system modelled was the IBM System/360-75 at the University of California, Santa Barbara (UCSB). The system consisted of 512 kilobytes of high-speed core, two megabytes of bulk core, two high-speed selector channels, and a multiplexor channel. Auxilary storage consisted of two IBM 2314's on each selector channel, and various peripherals on the multiplexor channel. Figure 4.1 represents the system as an open queueing network.

figure 4.1 Job Flow in the IBM System/360-75 of UCSB



4.2.2 The Models

In the Chiu et al. paper there are two different models of closed systems: the Cyclic Queue Model (CQM) and the Central Server Model (CSM). The characteristics of the CQM model are similar to the machine repairman model described in [Koen 60]. The CSM model differs from the CQM model in that each I/O processor has a separate queue with its own probability of jobs leaving the CPU entering that queue (see figures 4.2 and 4.3). The CSM is discussed in Appendix H, section H.7.1. For ease of exposition CQM

and CSM are used when referring to the models in Chiu et al. and CQM* and CSM* are used when referring to the QNEMU models.

figure 4.2 Cyclic Queue Model [Chiu 75]

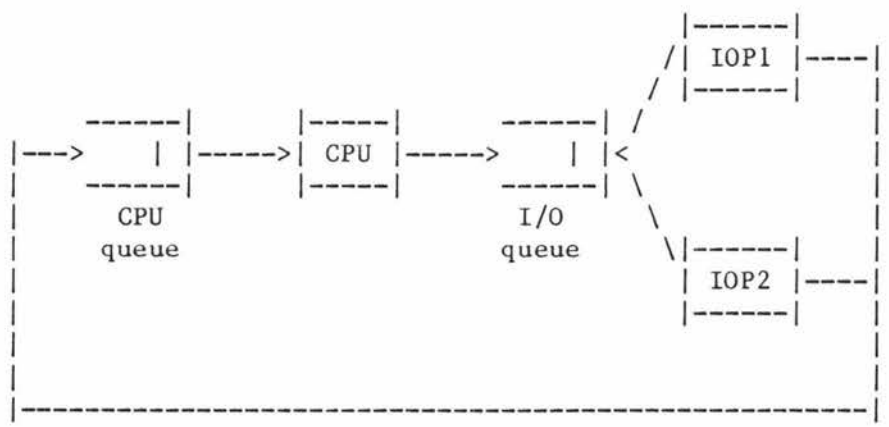
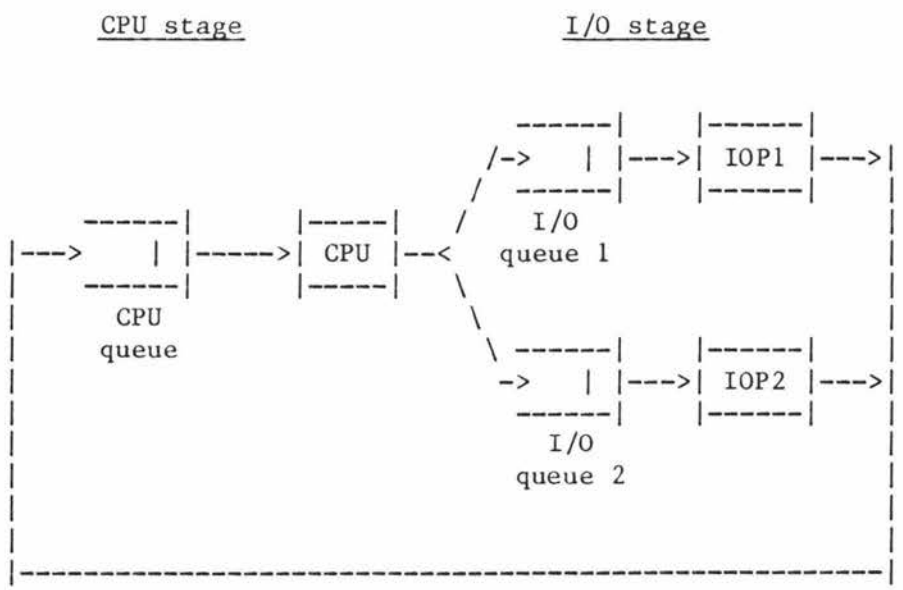


figure 4.3 Central Server Model [Chiu 75]



There are a number of simplifying approximations and assumptions which have had to be made to permit the models to be solvable. Even though these may seem gross approximations to the real system the CQM, CSM, CQM* and CSM* models are reasonably accurate.

One of the simplifying assumptions made is the exclusion of the activity of the multiplexor channel from the models, as the processing is completely overlapped with the CPU processing and requires very little CPU time.

In reality there is a job flow in the system with jobs arriving at the input queue, being processed, and leaving the system. However the models assume that there is an infinite supply of jobs, which enables the models to be closed with a fixed number of circulating jobs. A job in execution can be characterized by an alternating series of CPU requests and I/O service requests until completed. The degree of multiprogramming can vary with dynamic partitioning (OS/MVT); however an approximation is used in the models from data collected from the IBM System Management Facility (SMF), for the (fixed) average multiprogramming level, N.

The CQM and CQM* models use a single I/O queue for the two I/O processors instead of a queue for each processor as the CSM and CSM* models use. This simplifying assumption is used as the resource allocator of the system tends to balance the load on the two I/O processors. In the models, I/O operations are not allowed to overlap the CPU processing even though the system does have

facilities to handle it. This assumption is supported by the behaviour reported on a similar system in [Full 71].

Both the compute-time distribution of the CPU and the I/O-time distribution are approximated by the negative-exponential distribution. The means were derived from system measurements. The system is, in fact, better modelled by hyper-exponential compute-times; modelling using a negative-exponential should therefore give slightly pessimistic results.

Both the CPU and I/O queues are modelled using a FCFS service discipline. The actual scheduling is preemptive-priority but is closely approximated by a FCFS scheduling discipline with a negative-exponential distribution. The I/O queues are also modelled with FCFS. The negative-exponential distribution is a poor approximation to the measured service times but is easily handled in the models.

4.2.3 Experiments

Chiu et al. performed a number of experiments. The results obtained from the two models CQM and CSM were compared with the values actually measured on the system. For each experiment the CQM* and CSM* model results were also obtained and compared with both the measured values and the modelled values from Chiu et al..

Experiment 1

The first experiment they carried out involved the use of a controlled workload, consisting of fifty jobs chosen to be representative of the daily batch workload. During this experiment the time-sharing system was disabled so as to provide a reproducible environment. The experiment involved the modelling and measuring of the system before and after slow 8 microsecond bulk core (LCS) was replaced with 1.8 microsecond core (ECM).

The first run was carried out and the measured results in the LCS column in table 4.4 were obtained. The core was then replaced and the measured results in the ECM column were obtained.

The two models, CQM* and CSM* as shown in figures 4.2 and 4.3 were entered into QNEMU along with the routing probabilities and service rates given in table 4.4. These models were then analyzed with a workload of 2.2 jobs, as that was the average multiprogramming level obtained from SMF for the run. As can be seen from table 4.4 the CPU utilisations for the CQM and CQM* models are slightly higher than the measured values and the CPU utilisations for the CSM and CSM* models slightly lower. A similar result is obtained for the average server utilisation at the I/O combined device and I/O utilisations of each I/O device.

In general the CSM and CSM* models will produce an utilisation figure less than the measured value, as the two queues mean that for a proportion of the time when multiple jobs are in the system, jobs will queue at a single I/O device while the other is free. The CQM and CQM* models obviously do not suffer from this, as they have only a single queue with dual servers.

table 4.4 Performance Figures Before and After Memory Upgrade

	LCS			ECM		
	measured	CQM*	CSM*	measured	CQM*	CSM*
CPU service rate	34.84	34.843	34.843	49.75	49.75	49.75
I/O service rate	21.41	21.41	21.41	21.41	21.41	21.41
Routing probability 1	0.419	-	0.420	0.523	-	0.523
Routing probability 2	0.581	-	0.580	0.477	-	0.477
CPU utilisation	0.677	0.689	0.589	0.567	0.573	0.468
I/O combined utilisation	0.551	0.561	0.480	0.657	0.666	0.544
I/O 1 utilisation	-	-	0.403	-	-	0.569
I/O 2 utilisation	-	-	0.556	-	-	0.519
Throughput	23.6	24.009	20.533	28.1	28.5	22.18

Experiment 2

The next experiment performed by Chiu et al. varied the number of initiators in the system during the observation period. The workload was the fixed sample of 50 jobs used in the first experiment, and the ECM memory was used. Table 4.5 gives both the measured and model results for CPU utilisation. Considering the assumptions for the QNEMU models the results are extremely accurate.

table 4.5 Comparison of Measured and Modelled Results

Workload	Service Rate		CPU Utilisation				
			Modelled				Meas.
	CPU	I/O	CQM	CQM*	CSM	CSM*	
1.00	22.27	28.41	0.560	0.560	0.560	0.560	0.60
1.93	22.88	25.10	0.800	0.800	0.740	0.738	0.78
2.82	22.94	24.04	0.900	0.900	0.840	0.837	0.85
3.14	22.99	22.47	0.915	0.915	0.855	0.849	0.91
3.99	22.94	21.23	0.945	0.955	0.890	0.893	0.92

Experiment 3

Chiu et al. also analyzed a series of models where the load factor (i.e. service rate of the I/O devices / service rate of the CPU) on the system is varied. The performance index of interest was the CPU utilisation. In their paper the CPU utilisation was plotted against the workload. An estimate of the results for CQM from the graph in the paper is contained in table 4.6 as are the results from QNEMU. Again the CQM* results appear to be very close to the CQM results.

table 4.6 Model Comparison Between [Chiu 75] and QNEMU

Load Factor	CQM	CQM*	CSM*
3	0.45	0.454	0.355
3.5	0.50	0.504	0.401
4	0.55	0.548	0.444
4.5	0.59	0.588	0.484
5	0.62	0.623	0.520
5.5	0.65	0.654	0.553
6	0.68	0.682	0.584
6.5	0.71	0.706	0.611
7	0.73	0.728	0.637

Experiment 4

Chiu et al. analyzed a series of models where the workload was varied. Again the performance index of interest was the CPU utilisation. The values taken from the paper and included in table 4.7 have a small margin of error as they were read off a graph. The results obtained from QNEMU are in most cases identical to these values.

table 4.7 CPU Utilisation Comparison with the Load Factor = 0.5

Workload	CPU utilisation			
	CQM	CQM*	CSM	CSM*
2	0.60	0.600	0.50	0.500
3	0.72	0.714	0.60	0.600
4	0.78	0.778	0.67	0.667
5	0.82	0.828	0.71	0.714

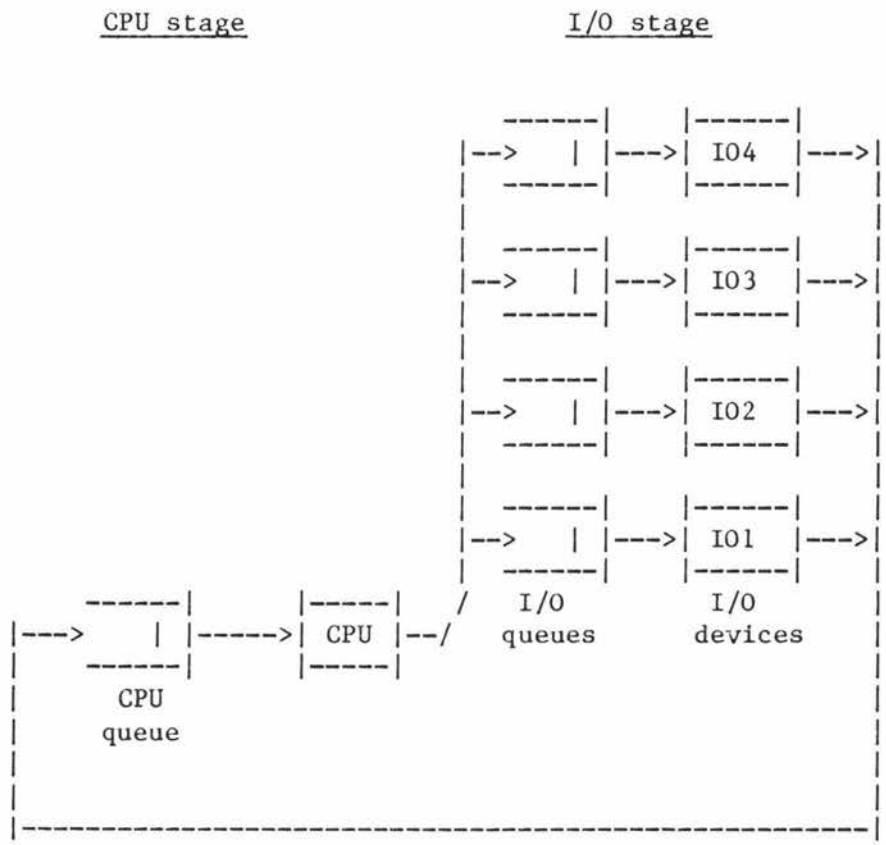
4.3 Validation Exercise 2

4.3.1 Introduction

The second validation exercise makes use of the analysis of an hypothetical system discussed in [Bask 75]. The particular model of interest is given in figure 4.8.

The system contains a CPU and four I/O devices, which are modelled with a separate queue for each I/O device. This model is another example of the Central Server Model described in Appendix H, section H.7.1. The CPU is modelled with a processor sharing (PS) service discipline, and the four I/O devices are each modelled with a first come first served load independent service discipline (FCFS).

figure 4.8 Central Server Model used in [Bask 75]



The service distributions at all the service centres are assumed negative exponentially distributed, and the means of the I/O devices are assumed fixed over the range of jobs modelled. The mean service rates of the I/O devices and of the CPU are given in table 4.9 for the different workloads. Table 4.9 also contains both the results given in the paper and the results obtained from QNEMU. The QNEMU results are again labelled with an asterisk for each workload.

table 4.9 Results from Performance Comparison with [Bask 75]

Workload	Utilisations					Routing Probabilities from CPU to ...				CPU service
	CPU	IO1	IO2	IO3	IO4	IO1	IO2	IO3	IO4	rate
1	0.600	0.400	0.000	0.000	0.000	1.0	0.0	0.0	0.0	1.0
1*	0.600	0.400	0.000	0.000	0.000	1.0	0.0	0.0	0.0	1.0
2	0.588	0.322	0.333	0.222	0.143	0.336	0.232	0.232	0.199	2.439
2*	0.589	0.325	0.330	0.220	0.144	0.340	0.230	0.230	0.200	2.439
3	0.631	0.308	0.532	0.354	0.228	0.233	0.268	0.268	0.230	3.139
3*	0.631	0.304	0.535	0.356	0.228	0.233	o.268	0.268	0.230	3.139
4	0.665	0.303	0.664	0.442	0.284	0.193	0.282	0.282	0.242	3.536
4*	0.666	0.306	0.660	0.440	0.289	0.195	0.280	0.280	0.245	3.536
5	0.689	0.300	0.754	0.503	0.323	0.173	0.290	0.290	0.248	3.780
5*	0.689	0.300	0.755	0.503	0.321	0.173	0.290	0.290	0.247	3.780
6	0.708	0.299	0.818	0.545	0.350	0.161	0.294	0.294	0.252	3.934
6*	0.707	0.288	0.818	0.546	0.358	0.155	0.294	0.294	0.257	3.934
7	0.722	0.299	0.863	0.575	0.370	0.154	0.296	0.296	0.254	4.034
7*	0.725	0.302	0.860	0.574	0.376	0.155	0.294	0.294	0.257	4.034
8	0.734	0.300	0.896	0.597	0.384	0.149	0.298	0.298	0.255	4.100
8*	0.740	0.313	0.892	0.595	0.390	0.155	0.294	0.294	0.257	4.100

The routing probabilities used in a number of the above analyses differ slightly from the routing probabilites used in the paper. This is due to the fact that the algorithm used in QNEMU requires that the determinant of the routing probability matrix be equal to zero [Oldf 82]. If it is not, no non-trivial solution exists to calculate the normalisation constant G (see Product Form in Appendix H, section H.7.1), which means that no analysis of the network is possible. For a number of the networks in the Baskett paper, the determinant is non-zero. Slight adjustments in the

routing probabilities were made for these networks, as given in table 4.9, to facilitate analysis.

