CLM-PDN 2/74

A GENERAL PURPOSE QUEUEING MECHANISM

by

D A Fox and T Lang

ABSTRACT

The queueing mechanism is designed to support any kind of batch processing facilities in which tasks have to be queued. It was designed for a small machine, to be simple, flexible and yet provide comprehensive facilities for job control and management.

U.K.A.E.A. Research Group

Culham Laboratory

Abingdon

Berkshire

May 1974

CONTENTS

1. Introduction

A general purpose queueing system[1] is an important component in an intelligent satellite system since many different tasks have to be performed by several different processes. These tasks are processed at different rates and so have to be placed on queues until the process is ready to handle them. A queueing system is needed to administer these queues.

This article describes the design and implementation of such a system.

2. Design Criteria

1. The queueing mechanism is to work on a small (i.e. 16 bit) machine equipped with disc backing store.
2. The software should occupy as little core store as possible.
3. Portable - so that coding can be moved to different machines with the minimum of effort.
4. The system should be as general as possible and be independent of the nature of any tasks handled by the system.

The queueing system should also have the following facilities:

5. Spooling - the intermediate buffering of tasks on backing store. This is required as different processes will service and generate tasks at different rates. This facility also allows tasks to be generated even if the service breaks down (e.g. jobs for transmission to remote machines can still be added to the appropriate queues).
6. Fail-soft - the ability to restart the system after system breakdowns without loss of data.
7. Resource Pooling - sharing tasks according to loading e.g. several line printers handling the same print queues.

This provides a continued, though degraded, service in the event of individual break downs.

8. Queue selection - selecting tasks for service either by priority or "round robin", e.g. printing short files before long ones.
9. Operator control - allowing the operator to control and monitor the system loading.
10. Accounting - simple accounting and statistics should be available e.g. number of entries serviced by a process.

3. System Design

The queueing system must be capable of supporting any kind of "batch processing" requirement which involves input activities putting entries on queues, and processing activities subsequently removing them.

3.1 Basic Philosophy

To indicate the flexibility of the queueing mechanism, it is convenient to first consider the basic philosophies underlying its design. A queue is made up of a list of filenames; each file is presumed to hold data which requires processing by some "server". Also associated with each file name is the name of the user who placed the entry on the queue, and a set of markers which can be used to convey additional information to the queue server (e.g. to indicate that the file is to be deleted when its contents have been processed).

A queue is held in a backing store file. The file is of fixed length (i.e. fixed at system generation time) and used as a circular buffer. All entries are treated as first-in-first-out. Priority scheduling is achieved by planting entries on different queues. (This approach simplifies queue maintenance, and minimises the number of backing store accesses required to add and remove

queue entries, whilst still leaving a good level of operational convenience).

The information describing all the queues (e.g. pointers to the head and tail of each queue) are held in a single compact data block. (This is convenient where one server needs to inspect several queues). Also held in the queue control block is a queue/server matrix. This has a row for each queue and a column for each server. If there is a non-zero entry at the intersection of a row with a column, then the corresponding server is required to process the indicated queue. In the simplest case there is just one server to each queue. Where one server serves several queues, then the next queue to be served can be chosen in several ways e.g. on a round robin basis, or according to a priority rating. In the latter case the entries in the column of the queue/server matrix indicate the relative queue priorities. It is also possible for several servers to serve just one queue - this is indicated by several entries in the same row of the queue/server matrix.

Operator control of the system is effected by a program which accesses the queue control block. This program will provide facilities to

(a) list and alter the queue/server matrix

(b) list queue details, clear queues, reset queues (for warm restarts)

(c) list server details, turn servers off.

Whenever a new server process is activated by the operator, it will first ask from which column in the queue/server matrix it is to take its instructions, and provided no other server is already using the same column, it can proceed accordingly.

The queueing system as described above allows the system manager great flexibility in the control of the overall system. Queues

may be utilised to store tasks which are held up by problems elsewhere in the computing system - e.g. mainframe or communication line failures. Less important work can be held on queues which are only serviced in off-peak periods. Tasks can be split into streams according to relative priorities. These streams can be interleaved, so that lower priorities are only served whilst no higher priority work is waiting. (The utilisation of high priority queues can be rationed via the use of the accounting system). Work for different RJE links may be interleaved on the same Satellite peripherals (i.e. a card reader/line printer pair). In the case of very high transmission rates, several sets of peripherals may be set to support one line. In the case of very low transmission rates, one set of peripherals could share its load over several communication lines.