The results show very close agreement between the two models.

4.4 Validation Exercise 3

The third validation exercise is based on a discrete simulation of a tanker fleet. This particular example can be found in [Prit 74] as an example of the use of GASP IV, a combined continuous/discrete FORTRAN-based simulation language. This validation exercise is an example of how QNEMU can be applied to problems traditionally handled by simulation techniques that require a much larger input of resources to reach similar results.

The system to be modelled consists of a fleet of 15 tankers. The tankers are loaded with oil in Alaska and travel individually by sea to a single unloading wharf in Seattle. There are 15 loading wharves in Alaska which can load on average 50,000 barrels per day. Each tanker can nominally hold 150,000 barrels. The times to load and unload a tanker are negative-exponentially distributed with a mean unloading rate of 200,000 barrels per day. Both transit times are normally distributed, but in the QNEMU analysis the negative-exponential distribution will be used. The transit time between Alaska and Seattle has a mean of five days and between Seattle and Alaska a mean of four days.

The simulation using GASP IV was carried out for 365 days to obtain estimates of dock utilisation, tanker round trip time, tanker waiting time, and the number of tankers waiting for unloading.

The simulation program in GASP IV involved the coding of five FORTRAN procedures containing 195 lines of source code. The results of the QNEMU analysis can be found as Session Two in Appendix D. The response time taken for the actual QNEMU analysis on the Prime 750 system is of the order of seconds, while the time taken to input the network is of the order of minutes.

The server details used in the QNEMU analysis are given in table 4.10, and the results of both analyses in table 4.11.

Table 4.10 QNEMU Input Parameters

Node	Queueing Discipline	Service Rate tankers/day
Alaska	Infinite Server	0.3333
AtoS	Infinite Server	0.20
Seattle	FCFS 1 Server	1.3333
StoA	Infinite Server	0.250

Table 4.11 GASP IV and QNEMU Results of Tanker Simulation

Analysis Method	Seattle mean wait time days	Seattle mean queue length tankers	Seattle mean dock utilisation	Round Trip mean time days
GASP IV	1.74	1.8	0.79	13.5
QNEMU	2.287	2.398	0.787	14.287

The utilisation compares very closely between QNEMU and GASP IV, but the other results from QNEMU tend to be pessimistic, due in part to the use of the negative-exponential distribution instead of the normal distribution. In addition to this the calculation of the server utilisation depends on only the first moment (mean) of the distribution, whereas the other results depend on the higher moments which differ between the various distributions.

5 CONCLUSIONS and FURTHER RESEARCH

The QNEMU package was successfully implemented. From the experience gained in this implementation the use of compiler construction techniques to produce user friendly interfaces is felt to be more than justified. In addition this implementation has shown, that even with the conflicting design goals of portability and user friendliness, a useful software package can be developed which satisfies both goals to a reasonably high degree. The actual degree of both user friendliness and portability of QNEMU will be future areas of research, as evaluating both these areas would require much detailed research outside the scope of this thesis.

Chapter One has outlined a number of objectives which it is suggested that designers of interactive systems should consider when developing new interactive packages, or even when improving existing software packages. It is by no means a complete study of this area, but does introduce much of the thinking behind the design of the QNEMU package. In addition the idea of applying many of the techniques developed for compiler construction to interactive packages is introduced.

Chapter Two gives an user overview of the QNEMU package with the user friendly features prominent throughout. This chapter may be more detailed in some areas than required, and may not provide the detail in other areas, but hopefully the general "flavour" of the package is given. A more detailed view can be obtained by the actual use of the package and by reading the user manual [QNEM 84].

Chapter Three describes the implementation of many of the important design features discussed in chapter one, with an emphasis on the implementation features used to provide portability of the software and the user friendly interface. This chapter highlights the use of compiler construction techniques used in the implementation of QNEMU.

Chapter Four describes the validation of the QNEMU package carried out to date. It is felt that the underlying algorithms can be assumed to provide relatively accurate results, with a reasonable degree of confidence.

5.1 Further Research

There are a number of areas in which further research could usefully be carried out. The general areas are in the extension of the package, the application of the package to a large range of problems, and an evaluation of how well QNEMU has met the portability and user friendly design objectives.

The easiest of these to carry out would be to investigate the portability of the QNEMU package, by attempting to transfer it to both other Prime systems and also non-Prime systems.

Three important extensions of the package are:

- the inclusion of multiple class network analysis;

- the ability to automatically tabulate results for, amongst other things, a varying multiprogramming level, or varying service rate at a single node;

- the ability to produce graphs of the tabulated results.

Finally, the use of QNEMU in the research of the performance of current and proposed computer systems. Also a study of the application of the QNEMU package to many of the problems traditionally handled by simulation techniques.

APPENDIX A - Syntax Diagrams of the QNEMU Command Language

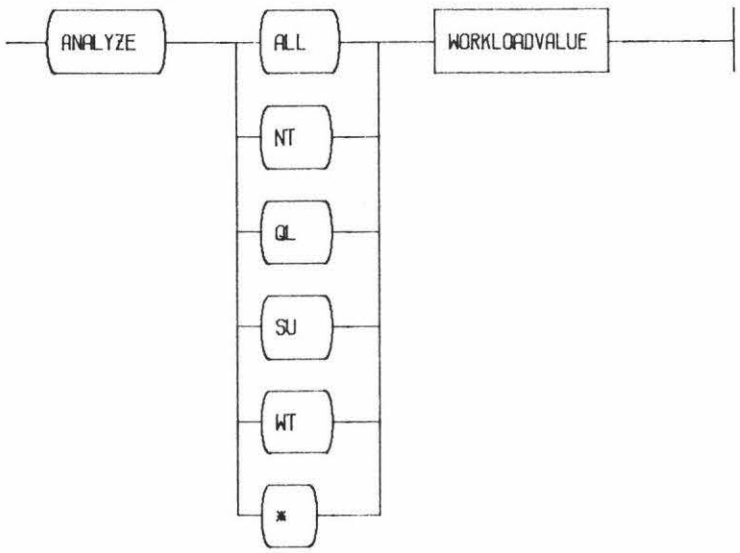
This appendix contains the tramline diagrams of the QNEMU command language. For readers not familiar with the tramline syntax diagrams as given here a brief description of their general structure follows.

The terminal symbols that the user is actually required to enter appear in the boxes with rounded sides, e.g. the box containing ALL in the first diagram.

The non-terminals, which the user does not enter, but which are described in terms of terminal symbols elsewhere in the tramline diagrams, appear in the square boxes, e.g. the box containing WORKLOADVALUE in the first diagram.

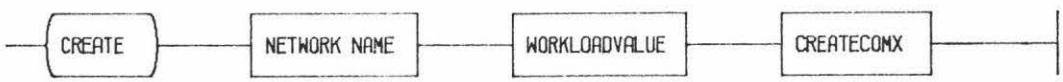
The tramline from which all the other tramlines come from is that describing the non-terminal SESSION.

ANALYZECOM

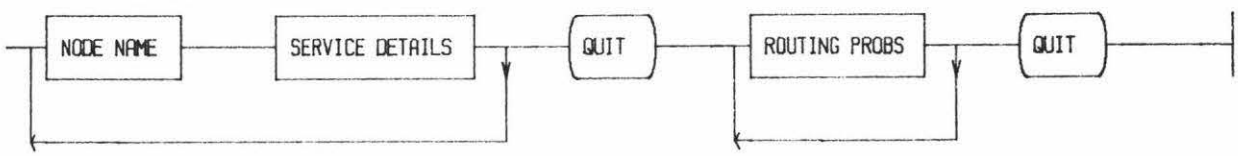


COPYCOM

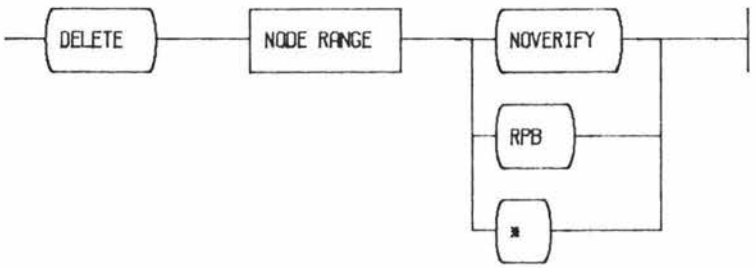
CREATECOM



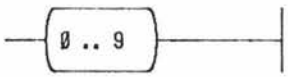
CREATECOMX



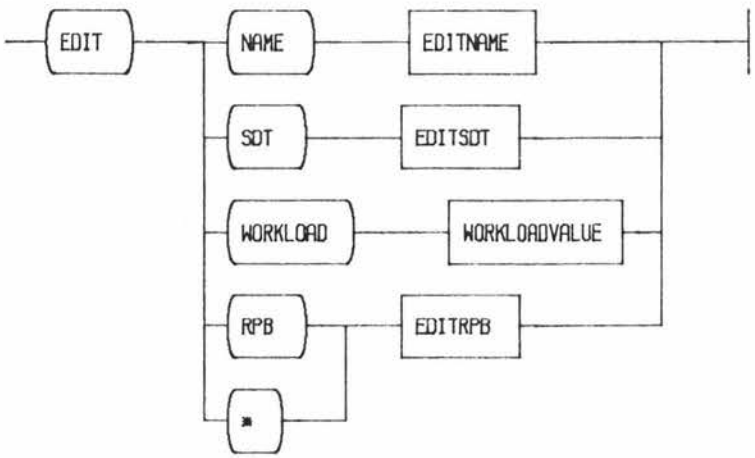
DELETECOM



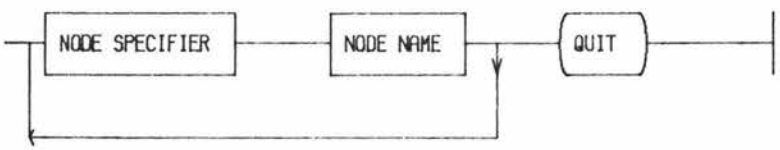
DIGIT



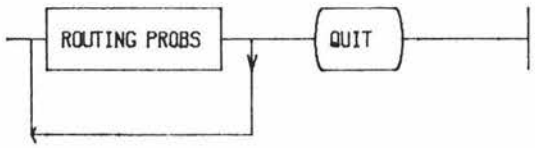
EDITCOM



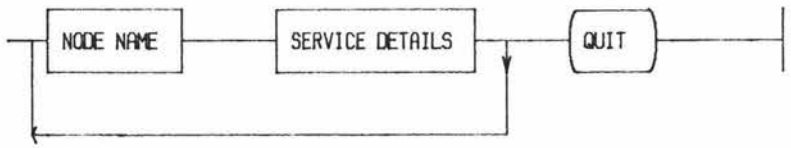
EDITNAME



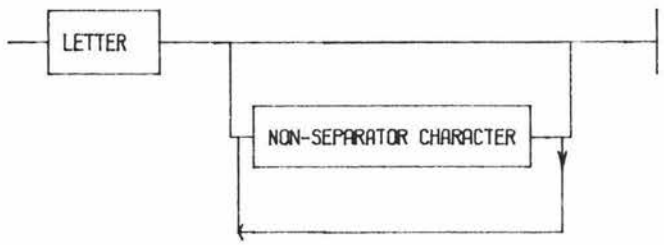
EDITRPB



EDITSDT

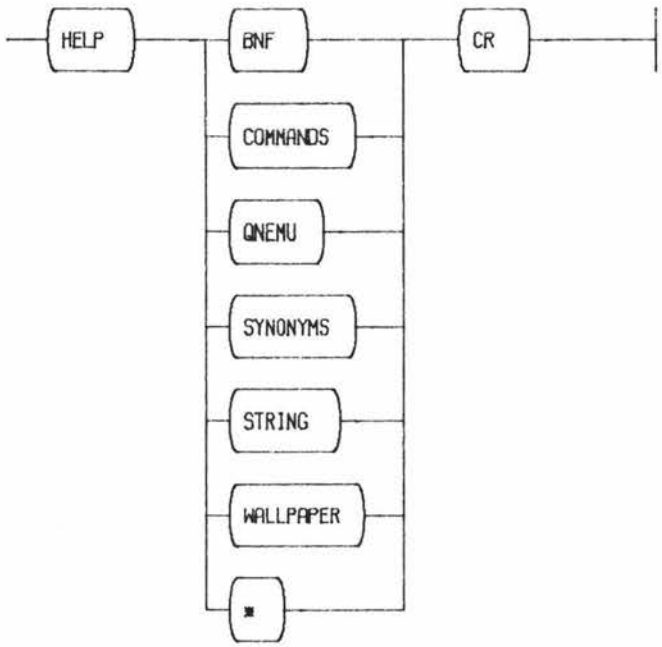


FILE NAME

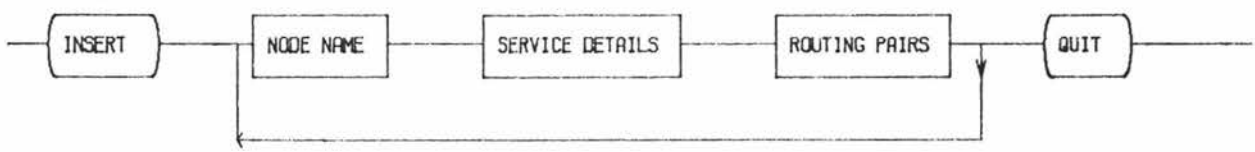


GETCOM

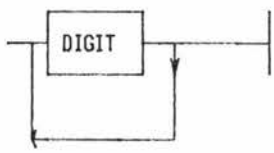
HELPCOM



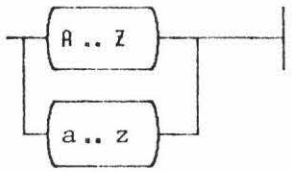
INSERTCOM



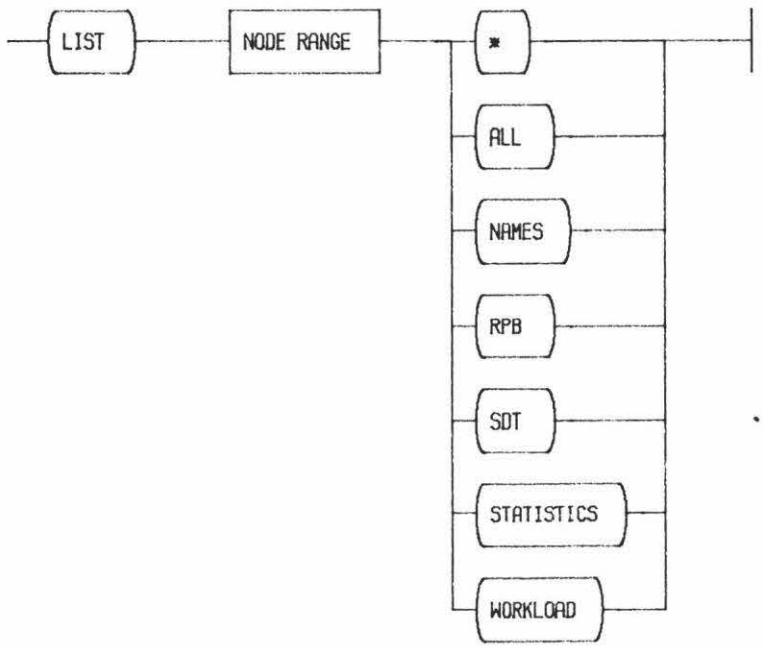
INTEGER



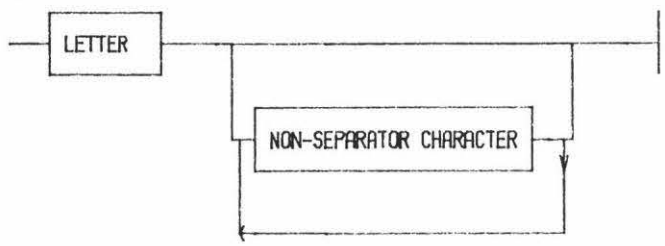
LETTER



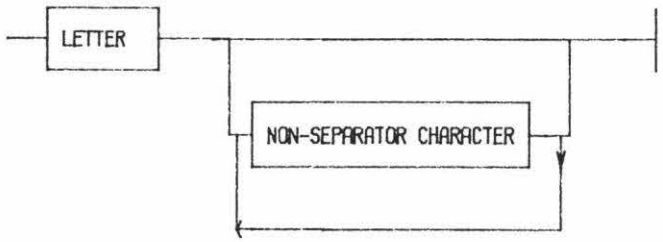
LISTCOM



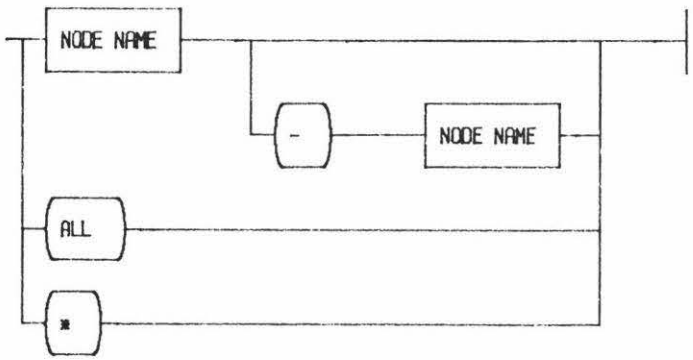
NETWORK NAME



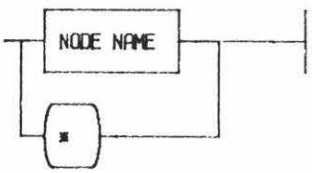
NODE NAME



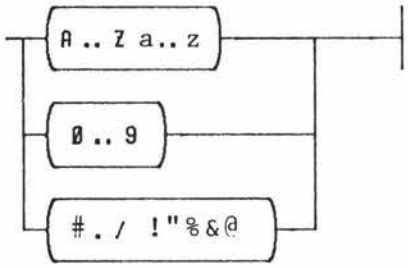
NODE RANGE



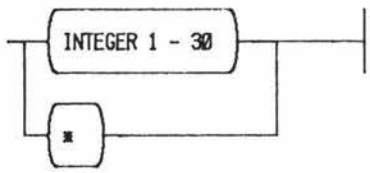
NODE SPECIFIER



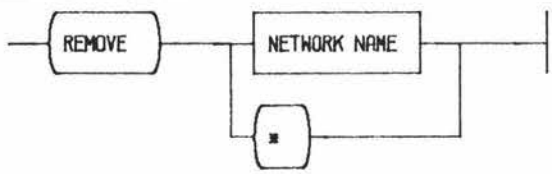
NON-SEPARATOR CHARACTER



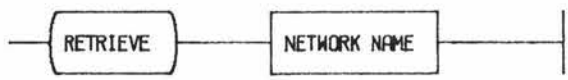
NUM OF SERVERS



REMOVECOM

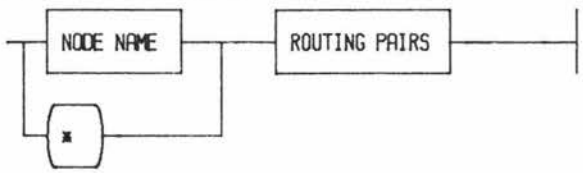


RETRIEVECOM

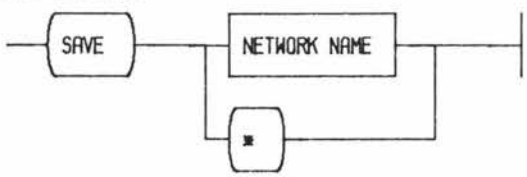


ROUTING PAIRS

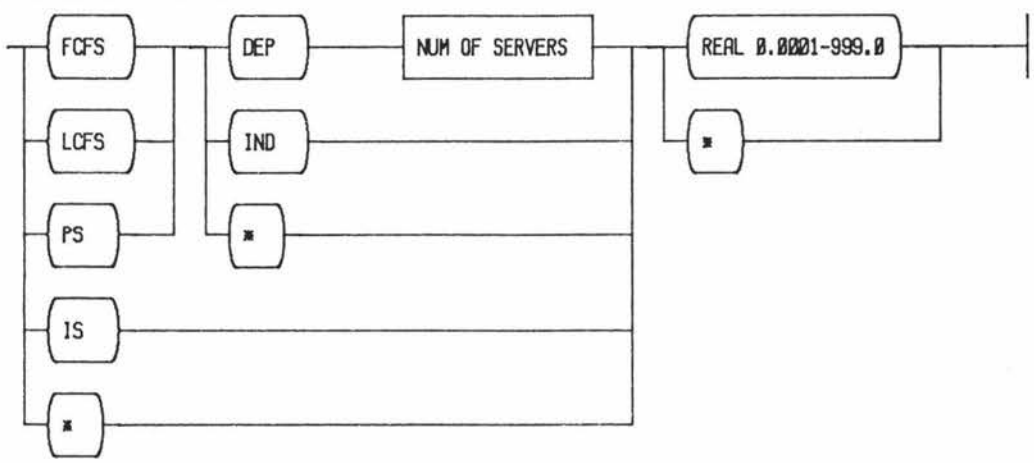
ROUTING PROBS

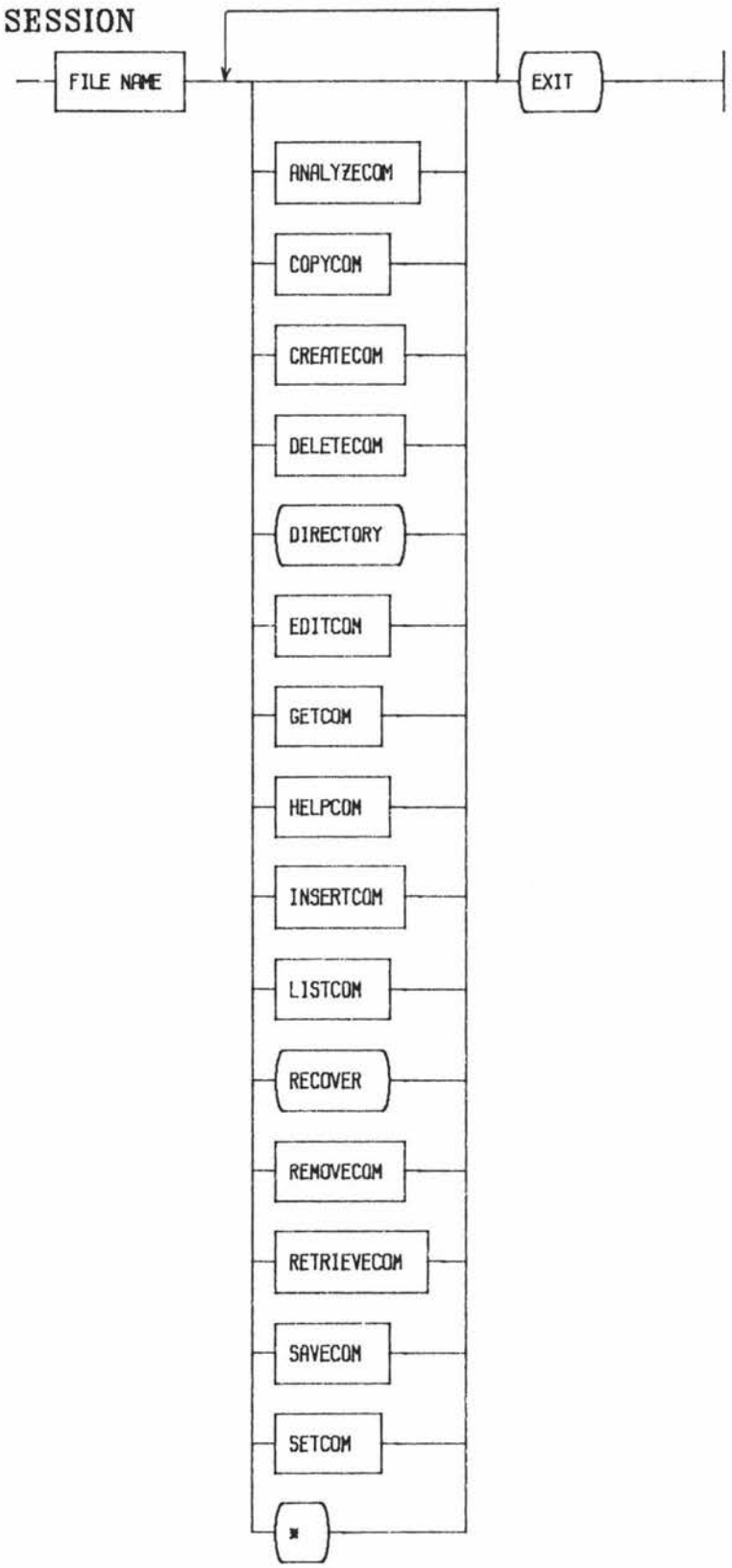


SAVECOM

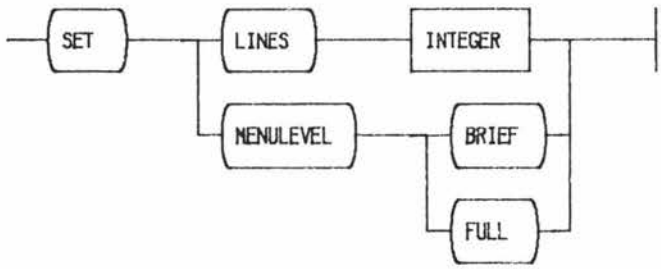


SERVICE DETAILS

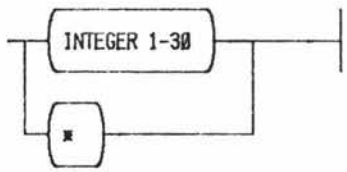



SESSION
FILE NAME
EXIT
ANALYZECOM
COPYCOM
CREATECOM
DELETECOM
DIRECTORY
EDITCOM
GETCOM
HELPCOM
INSERTCOM
LISTCOM
RECOVER
REMOVECOM
RETRIEVECOM
SAVECOM
SETCOM
*

SETCOM



WORKLOADVALUE

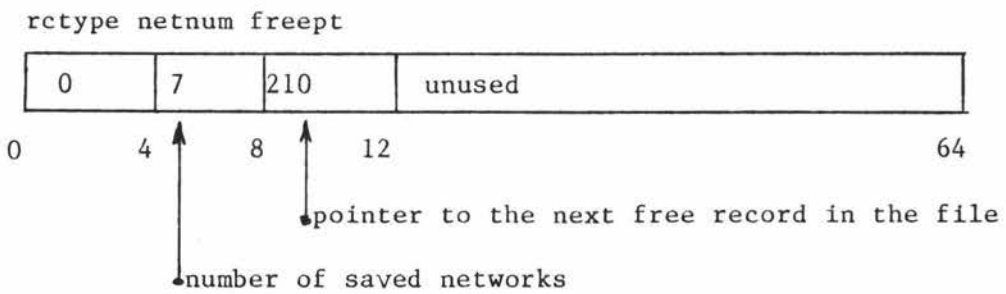


APPENDIX B - Library File Record Formats

The following describes the formats of the different records contained in the library file. The variable names in the diagrams (e.g. rctype) are the actual names used in the relevant QNEMU subroutines. The values given in the record fields are sample values for each field. The numbers below each record indicate the byte count of the last byte in each field, and a description accompanies most fields with an arrow to the field it describes.

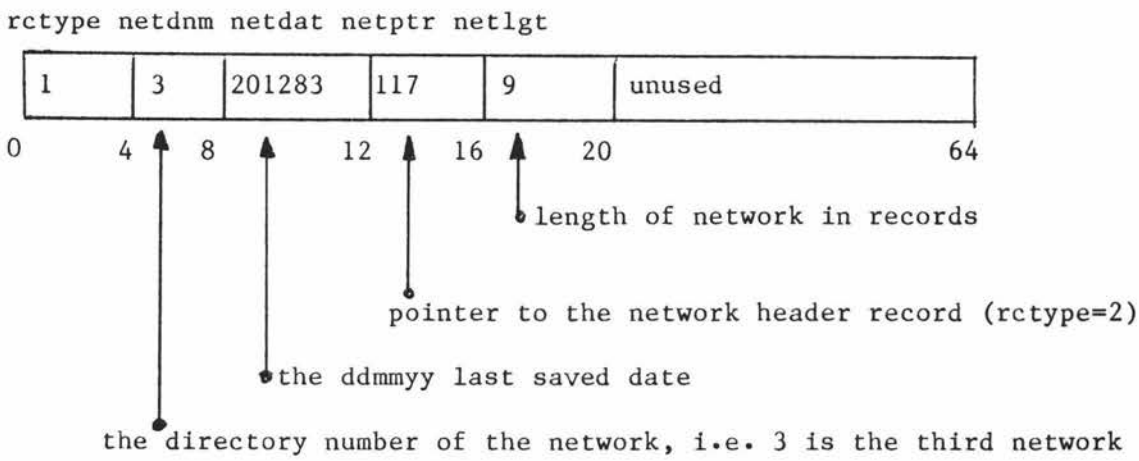
Directory Header Record, rctype = 0

The directory header record contains the details of the network directory for this library file.