The general purpose queueing system also maintains some accounting statistics. The queue control block holds individual totals for the number of jobs processed from each queue and by each server. One of the facilities offered by the operator control program is the ability to list these totals (optionally resetting the entries to zero at the same time).

3.2 Basic Procedures

The mechanism has been designed as a set of procedures, two to be called by generating activities which plant entries on queues, and three to be called by serving activities which remove entries from queues.

The means of driving these procedures from calling processes is left to the implementor. The possibilities are:

(a) Embed the procedures in the calling processes

(b) Plant the procedures in a separate segment, but still as part of the calling process

(c) Embody the procedures in a separate activity, called via some message mechanism of the supervisor.

It is likely that (c) should be preferred.

3.2.1 Planting Entries on Queues

Procedure CHECKQUEUE can be called by any process which knows that it is about to generate a new queue entry. Its primary purpose is to confirm that a specified queue is available for use and that it has room for a further entry. The procedure is actually a function, and the success or failure of the check is indicated by the value it returns. (All queueing procedures use this technique.) It may be used to give the generating process early warning of any problems, but does not guarantee that when it actually attempts to plant an entry on the queue, using procedure ADD TO QUEUE, it will definitely be successful.

A process which is going to generate a new queue entry may well need to create a new file to hold the material of that entry. This new file has to be given a name which is unique within the file system. To assist in generating this name, CHECK QUEUE will optionally return, on request, a set of four alphanumeric characters. These characters may be used directly to form the new file name; however it is anticipated that in general they will be appended to the end of the 8 character user name to give a more informative file name. The 4 character label will be "incremented" upon each generation.

A flowchart for CHECK QUEUE is given in Appendix B (together with the flowcharts for ADD TO QUEUE and SERVE QUEUES).

Procedure ADD TO QUEUE is used to plant an entry on a queue. The caller must specify:

queue number
file name (of file containing material to be processed)
user name (of user initiating the entry)
marker word (see section 3.3)

Upon return, if the entry has been successfully appended to its queue, the caller is informed of:

queue status (i.e. whether or not it is currently being served)
position of entry in queue
tag number

The tag number identifies the physical position of the entry in the queue file. This tag number may subsequently be used to service status queries or a cancellation request by direct access, and obviates the need to search the whole queue to identify a particular entry.

3.2.2 Removing Entries from Queues

When a serving process first starts up, it should identify itself to the queueing mechanism via procedure SERVE QUEUES. The caller must specify its server number - i.e. the column in the queue/server matrix from which it will take its instructions. The procedure checks the validity of the server number and that no other server is operating with the same number.

To obtain details of the next entry requiring service, procedure GET ENTRY should be used. The caller must again specify his server number. The procedure will return all the details of the next entry requiring service - i.e.

file name
user name
marker word

If there is no entry requiring service, the calling process will be suspended until one becomes available. The flowchart for GET ENTRY is given in Appendix B.

Procedure GET ENTRY marks the corresponding queue entry as being serviced, but does not remove it from the system. When service is completed, removal may be effected by a call on REMOVE

ENTRY. The caller need only specify his server number. The flowchart for this procedure is given in Appendix B.

3.3 Data Structure

The format of the queue control block is as follows (the CLSD[2] manifest constant/integer names are given in brackets):-

Word	0	number of queues (NQUEUES)
"	1	pointer to start of queue descriptions (QDESCRIPTOR)
"	2	number of servers (NSERVERS)
"	3	pointer to start of server descriptions (SDESCRIPTOR)
"	4	pointer to start of queue/server matrix (QSMATRIX)
"	5-8	unique 4 character alphanumeric label (one character per word)

The queue descriptions are stored sequentially, each description consisting of 9 words

Word	0	pointer to start of reserved queue area (QSTART)
"	1	" " end " " " " (QEND)
"	2	pointer to head of queue (QHEAD)
"	3	" " tail " " (-1 = queue full) (QTAIL)
"	4	number of entries (QENTRIES)
"	5	tag number for next entry (NEXTAG)
"	6	queue status (-1 = do not use (QSTATUS) 0 = may be used)
"	7	number of entries serviced since accounting last performed (QSERVICE)
"	8	number of entries queue will hold (QMAXENTRIES)

The tag number of a queue entry is the relative serial number of that entry from the time the queue was first used. When specified in conjunction with a STATUS queue, it can be used to determine whether that entry has been serviced or its relative position in the queue. When specified in conjunction with a CANCEL request it can be used to access the corresponding queue entry directly - i.e. position in queue file = tag number/size of queue. (A CANCEL command should of course carry out some checks - e.g. a comparison of user names - before altering the queue entry.)