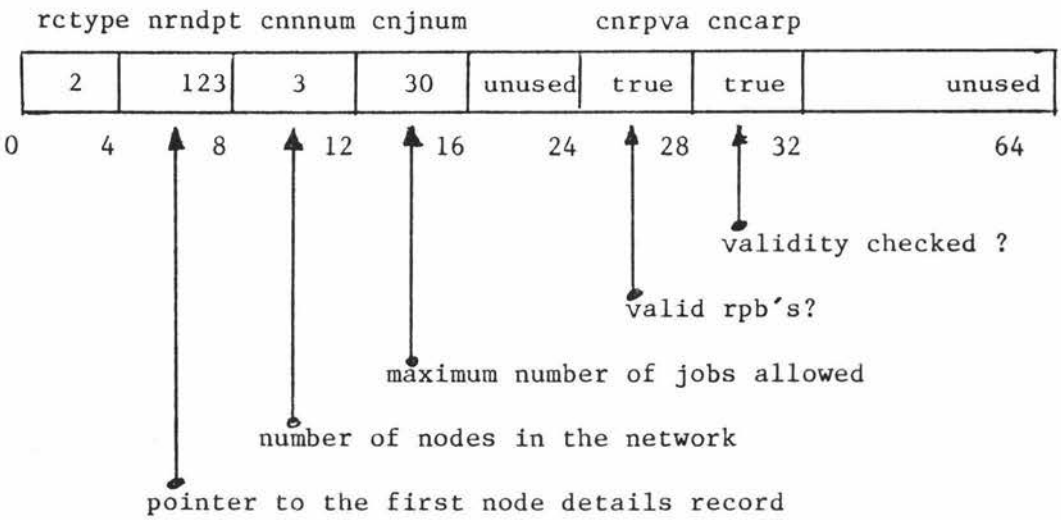
Directory Entry Record, rctype = 1

The directory entry record i contains directory details of network i.



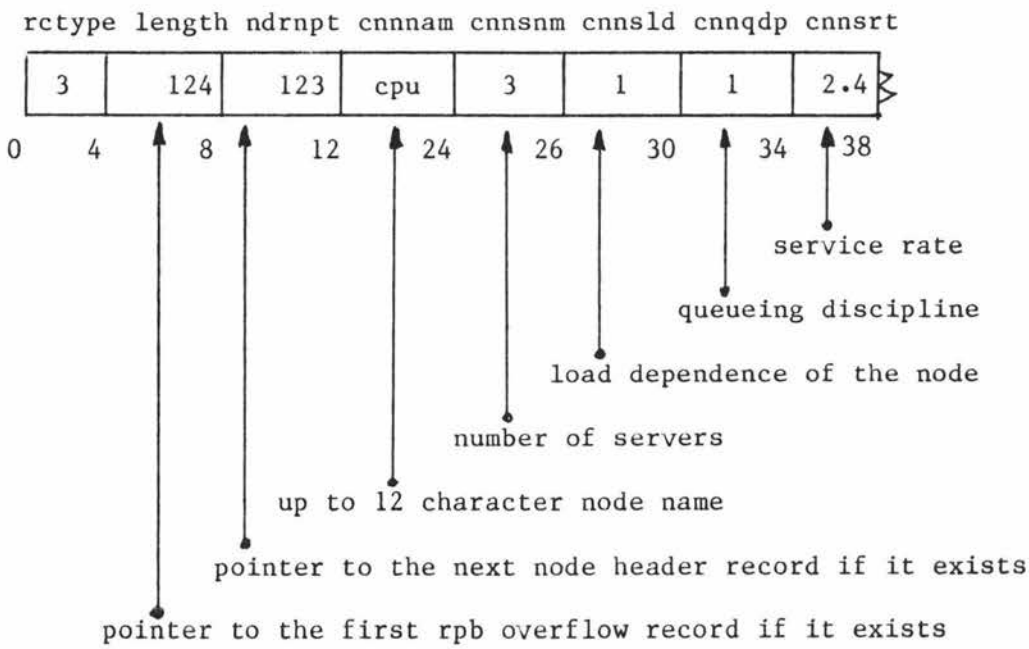
Network Header Record, rctype = 2

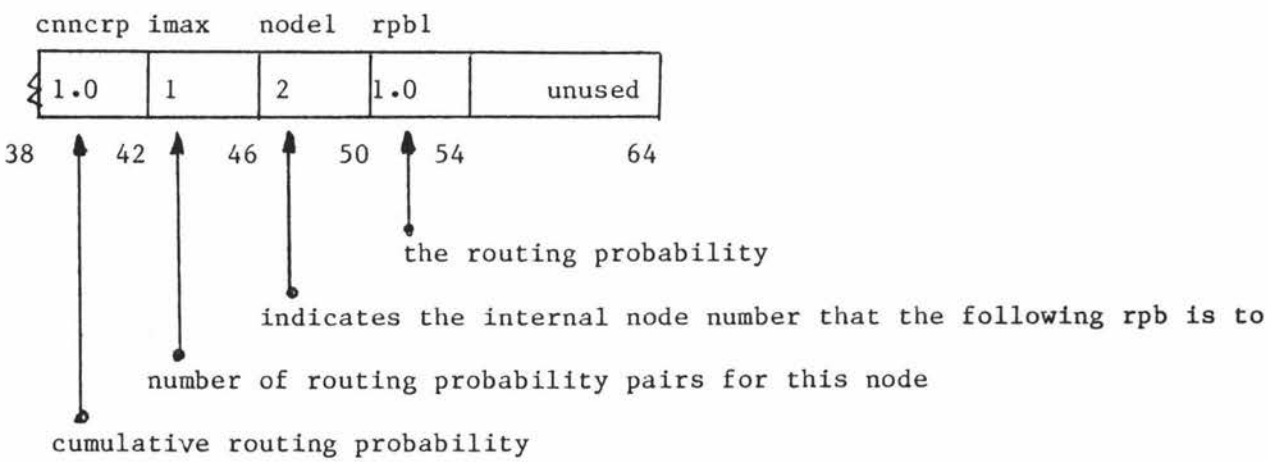
The network header record contains the workload details and general information on the network.



Node Header Record, rctype = 3

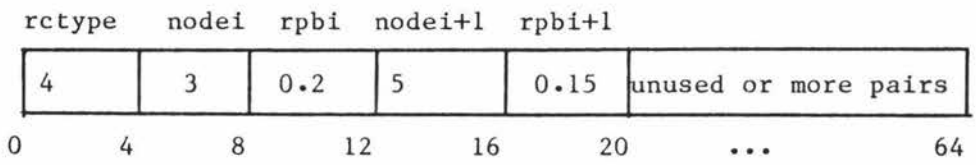
This contains the details of each node in the network, apart from any overflow routing probabilities which are contained in Node RPB records.





Node RPB Record, rctype = 4

These records only exist if the number of routing probability pairs leaving a node are more than one, then these records are used for the overflow.



APPENDIX C - QNEMU Consistency Checks

The following calculations describe two consistency checks which can be used on the performance results produced by QNEMU for any network.

Figure C.1, below, contains the performance data used in the examples. (Note that the figures given in the figure are only to 2D whereas QNEMU will actually produce 3D results.)

Check 1: Mean Server Utilisation

For any node in the network:

$$\text{m.s.u} = \frac{\text{mean node throughput (m.n.t.)}}{\text{num. of servers x service rate}}$$

example:

m.s.u. of cpu = 0.59 jobs; 1.44 / (1 x 2.439) = 0.590 jobs

m.s.u. of io2 = 0.33 jobs; 0.49 / (1 x 1.5) = 0.327 jobs

m.s.u. of io3 = 0.33 jobs; 0.33 / (1 x 1.0) = 0.330 jobs

m.s.u. of io4 = 0.22 jobs; 0.33 / (1 x 1.5) = 0.220 jobs

m.s.u. of io5 = 0.14 jobs; 0.29 / (1 x 2.0) = 0.145 jobs

Check 2: Sum of Mean Queue Lengths

The sum of m.q.l's = workload

example:

The sum of m.q.l's = 0.8 + 0.39 + 0.4 + 0.25 + 0.16

= 2.00 jobs

figure C.1 Example for Qnemu Consistency Checks

Network name	Date last saved	maximum workload	mean server util. : 2.00 jobs mean node thruput : 2.00 jobs mean queue length : 2.00 jobs mean waiting time : 2.00 jobs
NEWELL	16/12/83	30	

node name	cum. rpb.	queue displ.	load dep.	num. ser.	service rate	analyzed values m.s.u.	m.n.t.	m.q.l.	m.w.t
cpu	1.00	PS	no	1	2.4390	0.59	1.44	0.80	0.56
io2	1.00	FCFS	no	1	1.5000	0.33	0.49	0.39	0.80
io3	1.00	FCFS	no	1	1.0000	0.33	0.33	0.40	1.21
io4	1.00	FCFS	no	1	1.5000	0.22	0.33	0.25	0.76
io5	1.00	FCFS	no	1	2.0000	0.14	0.29	0.16	0.54

t h e e n d

```
 QNEMU  net:NEWELL                    com:analyze    lines    1-  10 of   10
Status: Analyzed the statistics of the current network.

Key in ... QNEMU COMMAND,( e.g. HELP ) ... or M
```

APPENDIX D - Sample QNEMU Sessions

This appendix contains the terminal screen images of two sample sessions.

The user input appears as the last line of each menu.

Session One

```
|-| |-| |-| |-| |-| |-| |-| |-| |-| |-| |-| |-|
|-|                                             |-|
|-| Please be patient, QNEMU is initialising    |-|
|-|                                             |-|
|-| |-| |-| |-| |-| |-| |-| |-| |-| |-| |-| |-|

                  T E N

                 N I N E

                E I G H T

                S E V E N

                  S I X

                 F I V E

                 F O U R

                T H R E E

                  T W O

                  O N E

 * * * * * * B L A S T   O F F * * * * * *
```

Queueing Network Evaluator of Massey University.

```
|-----------------------------------------------------------|
|                                                           |
| Specification of the FILE NAME the networks are held in   |
| the system.                                               |
|                                                           |
| If it doesn't already exist you will be prompted to find  |
| out if you wish a new file to be created.                 |
|                                                           |
| There is no way of listing the files.  You must find out  |
| the name of the file you wish to work with before entry.  |
|                                                           |
|   EXIT       :to stop execution of this package.          |
|                                                           |
|-----------------------------------------------------------|
```

Status: Initialised the Qnemu software package.

Key in ... the existing or new FILE NAME ... or EXIT.
thesis.ch4.dam

Queueing Network Evaluator of Massey University.

Command	Mnemonic	Command	Mnemonic
ANALYZE	A	COPY	CO
CREATE	C	DELETE	DEL
DIRECTORY	D	EDIT	E
EXIT	EXIT	* HELP	H
LIST	L	GET	G
RECOVER	REC	INSERT	I
REMOVE	REM	RETRIEVE	RET
SAVE	S	SET	SET

Current Network: a null network.

Status: thesis.ch4.dam file of networks has been read.

Key in ... QNEMU COMMAND,(e.g. HELP) ... or M
DIRECTORY

Saved network name	Date saved dd/mm/yy	Remove tag status
CHIU.3.14	23/12/83	
CHIU.3.99	23/12/83	
CHIU.CQM.ECM	15/12/83	
CHIU.CQM.LCS	15/12/83	
CHIU.CSM.ECM	23/12/83	
CHIU.CSM.LCS	17/12/83	
CHIU.ECM.FIG7	15/12/83	
CHIU.ECM.FIG7.CS	15/12/83	
CHIU.ECM.TAB2	17/12/83	
CHIU.ECM.TAB2.CS	17/12/83	
NEWELL	16/12/83	
NEWELL1	16/12/83	
NEWELL2	16/12/83	
QL.TEST	18/12/83	
TANKER.SIMULATION	17/12/83	
WRONG.EXAMPLE	18/12/83	

t h e e n d

QNEMU fil:thesis.ch4.dam com:directory lines 1- 16 of 16

Key in ... M to get the QNEMU menu ... or the QNEMU COMMAND.
get chiu.cqm.lcs

```
                Queueing Network Evaluator of Massey University.

|-------------------------------------------------------|
|                                                       |
|  Command     Mnemonic            Command     Mnemonic |
|  --------    ----------          ---------   ---------- |
|  ANALYZE        A                  COPY         CO     |
|  CREATE         C                  DELETE       DEL    |
|  DIRECTORY      D                  EDIT         E      |
|  EXIT           EXIT             * HELP         H      |
|  LIST           L                  GET          G      |
|  RECOVER        REC                INSERT       I      |
|  REMOVE         REM                RETRIEVE     RET    |
|  SAVE           S                  SET          SET    |
|                                                       |
|    Current Network:CHIU.CQM.LCS                       |
|                                                       |
|-------------------------------------------------------|

Status: Get network from the disk library completed.

Key in ... QNEMU COMMAND,( e.g. HELP ) ... or M
set lines 100

                Queueing Network Evaluator of Massey University.

|-------------------------------------------------------|
|                                                       |
|  Command     Mnemonic            Command     Mnemonic |
|  --------    ----------          ---------   ---------- |
|  ANALYZE        A                  COPY         CO     |
|  CREATE         C                  DELETE       DEL    |
|  DIRECTORY      D                  EDIT         E      |
|  EXIT           EXIT             * HELP         H      |
|  LIST           L                  GET          G      |
|  RECOVER        REC                INSERT       I      |
|  REMOVE         REM                RETRIEVE     RET    |
|  SAVE           S                  SET          SET    |
|                                                       |
|    Current Network:CHIU.CQM.LCS                       |
|                                                       |
|-------------------------------------------------------|

Status: The number of lines on the terminal screen has been set to: 100

Key in ... QNEMU COMMAND,( e.g. HELP ) ... or M
a all 2.2
```

```
|-------------------------------------------------------------------------|
|       Network name        | Date last|maximum | mean server util. : 2.20 jobs|
|---------------------------|  saved   |workload| mean node thruput : 2.20 jobs|
|CHIU.CQM.LCS               |----------|--------| mean queue length : 2.20 jobs|
|                           | 15/12/83 |   30   | mean waiting time : 2.20 jobs|
|-------------------------------------------------------------------------|
| node name    cum.  queue  load num.  service |       analyzed values      |
|              rpb.  displ. dep. ser.   rate    | m.s.u.  m.n.t.  m.q.l.  m.w.t|
|-------------------------------------------------------------------------|
| chanl        1.000  FCFS  yes    2   21.4100 |   0.561 24.009  1.164  0.048 |
|                                               |                              |
| cpu          1.000  FCFS  no     1   34.8430 |   0.689 24.009  1.036  0.043 |
|                                               |                              |
|------------------------------ t h e  e n d -------------------------------|
 QNEMU  net:CHIU.CQM.LCS                com:analyze    lines    1-   4 of    4

Key in ... M to get the QNEMU menu ... or the QNEMU COMMAND.
exit
```

File saved :
thesis.ch4.dam

```
####   #   #   ####          ####   #   #   ####
#  #    # #    #             #  #    # #    #
####     #     ####          ####     #     ####
#  #     #     #             #  #     #     #
####     #     ####          ####     #     ####
```

Session Two

```
|-| |-| |-| |-| |-| |-| |-| |-| |-| |-| |-| |-|
|-|                                               |-|
|-| Please be patient, QNEMU is initialising      |-|
|-|                                               |-|
|-| |-| |-| |-| |-| |-| |-| |-| |-| |-| |-| |-|
```

T E N

N I N E

E I G H T

S E V E N

S I X

F I V E

F O U R

T H R E E

T W O

O N E

* * * * * * B L A S T O F F * * * * * *

Queueing Network Evaluator of Massey University.

```
|-----------------------------------------------------|
|                                                     |
| Specification of the FILE NAME the networks are held in |
| in the system.                                      |
|                                                     |
| If it doesn't already exist you will be prompted to find |
| out if you wish a new file to be created.           |
|                                                     |
| There is no way of listing the files.  You must find out |
| the name of the file you wish to work with before entry. |
|                                                     |
|   EXIT      :to stop execution of this package.     |
|                                                     |
|-----------------------------------------------------|
```

Status: Initialised the Qnemu software package.

Key in ... the existing or new FILE NAME ... or EXIT.
thesis.ch4.dam

Queueing Network Evaluator of Massey University.

Command	Mnemonic	Command	Mnemonic
ANALYZE	A	COPY	CO
CREATE	C	DELETE	DEL
DIRECTORY	D	EDIT	E
EXIT	EXIT	* HELP	H
LIST	L	GET	G
RECOVER	REC	INSERT	I
REMOVE	REM	RETRIEVE	RET
SAVE	S	SET	SET

Current Network: a null network.

Status: thesis.ch4.dam file of networks has been read.

Key in ... QNEMU COMMAND,(e.g. HELP) ... or M
get tanker.simulation

Queueing Network Evaluator of Massey University.

Command	Mnemonic	Command	Mnemonic
ANALYZE	A	COPY	CO
CREATE	C	DELETE	DEL
DIRECTORY	D	EDIT	E
EXIT	EXIT	* HELP	H
LIST	L	GET	G
RECOVER	REC	INSERT	I
REMOVE	REM	RETRIEVE	RET
SAVE	S	SET	SET

Current Network:TANKER.SIMULATION

Status: Get network from the disk library completed.

Key in ... QNEMU COMMAND,(e.g. HELP) ... or M
a all 15

Network name	Date last saved	maximum workload	mean server util. :15.00 jobs mean node thruput :15.00 jobs mean queue length :15.00 jobs mean waiting time :15.00 jobs
TANKER.SIMULATION	17/12/83	30	

node name	cum. rpb.	queue displ.	load dep.	num. ser.	service rate	analyzed values m.s.u.	m.n.t.	m.q.l.	m.w.t
alaska	1.000	IS	no	inf.	0.3333	0.210	1.049	3.146	3.000
atos	1.000	IS	no	inf.	0.2000	0.350	1.049	5.244	5.000
seattle	1.000	FCFS	no	1	1.3333	0.787	1.049	2.398	2.287
stoa	1.000	IS	no	inf.	0.2500	0.280	1.049	4.195	4.000

t h e e n d

QNEMU net:TANKER.SIMULATION com:analyze lines 1- 8 of 8

Key in ... M to get the QNEMU menu ... or the QNEMU COMMAND.
d

Saved network name	Date saved dd/mm/yy	Remove tag status
CHIU.CQM.ECM	15/12/83	
CHIU.CQM.LCS	15/12/83	
CHIU.CSM.ECM	17/12/83	
CHIU.CSM.LCS	17/12/83	
CHIU.ECM.FIG7	15/12/83	
CHIU.ECM.FIG7.CS	15/12/83	
CHIU.ECM.TAB2	17/12/83	
CHIU.ECM.TAB2.CS	17/12/83	
NEWELL	16/12/83	
NEWELL1	16/12/83	
NEWELL2	16/12/83	
QL.TEST	18/12/83	
TANKER.SIMULATION	17/12/83	

t h e e n d

QNEMU fil:thesis.ch4.dam com:directory lines 1- 13 of 13

Key in ... M to get the QNEMU menu ... or the QNEMU COMMAND.
exit

File saved :
thesis.ch4.dam

```
####   #   #   ####          ####   #   #   ####
#  #    # #    #             #  #    # #    #
####     #     ####          ####     #     ####
#  #     #     #             #  #     #     #
####     #     ####          ####     #     ####
```

APPENDIX E - Selected QNEMU Subroutines

The following three subroutines are representative of the Qnemu source code and show a number of the features referenced in the description of the Qnemu implementation. The first is described briefly in section 3.4 as a high level structure diagram.

```
      subroutine getsym(menu,error)
c*************************************************************************
c* read the next symbol from the input line stored in inline.
c* There are the following global parameters affected by this subroutine.
c*       sym       returns the value of the command or symbol type found.
c*       relval    returns the real value if a real or integer entered.
c*       intvax    returns the integer value when a number is entered or
c*                 returns the number of characters in strval if sym=strsym
c*       intval    returns the index of the node found if nodfnd is true.
c*       intva2    returns the index of the networks position in directory.
c*       strval    returns the characters found if sym=strsym.
c*       strvau    returns the uppercase strval characters found if sym=strsym.
c*       strvax    returns the lowercase characters found if sym=strsym.
c*       nodfnd    returns true if a node specifier was found(i.e. a integer in
c*                 the valid range or a valid node name .)
c*       netfnd    returns true if a network name in the directory was found.
c*
c*  note commands are upcased but case is significant for all other strings.
c*************************************************************************
$INSERT CONST2.F77

c        declare the parameters.

      integer menu,
     &        error

      external menupt,
     &         listcn

c        declare the local variables.

      integer i,j,k,
     &        val,
     &        expn,
     &        top,
     &        bot,
     &        mid
      character ch
```

```
      integer tracun
      parameter ( tracun=19)
c*******************************************************************************
      character lowcas*26,
     &        upcase*26,
     &        nonsep*26
c*******************************************************************************
      character nnupce*namemx
      intrinsic ichar,
     &        char
c*******************************************************************************

c        declare some user macros.

      logical digit,
     &        alpha,
     &        godchr
      integer eval
      character c
      digit(c)=((c .ge. '0') .and. (c .le. '9'))
      alpha(c)=(((c .ge. 'A') .and. (c .le. 'Z')).or.
     &          ((c .ge. 'a') .and. (c .le. 'z')))
      eval(c)=val*10+(ichar(c)-ichar('0'))

c*******************************************************************************

      lowcas='abcdefghijklmnopqrstuvwxyz'
      upcase='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      nonsep='!.#$%&

c*******************************************************************************

  100 continue
      strval=blank
      strvau=blank
      strvax=blank
      sym=invsym
      intvax=0
      intval=0
      intva2=0
      relval=0.0
      nodfnd=.false.
      netfnd=.false.

c      test for end of input or an error.Print prompt and any error message.

      if ((colptr .gt. lascol) .or. (error .gt. 0)) then
         call menupt(menu,error)
         read(termun,fmt(19))inline
         write(tracun,fmt(19))inline
         inline(inlnmx:inlnmx)=semico
         colptr=1
         lascol=inlnmx
         error=-1
```

```
      end if

c             search for the next symbol in the input line inline.

      do 200 i=colptr,lascol
         if (inline(i:i) .ne. blank)  goto 210
  200 continue
      i=lascol+1
  210 continue
      colptr=i

c        test for end of input.

      if (colptr .gt. lascol) then
         sym=crsym
      else

c        some symbol to be scanned.

         ch=inline(colptr:colptr)
         if (ch .eq. hyphen) then
              sym=ransym
              colptr=colptr+1
         else if((ch .eq. semico) .or. (ch .eq. comma)) then
              sym=crsym
              colptr=colptr+1
         else if(ch .eq. asteri) then

c          an asterisk stands for the default value.

              sym=astsym
              colptr=colptr+1
         else if (digit(ch) .or. (ch .eq. period)) then

c             scan a number.

              val=0
              sym=intsym
 1205         continue
              if((.not. digit(ch)) .or. (val .ge. maxint))goto 1210
                   val=eval(ch)
                   colptr=colptr+1
                   ch=inline(colptr:colptr)
                   goto 1205
 1210          continue
               intvax=val
               relval=val
               if (ch .eq. period) then

c                   scan the real portion.

                    expn=0
 1220               continue
                         colptr=colptr+1
```

```
                    ch=inline(colptr:colptr)
                    if(.not. digit(ch)) goto 1230
                       expn=expn+1
                       val=eval(ch)
                 goto 1220
1230             continue
                 sym=relsym
                 relval=val*10.0**(-expn)
               end if

      else

c           scan a string of non-separator characters(i.e. ; -  ,   )

            if (.not. alpha(ch)) goto 80000
            sym=strsym
1310        continue

c           test for a separator character.

            if ((ch .eq. blank) .or. (ch .eq. hyphen) .or.
     &           (ch .eq. semico) .or. (ch .eq. comma)) goto 1350
            godchr=.false.
            do 1315 i=1,26
               if (ch .eq. lowcas(i:i)) godchr=.true.
               if (ch .eq. upcase(i:i)) godchr=.true.
               if ( ch .eq. nonsep(i:i)) godchr=.true.
1315        continue

            if (.not. ( godchr .or. digit(ch)) ) goto 80000

                 intvax=intvax+1
                 if(intvax .lt. namemx) then
                      strval(intvax:intvax)=ch
                      strvau(intvax:intvax)=ch
                      strvax(intvax:intvax)=ch
                      do 1340 i=1,26
                          if (upcase(i:i) .eq. ch)
     &                       strvax(intvax:intvax)=lowcas(i:i)
                          if (lowcas(i:i) .eq. ch)
     &                       strvau(intvax:intvax)=upcase(i:i)
1340                  continue
                 end if
                 colptr=colptr+1
                 ch=inline(colptr:colptr)
            goto 1310
1350        continue

c       search for the string in the command list.

            bot=1
            top=stbtop
1360        continue
            if(bot .gt. top) goto 1390
```

```
                mid=(bot + top) / 2
                if(strvau .le. symtab(mid)) top =mid-1
                if(strvau .ge. symtab(mid)) bot =mid+1
              goto 1360
 1390         continue
              if (bot-1 .gt. top) then

c             found the string in the command lines.

                sym=stbtyp(mid)
              end if

c             search the list of nodes in the current network for the string.

              do 1400 i=1,cnnnum
                 nnupce=blank
                 do 1395 k=1,12
                   nnupce(k:k)=cnnnam(i)(k:k)
                   do 1392 j=1,26
                     if (upcase(j:j) .eq. nnupce(k:k))
     &                     nnupce(k:k)=lowcas(j:j)
 1392                continue
 1395              continue

                   if (nnupce .eq. strvax) then

c                    found a node name in the current network.

                     nodfnd=.true.
                     intval=i
                     go to 1450
                  end if
 1400         continue
 1450         continue

c             search the network directory for a network name.

              bot=1
              top=netnum
 1460         continue
              if(bot .gt. top) goto 1490
                   mid=(bot + top) / 2

c         upcase the network name characters for the search.

                   do 1475 k=1,namemx
                     nnupce(k:k)=netdnm(mid)(k:k)
                     do 1472 j=1,26
                       if (upcase(j:j) .eq. nnupce(k:k))
     &                     nnupce(k:k)=lowcas(j:j)
 1472                continue
 1475              continue

                   if(strvax .le. nnupce) top =mid-1
```

```
                    if(strvax .ge. nnupce) bot =mid+1
                goto 1460
 1490           continue
                if (bot-1 .gt. top) then

c               found a network in the network directory.

                    netfnd=.true.
                end if

c                 return the position to insert a new networkname if required.

                intva2=mid
             end if
          end if

c         test for a request to print the full menu.

          if ( sym .eq. mensym) then
             if ( menulv .ne. breflv ) then
                 warnvl=65
             else
                 menulv=menllv
                 colptr=lascol+1
             end if
             goto 100
          end if

c     test for a help symbol and not the help command.

          if (( sym .eq. hsym ) .and. (hlpstr .ne. blank))then
             strval=hlpstr
             call listcn(10,1,1,1)
             colptr=lascol+1
             goto 100
          end if

          goto 90000
80000     continue

c     an invalid character was entered which is not recognised by the scanner.

          sym=invsym
          colptr=colptr+1
90000     continue
          return
          end
```

```
      subroutine menupt(menumb,error)
c*************************************************************************
c* Print the menu to the terminal, the level of menu depends on the value of
c*
c*  menulv                          description of the menu printed.
c*
c*    3                                   full menu for naive user
c*    2                                   full prompt line
c*    1                                   batch mode, no prompts or menus
c*************************************************************************
$INSERT CONST2.F77
c*************************************************************************
      integer i,
     &        menumb,
     &        error,
     &        j
      character
     &        status*lnszmx,
     &        temlin*lnszmx
c*************************************************************************
      status=blank
      if ( warnvl .gt. 0) then
           status=wrnlin(warnvl)
      end if
      if ( error .gt. 0)  status=errlin(error)
      if (( menulv .eq. fulllv ) .and. (warnvl .ne. 0)) then
         do 10000 i=menptr(menumb),menptr(menumb+1)-1
            write(termun,fmt(19))menlin(i)
10000    continue

      end if

      if ( menulv .eq. menllv ) then
         menulv=breflv

c              print the menu

         if (( menumb .ge. 1) .and. ( menumb .le. 2)) then
            temlin=menlin(menptr(1)+14)
            menlin(menptr(1)+14)=temlin(1:27) // cnname
         end if
         do 20000 i=menptr(menumb),menptr(menumb+1)-1
            write(termun,fmt(19))menlin(i)
20000    continue
      end if

c        test for an error in the last command.

      if(status .ne. blank) then

c    test for multiple blanks in the status line not at the end to remove.

         do 12000 i=79,1,-1
            if (status(i:i) .ne. ' ') then
```

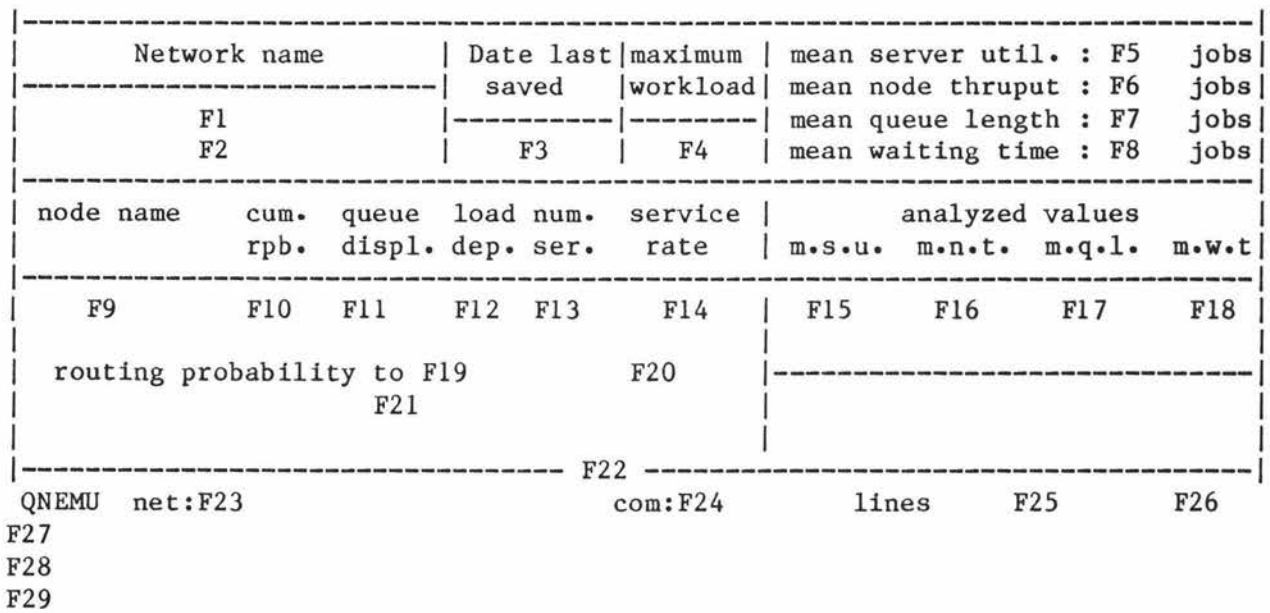
```
               do 11000 j=i,8,-1
                  if (status(j-1:j) .eq. '  ')
     &                  status(j-1:78)=status(j:79)
11000             continue
                  goto 13000
               end if
12000       continue
13000       continue
            write(termun,fmt(19))status
         end if
         write(termun,fmt(19))' '
         write(termun,fmt(19))prmlin(menumb)
         return
         end
```

```
      subroutine newdir(*)
c*************************************************************************
c* Save a null directory into a new DAM file.
c*************************************************************************
$INSERT CONST2.F77
c*************************************************************************
      integer*4 i,j,
     &        recptr,
     &        rctype
c*************************************************************************
  100 continue
      open(UNIT=FILEUN,FILE=DFILNM,STATUS='NEW',ACCESS='DIRECT',
     &        FORM='UNFORMATTED',RECL=FLLGMX,ERR=9000)

c     write the file directory from the opened file.

      rctype=0
      recptr=1
      netnum=0
      freept=101
      write(FILEUN,REC=RECPTR,ERR=9000)rctype,netnum,freept

c        finished writing in the network directory.

      close(FILEUN,ERR=9000)
      goto 20000
 9000 continue

c        some error has occurred therefore PRINT AN ERROR MESSAGE.

      close(FILEUN,ERR=10000)
10000 continue
      return 1
20000 continue
      end
```

APPENDIX F - Fields in the Display Menus

There are a number of Qnemu display menus used to list the network details. These are of a standard format which can be broken up into a number of common fields. In figure F.1 the fields have been identified by the symbols F1 to F29 and below is a brief description of each field.

figure F.1 Display Menu with the Variable Fields Labelled for Reference



Commom Network Display Field Descriptions

F1 :The name of the current network. (e.g. testnet.1)

F2 :Contains **Tagged** if the current network is tagged for removal else blank.

F3 :The date the current network was last saved, DD/MM/YY. Blank if never saved.

F4 :The maximum integer number of jobs the current network can be analyzed for.

F5 :The number of jobs the server utilisations were last analyzed for, to two decimal places.

F6 :The number of jobs the mean node throughputs were last analyzed for, to two decimal places.

F7 :The number of jobs the mean queue lengths were last analyzed for, to two decimal places.

F8 :The number of jobs the mean wait time were last analyzed for to two decimal places.

F9 :The name of the node the details displayed on the line refer to.

F10 :The cumulative routing probability from the node displayed to all other nodes in the current network. This should be 1.00 for the routing probabilities to be valid for analyzing.

F11 :The mnemonic representing the queueing discipline of the server(s) at the node.

F12 :The load dependence indicator, DEP if load dependent else IND.

F13 :The number of identical multiple servers at a PS,FCFS or LCFS node, else inf. if it is an IS node.

F14 :The service rate of the node in jobs/unit of time.

F15 :The mean server utilisation with the number of jobs displayed in F5. See the ANALYZE command option SU for further information.

F16 :The mean node throughput for the number of jobs displayed in F6. See the ANALYZE command option NT for further information.

F17 :The mean queue length for the number of jobs displayed in F7. See the ANALYZE command option QL for further information.

F18 :The mean wait time for the number of jobs displayed in F8. See the ANALYZE command option WT for further information.

F19 :The name of the node the routing probability is to.

F20 :The routing probability value from the node in F9 to the node in F19.

F21 :Contains an error message if the routing probabilities for the node are invalid in some way. Null otherwise.

F22 :The end of information indicator which is as in figure 2.10 until the last line of the information is displayed in the line above and then it is as shown in figure 2.9.

F23 :The current network name, or the library file name.

F24 :The Qnemu command currently being executed.

F25 :The line numbers range of the information text currently on display.

F26 :The total number of lines in the information text in total.

F27 :The display subcommand prompt line if the information takes up more than a single display menu, else blank. See figure 2.12.

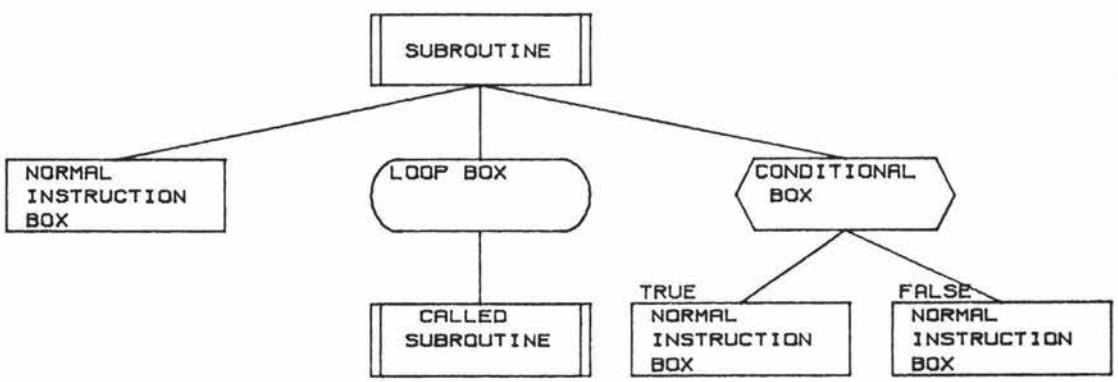
F28 :The prompt line if the information can be displayed in a single display menu, else null. See figure 2.11.

F29 :The user input line.

APPENDIX G - Structure Diagram Code Summary

The structure diagrams used throughout the description of the implementation of QNEMU have a standard structure. The name of the subroutine which the high level structure diagram describes is always contained in the top box. Similar boxes elsewhere in the structure diagram represent calls made on those subroutines. Boxes with angled sides represent conditional boxes and boxes with rounded sides represent loops. (See figure G.1 below.)

figure G.1 Sample Stucture Diagram



APPENDIX H - Performance Evaluation

H.1 Introduction

The performance evaluation of systems is of importance in a large number of areas, including computing, communication, industry and transportation. Many require similar types of information and the techniques which can be used are basically the same, with modifications to suit the particular system under study.

Prediction is the estimation of performance measures before the system is in existence, and measurement is the recording of the performance data of existing systems.

The need for system prediction exists both in the initial design stages, when decisions on the actual systems to build or purchase are required, and in the ongoing study of how a system can be changed to improve its performance. Measurement can be undertaken on existing systems in all stages of its lifetime, to hopefully enable decisions to be made which will improve the performance of the system.

Our main interest is in the performance of computer systems and in gereral we shall restrict our discussion to that area, however we stress that the modelling techniques to be described have wider applicability.

In the initial computer system design, prediction allows decisions as to what choice of hardware (e.g. size and number of devices), operating system components (e.g. service disciplines) and other software decisions (e.g. application packages) would give the best performance. Performance evaluation studies are also of importance when computer manufacturers are designing new products. They are used to predict the performance of the new products for specific applications and to estimate the suitability of the new products in meeting the requirements of potential users. Suppliers of computer systems are also required to submit configuration proposals to prospective purchasers. Frequently alternative configurations are measured or modelled to ascertain whether or not they can meet the needs of the prospective purchasers [Triv 80].

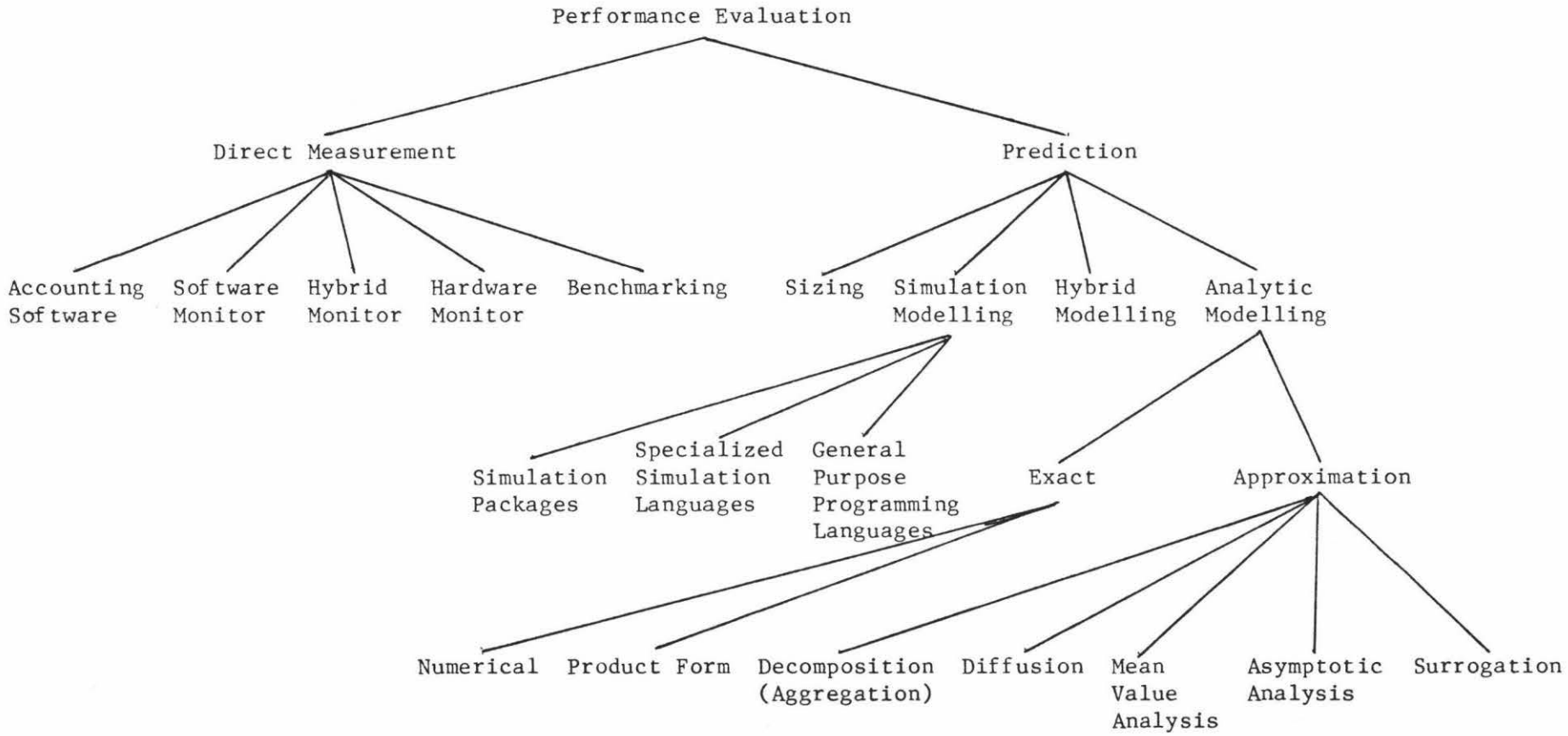
Once installed a computer system requires what is generally known as tuning to approach its full potential under its particular workload. Frequently computer performance is left largely to intuition and experience rather than to objective decisions based on performance figures. This in part is due to the lack of economical and reliable performance tools, but also to the lack of a systematic procedure for the performance evaluation task. A number of publications have provided useful guides to performance evaluation, including a selection of papers in [Semi 79].

There are a number of different computer performance techniques and tools in use to varying degrees. They include accounting software, software monitors, hardware monitors, hybrid monitors, sizing,

simulation and analytic models.

The different techniques can be classified by whether they directly measure a system or whether they predict the performance measures of interest (figure H.1).

figure H.1 Computer Performance Evaluation Techiniques



H.2 Measurement

Measurement techniques have two major drawbacks, but have the benefits of accuracy and credibility if used correctly. The first drawback is that they cannot be used on a system which is not operational, and the second is that they are complex activities which require considerable expense in both human time and machine costs.

Measurement data can be obtained in three ways, namely

- accounting software
- software monitors
- hardware monitors

Accounting software can provide a number of performance measures which can be used to evaluate the overall performance of the system [Cox 79], and can be viewed as a type of software monitor.

Software monitors are specialised pieces of software running very close to, or inside, the operating system of the computer, and periodically sample parts of the system collecting performance data.

Hardware monitors are specialised hardware devices that use probes attached to the internal circuitry of the computer system to sample the state changes at these points.

APPENDIX H

Hybrid monitors are combinations of software and hardware monitors which allow statistical data to be collected from both software running in the system and from probes attached to the hardware. Normally they have a specialised computer both controlling the data collection and the storage of data. Both software monitors and to a lesser extent hybrid monitors have a problem in that they actually use the resources of the system they are monitoring, and therefore they will influence the performance measures obtained. Hardware monitors do not suffer from this problem, but on the other hand cannot measure some useful performance metrics, such as the utilization of operating system modules.

Benchmarking is a widely used technique which attempts to obtain performance measures of a system by running an artificial, but hopefully representative, workload. The major problem with benchmarking is the difficulty in obtaining benchmarks that accurately represent the workload of a system. Truly representive benchmarks are usually extremely expensive to set up, in terms of machine time, manpower and time costs, and at least one manufacturer (the main user group of benchmarking) has set up fixed "Corporate Benchmarks" to run on both their own and other suppliers equipment for comparative measurements.

H.3 Prediction

In a large number of performance studies the systems being evaluated do not exist to the degree that direct measurement techniques can be carried out. In such cases prediction techniques must be used to estimate the performance values of the system. There are four general prediction techniques as shown in figure H.1, namely:

- Sizing
- Simulation Modelling
- Hybrid Modelling
- Analytic Modelling

Sizing is a technique largely used by manufacturers when carrying out performance evaluations on proposed systems. This technique normally involves the calculation of a large number of performance metrics by simple calculations. With simulation, hybrid and analytic modelling techniques, networks of queues are usually involved.

H.4 Models of Queueing Networks

A queueing network is a system of interconnected servers and queues. A server (resource, device, processor, shop assistant, bank teller) services a job (customer, task, program, vehicle) which must wait for service in the queue associated with that server. For ease of exposition, a server and its queue will be called a node. Upon completion of service at a device, a job leaves that device's node and enters the queue of another device, there to await service. Job(s) receiving service still remain in the server's queue.

In queueing network models, jobs move from node to node in a probabilistic fashion, defined by a set of routing probabilities. The probability that a job enters node j upon completion of service at node i is the routing probability Pij. The amount of time a customer is served at a device is governed by the service distribution for that device.

The manner in which customers enter, wait, and are served in a queue is defined by its queueing discipline (eg. first come first served, FCFS). This is also known as the scheduling discipline. On arrival of a job the queueing discipline is used to decide where the arriving job should be positioned in the queue. If it is decided it should be at the front of the queue the queueing discipline is also used to decide whether to swap it with the job being served or not.

A queueing network is closed if jobs may neither enter nor leave the network, thus making the number of jobs in the network a constant. A queueing network in which a job enters the network precisely when another job exits can be considered closed, as the number of jobs in the system appears constant. The average number of jobs is called the workload (degree of multiprogramming) of a system.

A queueing network is open if the jobs enter from external sources and exit to a sink or sinks. An open network may be connected such that a job can receive service many times or only once at any given server in its traversal of the network, before finally leaving the network.

H.5 Simulation

Simulation is a popular technique which uses a computer program to model the computer system being evaluated. The simulation program can vary in detail from a very abstract level to a level of detail very close to the actual system. Taken to the extreme the actual system itself can be thought of as the ultimate model.

Simulation techniques can be applied to a wide range of systems without the limitations which exist in analytic techniques.

However, there are three main problems with using simulation techniques to evaluate systems. The first is the expense of constructing a simulation program; the second is the computational expense of running the program; and, lastly, the requirement to statistically analyze the results (as with any of the techniques). In terms of simulation software in general, the specialized packages (e.g. CASE [TESD], SAM [ADR 75]) operate at a fairly high level of abstraction that use interconnected devices as the basic entities. Below these are the simulation languages, which allow the user to write procedures to simulate the various devices in the system (e.g. SIMSCRIPT, SIMULA, GPSS, GASP IV). The lowest level is the general pupose programming languages. The higher the level, the simpler it should be to model a system. However, conversely the level of accuracy possible should decrease, as should the efficiency of execution attainable. A detailed comparison of the different simulation techniques with advantages and disadvantages can be found in [Kreu 79, Dock 79].

In summary simulation can provide useful evaluations of a wide range of systems but can be very time consuming and costly to run.

H.6 Hybrid

The Hybrid modelling approach uses a combination of simulation modelling and analytic modelling techniques. Parts of the system or subsystems are modelled with analytic techniques, and then the rest is modelled using simulation [Schw 78]. Potentially hybrid monitoring can encompass the best features of both simulation and analytic modelling whilst minimising on their unsatisfactory features.

H.7 Analytic Models

There are a number of different types of analytic models but they are all characterised by the production of deterministic results. Analytic models are based on a number of numerical methods which apply to even the most complex of system models. The numerical methods are of limited practical use for the majority of models as they require the solution of sets of linear equations, the number of which becomes very large even for relatively small models [Saue 81]. The numerical methods are based on traditional Markovian queueing network theory [Klei 75]. A notable example of a package which uses numerical methods is the Recursive Queue Analyzer (RQA) developed at the University of Michigan [Wall 66]. More recent work has been carried out by Stewart [Stew 74, Stew 77].

The input parameters to analytic models include the number of jobs in the system, the service distribution, the mean service rate of the service distribution, and the mean number of visits a job makes to each node (or more commonly the routing probabilities). The evaluated performance values normally include the system throughput, and at each node in the system the throughput, the mean queue length, the mean server utilisation and the mean wait time of the jobs (which includes the queueing time and service time).

H.7.1 Product Form

The form of analytic modelling of most interest deals with product form models of systems, as these have computationally efficient algorithms. Only a limited subset of queueing networks can be modelled by product form, but these include a number of the more useful models, such as the central server model and the time sharing models.

A product form model is one in which a solution to the equilibrium distribution of network state probabilities exists in the form:

$$p(N) = \frac{p1(N1) \times p2(N2) \times \cdots pm(Nm)}{G}$$

The state of a network consisting of m nodes is defined if the state of each node is known and can be represented by the vector N = {N1, N2, . . . , Nm}. The Ni of each node is the number of jobs at that node. The pi(Ni) is the probability of finding node i in state Ni, and the p(N) is the probability that the network is in state N. The G is the normalizing constant that makes the sum of the state probabilities sum to one [Gord 67].

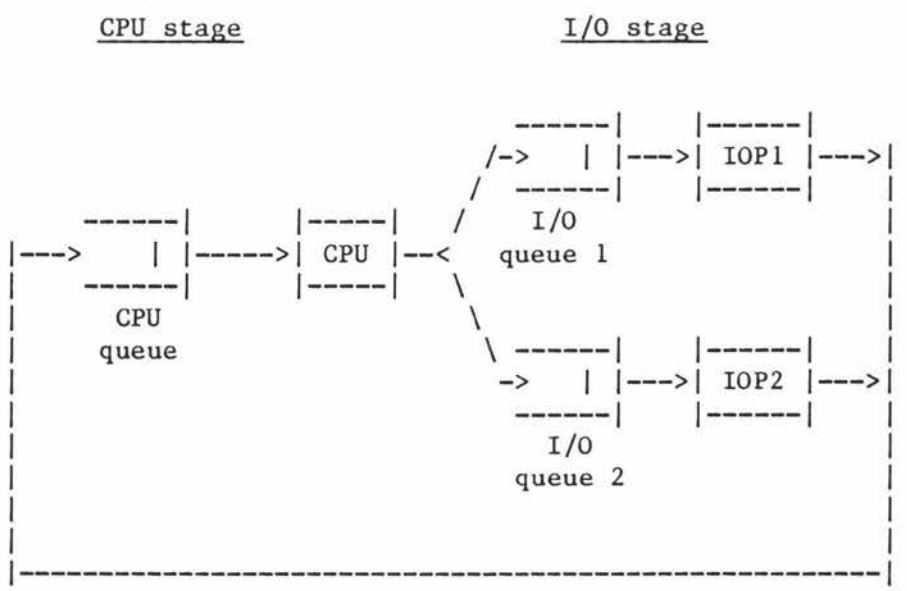
The earliest work on product form models involved the study of open networks of M/M/1 nodes [Jack 63]. An M/M/1 node has a negative-exponential interarrival time distribution (M for the Markov memoryless property), negative-exponential service

distribution, and one server. These models were restricted to homogeneous jobs and FCFS queueing disciplines. This work was followed by the special case simplifications of closed product form networks and queue length dependent service rates [Gord 67]. These results were used to successfully model multiprogramming systems using the central server model [Buze 71a]. The development of fast computational algorithms was the next major step which provided the ability to solve closed product form models efficiently [Buze 71b, Buze 73]. The next advance came when the models were extended to allow a number of different queueing disciplines, multiple classes of jobs in the workload, and non-exponential distributions for open, closed and mixed networks [Bask 75]. The queueing disciplines were extended from FCFS to include Processor Sharing (PS), Last Come First Served (LCFS), Preemptive Resume (LCFSPR) and Infinite Server (IS). A number of validation exercises and variations to the computational algorithms have been carried out since [Chiu 75, Saue 79b, Brue 80]. One of the main problems with these models is the number of assumptions which have to be made to model a system.

A commonly used closed model is the Central Server Model (CSM). This model is characterised by a single CPU with a number of I/O devices in parallel connected in a network (see figure H.2). A job in the system cycles in an infinite loop alternating service at the CPU with service at one of the I/O devices. The CSM model can be used to model a special case of open system, namely whenever a job departs the system another enters so as to keep the number of jobs

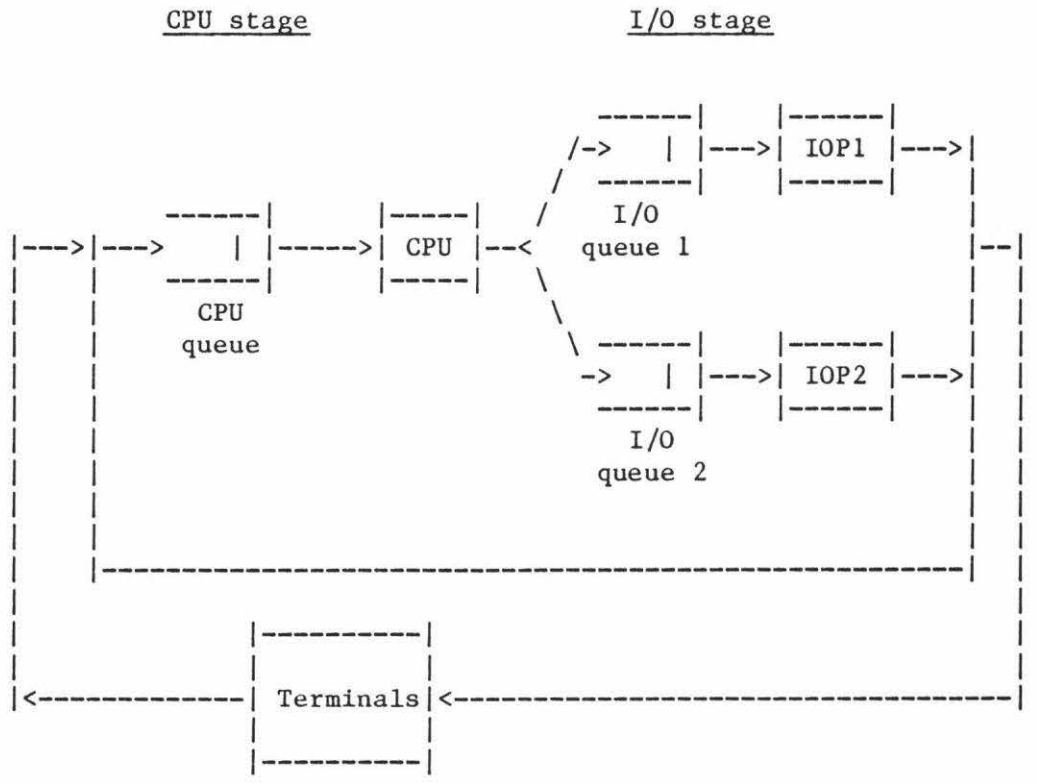
constant. This also assumes that the departing and arriving jobs have the same properties in terms of routing probabilities and service distribution at the nodes.

figure H.2 Central Server Model



Interactive systems can also be modelled using a variation of the CSM, in which another node is added to represent the terminals in the system. The terminals are normally modelled collectively as an infinite server node as it is assumed an interactive job in the system will always have a user at the terminal waiting whenever the job requires service at that device [Brow 75] (see figure H.3).

figure H.3 Central Server Model with Terminals



A number of performance studies also based on the CSM, have been carried out to investigate memory contention under both paged and non-paged memory management policies [Brow 77]. One well documented evaluation is of the VM/370 system in [Bard 78].

H.7.2 Operational Analysis

The robustness of the stochastic product form networks, in terms of the simplifying assumptions made to obtain results, led Buzen to develop a methodology based on empirical data. This he called Operational Analysis [Denn 78]. A number of the operational analysis requirements are consistent with stochastic modelling assumptions as the following taken from [Denn 78] indicate:

- All quantities must be precisely measureable and all assumptions stated so as to be directly testable. The validity of results should depend only on assumptions which can be tested by observing a real system for a finite period of time.

- The system must be flow balanced, i.e. the number of arrivals at a given device must be (almost) the same as the number of departures from that device during the observation period.

- The devices must be homogeneous, i.e. the routing of jobs must be independent of local queues and the mean time between service completions must not depend on the queue lengths of other devices.

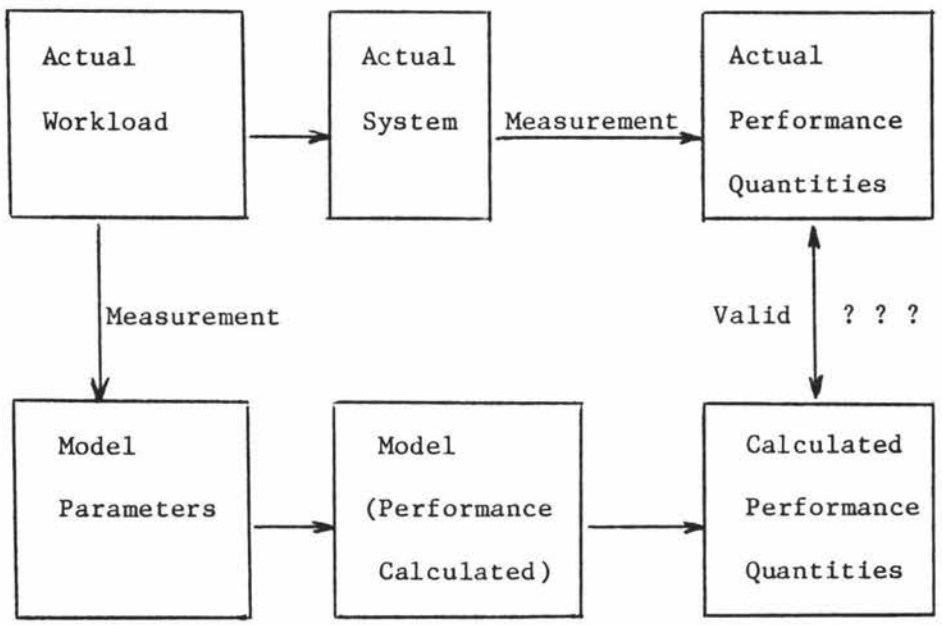
H.7.3 Approximate

A number of approximation methods have also been developed which allow models to be solved approximately when they do not meet the assumptions for a product form solution, or when the exact solutions are too expensive to obtain. The major three approximation methods are aggregation [Chan 75a, Chan 75b], diffusion [Koba 74] and mean value analysis [Brue 80]. Others include asymptotic analysis [Zaho 82] and surrogation [Jaco 82]. Further reading on most of these techniques can be found in [Chan 78].

H.8 Validation

Validation is required for all models. Typical steps in the validation of a model are described in figure H.4 [Denn 78]. A workload is run on the real system over a specified observation period and the performance measures gathered compared with those from the prediction. The model in the prediction can be assumed to provide reproducable results if a close comparison is obtained over a number of different observation periods. Once validated the model can be used with confidence in the prediction of future behaviour of the actual system. The measurement procedures for obtaining the actual performance metrics of a computer system can be found in [Rose 78]. Examples of QNEMU validations appear in Chapter 4.

figure H.4 Typical Validation Scheme



References and Bibliography

Alle 80 - Allen, A.O.

Queueing Models of Computer Systems

Computer, April 1980

ADR 75 - Applied Data Research

System Analysis Machine: Concepts and Facilities

Applied Data Research, Inc., 1975

ASQ 75 - MANUAL

Users' manual for the ASQ (Analytic Solution of Queues)

Information Research Associates, 1975

Bard 78 - Bard, Y.

The VM/370 Performance Predictor

Computer Survey, Vol 10, No. 3, September 1978

Bask 75 - Baskett, F; et al.

Open, Closed, and Mixed Networks of Queues with Different Classes of Customers

Journal of the ACM, Vol. 22, April 1975

Barr 71 - Barron, D.W; and Jackson, I.R.

The Evolution of Job Control Languages

Software - Practice and Experience, Vol. 2, 1972

Brow 82 - Brown, J.W.

Controlling the Complexity of Menu Networks

Communications of the ACM, Vol. 25, No. 7, July 1982

Brow 83 - Brown, P.J.

Error Messages: the Neglected Area of the Man/Machine Interface

Communications of the ACM, Vol. 26, No. 4, April 1983

Brow 75 - Browne, J.C. et al.

Hierarchical Techniques for Development of Realistic Models of Complex Computer Systems

Proc. IEEE 63, 6, June 1975

Brow 77 - Brown, R.M. et al.

Memory Management and Response Time

Communications of the ACM Vol. 20, No. 3, March 1977

Brue 80 - Bruell, S.C.; and Balbo, G.

Computational Algorithms for Closed Queueing Networks

Elsevier North Holland, New York, 1980

Buze 71a - Buzen, J.P.

Analysis of System Bottlenecks Using a Queueing Network Model

Proc. ACM SIGPOPS Workshop System Performance Evaluation, ACM, New York, 1971

Buze 71b - Buzen, J.P.

Queueing Network Models of Multiprogramming

Ph.D. Thesis, Division of Engineering and Apllied Science, Harvard University, Cambridge, Mass., 1971

Buze 73 - Buzen, J.P.

Computational Algorithms for Closed Queueing Networks with Exponential Servers

Communications of the ACM Vol. 16, No. 9, September 1973

Buze 78 - Buzen, J.P.

A queueing Network Model of MVS

Computer Survey Vol 10, No. 3, September 1978

Buze 80 - Buzen, J.P.; and Denning, P.J.

Measuring and Calculating Queue length Distributions

Computer, April 1980

Chan 75a - Chandy, K.M.; et al.

Parametric Analysis of Queueing Networks

IBM Journal of Research and Development, January 1975

Chan 75b - Chandy, K.M.; et al.

Approximate Analysis of General Queueing Networks

IBM Journal of Research and Development, January 1975

Chan 77 - Chandy, K.M.; et al.

Product Form and Local Balance in Queueing Networks

Journal of the ACM, Vol 24, No. 2, April 1977

Chan 78 - Chandy, K.M.; and Sauer, C.H.

Approximate Methods for Analyzing Queueing Network Models of Computing Systems

Computer Survey Vol 10, No. 3, September 1978

Chan 82 - Chandy, K.M.

Linearizer: A Heuristic Algorithm for Queueing Network Models of Computing Systems

Communications of the ACM, Vol. 25, No. 2, February 1982

Chan 83 - Chandy, K.M.; and Martin, A.J.

A Characterization of Product-Form Queueing Networks

Journal of the ACM, Vol. 30, No. 2, April 1983

Chan 75 - Chang, J.H.

Terminal Response Times in Data Communications Systems

IBM Journal of Research and Development, May 1975

Chiu 75 - Chiu, W.W.; et al.

Performance Analysis of a Multiprogrammed Computer System

IBM Journal of Research and Development, May 1975

Chow 75 - Chow, W.M.

Central Server Model for Multiprogrammed Computer Systems with Different Classes of Jobs

IBM Journal of Research and Development, May 1975

Cox 79 - Cox B G

Performance Evaluation Using Existing Accounting Information

presented at the N.Z.C.S. Seminar on Computer Performance Evaluation, Waikato University, 1979

Denn 78 - Denning, P.J.; and Buzen, J.P.

The Operational Analysis of Queueing Network Models

Computer Survey Vol 10, No. 3, September 1978

Dock 79 - Docker, T.W.G.

Some Aspects of Computer Simulation Modelling

presented at the N.Z.C.S. Seminar on Computer Performance Evaluation, Waikato University, 1979

Ferr 81 - Ferrari, D.

A Hybrid Measurement Tool for Minicomputers

Computer Performance Evaluation, North Holland 1981

Ferr 83 - Ferrari, D.

Measurement and Tuning of Computer Systems

Prentice Hall, Inc., Englewood Cliffs, 1983

Full 71 - Fuller S.; et al.

Measurement and Analysis of a Multiprogrammed Computer System

presented at the IEEE Workshop on Performance Evaluation, Argonne National Laboratory, 1971

Gord 67 - Gordon, W.J.; and Newell, G.F.

Closed Queueing Systems with Exponential Servers

Operations Research, Vol 15, No. 2, April 1967

Grah 78 - Graham, G.S.

Queueing Network Models of Computer System Performance

Computer Survey Vol 10, No. 3, September 1978

Hard 80 - Hardy, T. Jr.

The Syntax of Interactive Command Languages: A Framework for Design

Software - Practice and Experience, Vol. 12, 67-75 1982

Herz 75 - Herzog, U.; et al.

Solution of Queuing Problems by a Recursive Technique

IBM Journal of Research and Development, May 1975

Iran 71 - Irani, K.B.; and Wallace, V.L.

On Network Linguistics and the Conversational Design of Queueing Networks

Journal of the ACM, Vol. 18, No. 4, October 1971

Jack 63 - Jackson, J.R.

Job Shop-like Queueing Systems

Management Sci., Vol 10,No. 1, 1963

Jaco 82 - Jacobson, P.A.; and Lazowska, E.D.

Analyzing Queueing Networks with Simultaneous Resource Possession

Communications of the ACM, Vol. 25, No. 2, February 1982

Klei 70 - Kleinrock, L.

Analytic and Simulation Methods in Computer Network Design

AFIPS Conference Proceedings, SJCC, 1970

Klei 75 - Kleinrock, L.

Queueing Systems Volume 1: Theory

John Wiley, New York, 1975.

Klei 76 - Kleinrock, L.

Queueing Systems Volume 2: Computer Applications

John Wiley, New York, 1976.

Koba 74 - Kobayashi, H.

Application of the Diffusion Approximation to Queueing Networks

I: Equilibrium Queue Distributions

Journal of the Association for Computing

Koba 77 - Kobayashi, H.; and Konheim A.G.

Queueing Models for Computer Communication System Analysis

IEEE Transactions on Communications, January 1977

Koba 78 - Kobayashi, H.

Modeling and Analysis: An Introduction to System Performance Evaluation Methodology

Addison-Wesley Publishing Company, Inc., 1978.

Koen 60 - Koenigsberg E.

Finite Queues and Cyclic Queues

Operations Research, Vol. 8, 1960

Kreu 79 - Kreutzer, W.

Computer System Modelling and Simulation

presented at the N.Z.C.S. Seminar on Computer Performance Evaluation, Waikato University, 1979

Lam 83 - Lam, S.S.; and Luke Lien, Y.

A Tree Convolution Algorithm for the Solution of Queueing Networks

Communications of the ACM, Vol. 26, No. 3, February 1983

Larm 81 - Larmouth, J.

Fortran 77 Portability

Software-Practice and Experience, Vol. 11, 1981

Ledg 80 - Ledgard, H.; et al.

The Natural Language of Interactive Systems

Communications of the ACM, Vol. 23, No. 10, October 1980

Leun 82 - Leung, C.H.C.

A Simple Model for the Performance Analysis of Disc Storage Fragmentation

The Computer Journal, Vol. 25, No. 2, 1982

Ling 80 - Ling, R.F.

General Considerations on the Design of an Interactive System for Data Analysis

Communications of the ACM Vol. 23, No. 3, 1980

Lips 77 - Lipsky L.; and Church, J.D.

Applications of a Queueing Network Model for a Computer System

Computer Survey Vol. 9, No. 3, September 1977

McCr 82 - McCrea, R.C.; and Akers F.A.

Data-communication Networking: Components, Structure and Performance Analysis

Computer Performance, Vol. 3, No. 3, September 1982

Mads 78 - Madsen, J.

CCL-A High-level Command Language

Software - Practice and Experience, Vol. 9, 1970

Munt 78 - Muntz, R.R.

Queueing Networks: A Critique of the State of the Art and Directions for the Future

Computer Survey Vol 10, No. 3, September 1978

Norm 83 - Norman, D.A.

Design Rules Based on Analyses of Human Error

Communications of the ACM, Vol. 26, No. 4, April 1983

NAG 83 - Manual

NAG FORTRAN Library Manual, Mark 10

Numerical Algorithms Group, United Kingdom, 1983

Oldf 82 - Oldfield, B

QNEMU Computational Algorithm Implementation

Performance and Evaluation Group, Technical Report, Massey University, N.Z., 1982

Pars 78 - Parsons, I.T.

A Support System for Interactive Languages

Software - Practice and Experience, Vol. 9, 1979

Prim 80 - MANUAL

The Fortran 77 Reference Guide IDR4029

Prime Computer, Inc., Massachusetts 01701, U.S.A., 1980.

Prit 74 - Pritsker, A.A.B.

The GASP IV Simulation Language

John Wiley and Sons, Inc., 1974

QNEM 84 - MANUAL

QNEMU User Guide

Performance Evaluation Group, Massey University, N.Z., 1984. (In preparation)

Reis 75 - Reiser, M.; and Kobayashi H.

Queueing Networks with Multiple Closed Chains: Theory and Computational Algorithms

IBM Journal of Research and Development, May 1975

Reis 80 - Reiser, M.; and Lavenberg, S.S.

Mean-Value Analysis of Closed Multichain Queueing Networks

Journal of the ACM, Vol. 27, No. 2, April 1980

Rose 78 - Rose, C.A.

A Measurement Procedure for Network Models of Computer Systems

Computer Survey Vol 10, No. 3, September 1978

Salf 83 - MANUAL

The University of Salford FTN77 Reference Manual

University of Salford, England, 1983.

Sand 78 - Sandewall, E.

Programming in an Interactive Environment: the "Lisp"

Experience

Computing Surveys, Vol. 10, No. 1, March 1978

Saue 75 - Sauer, C.H.; and Chandy, K.M.

Approximate Analysis of Central Server Models

IBM Journal of Research and Development, May 1975.

Saue 79a - Sauer, C.H.; and Macnair, E.A.

Queueing Network Software for Systems Modelling

Software-Practice and Experience, Vol. 9, 1979

Saue 79b - Sauer, C.H.; and Chandy, K.M.

The Impact of Distributions and Disciplines on Multiple Processor Systems

Communications of the ACM, Vol. 22, No. 1, January 1979

Saue 80 - Sauer, C.H.; and Chandy, K.M.

Approximate solutions of Queueing Models

Computer, April 1980

Saue 81 - Sauer, C.H.; and Chandy K.M.

Computer Systems Performance Modelling

Prentice-Hall, Inc., Englewood Cliffs, 1981

Saue 82 - Sauer, C.H.; et al.

The Research Queueing Package: Past, Present, and Future

National Computer Conference, 1982.

Saur 83 - Sauer, C.H.

Computational Algorithms for State-Dependent Queueing Networks

ACM Trans. on Computer Systems, Vol. 1, No. 1, February 1983

Schw 78 - Schwetman, H.D.

Hybrid Simulation Models of Computer Systems

Communications of the ACM, Vol. 21, No. 9, September 1978

Schn 82 - Schneiderman, B.

The Future of Interactive Systems and the Emergence of Direct Manipulation

Technical Report, Dep. of Comp. Sc., Univ. of Maryl., 1982

Semi 79 - SEMINAR WORKBOOK

Computer Performance Evaluation

N.Z.C.S. Seminar on Computer Performance Evaluation, Waikato University, 1979

Shum 77 - Shum, A.W.

Queueing Models for Computer Systems with General Service Time Distributions

Ph.D. Thesis, Aiken Computation Laboratory, Harvard University, Cambridge, Mass., 1977

Spra 80 - Spragins, J.

Analytical Queueing Models

Computer, April 1980

Stew 74 - Stewart, B.A.

Markov Analysis of Operating System Techniques

Ph.D. Thesis, Queen's University of Belfast, July, 1974

Stew 77 - Stewart, B.A.

A New Approach to the Numerical Analysis of Markovian Models

Computer Performance, North Holland Publishing Company, 1977

Suri 83 - Suri, R.

Robustness of Queueing Network Formulas

Journal of the ACM, Vol. 30, No. 3, July 1983

Terp 83 - Terplan, K.

Communications systems performance and management

Computer Performance, Vol. 4, No. 1, March 1983

Tesd - Tesdata Corporation

CASE Technical Description

Tesdata Systems Corporation

Tows 80 - Towsley, D.F.

Queueing Network Models with State-Dependent Routing

Journal of the ACM, Vol. 27, No. 2, April 1980

Triv 80 - Trivedi, K.S.; and Kinicki, R.E.

A Model for Computer Configuration Design

Computer, April 1980

Wall 66 - Wallace, V.L.; and Rosenberg, R.S.

RQA-1, the Recursive Queue Analyzer

Tech. Rep. 2, Dept. of Elec. Eng., Univ. of Mich., Ann Arbor, February 1966

Wall 82 - Wallis, P.J.L.

The Preparation of Guidelines for Portable Programming in High-level Languages

Computer Journal, Vol. 25,No. 3, 1982

Wong 78 - Wong, J.W.

Queueing Network Modelling of Computer Communication Networks

Computer Survey Vol 10, No. 3, September 1978

Zaho 82 - Zahorjan, J.L.

Balanced Job Bound Analysis of Queueing Networks

Communications of the ACM, Vol. 25, No. 2, February 1982