The server descriptions are stored sequentially, each description consisting of 5 words :-

```
Word  0        Server status (-1 = do not use (SDONOTUSE)
                (SSTATUS)        0 = no server active with this
                                     number (SINACTIVE)
                                 1 = active (SACTIVE)
                                 2 = active and currently serving
                                     an entry (SACTIVESERVING)
                                 3 = suspended, awaiting entry
                                     to serve (SSUSPENDED)
                                 4 = server waiting but entry has
                                     been placed on queue (SSUSQREADY)
                                 5 = waiting - no queue(s) allocated
                                     to server  (SHASNOQUEUE))

  "     1        queue number of entry being served (SQNUMBER)
  "     2        position in queue of entry being served (SQPOSITION)
  "     3        number of entries served since start-up (SSERVICE)
  "     4        markers (bit 1 = 1 if termination has been
                  SMARKER                  requested (TERMINATE)
                  bits 13 - 15  = queue selection algorithm (SELECTALG)
                                1 = priority (PRIORITALG)
                                2 = round robin (ROUNDROBINALG))
```

The queue/server matrix is made up of one word entries; it has a row for each queue, a column for each server. A non-zero entry indicates the priority with which the server serves the corresponding queue (to be interpreted in conjunction with the server's queue selection algorithm).

A queuing system may be set up with any size of queue control block. One page would be sufficient to hold details of 14 queues with 6 servers.

All queue entries consist of 16 words:-

```
Words  0-7     file name (QFILENAME)
  "    8-11    user name (QUSERNAME)
  "    12      marker word (QMARKER)
  "    13      service indicators (QSERVICEIND)
               (bit 15 = 1 entry is present (Q ENTRY PRESENT))
               (bit 14 = 1 entry is being served) (QBEINGSERVED)
  "    14-15   spare
```

The marker word may be used to convey additional information to the server. Bit 15 of this word is always used to indicate 0/1 = do not delete/delete file after serving.
The significance, if any, of the remaining bits, will be specific to particular servers.

4. Software

The queueing system has been written in Culham Language for Systems Development (CLSD)[3] as the software is intended to be portable. (CLSD is a language designed to facilitate portability whilst generating efficient object code.)

The procedures have been written to allow the system to be implemented in any of the ways indicated in section 3.2

4.1 Coding Notes

No records (descriptors, queue entries) of the queue file overlap a page boundary (a page is the unit of disc transfer - on the Modular 1 this is 256 words) and all like records are contiguous on the backing store. These restrictions have two results a) only one disc buffer is required thus helping to keep the software small and b) only one basic routine is needed to read any record into core. Further, any word of the queue file can be accessed as it is merely a record of unit length.

This basic routine, DESCRIPTOR, gives the address of the required record in the disc buffer. The parameters are

(i) the record number (e.g. queue descriptor 5)

(ii) the record length (e.g. length = 9)

(iii) start address of the first record relative to the beginning of the queue file (e.g. first descriptor starts at word 9 of queue file)

All accesses to the queue file are made via this routine.

4.2 Disc Input/Output

The disc I/O is single buffered into the array given by the global constant "QFILEBUFF". Four routines handle the I/O.

(i) open q file

(ii) read q file (disc address)

(iii) write q file

(iv) release q

4.2.1 Open

This routine will normally be a dummy routine and only carry out a simple check to make sure the file is open. Normally, no alterations will be needed as it will be better to leave the queue file permanently "open" (i.e. open the queue file once at the beginning of each session).

4.2.2 Read/Write

A routine will be needed to read to/ write from a given disc page.

4.2.3 Release

This routine will not need alteration if the recommendation to leave the queue file permanently open is accepted.

Note that "write q file" is never called by the queue routines. To reduce disc accesses a routine "SET SAVEMARKER" is called to set a marker indicating that the contents of the disc buffer have been altered. This marker is examined by "read" and "release" and if set, the buffer is written to disc before carrying out the required operation.

4.3 Basic Queueing Procedures

The procedures listed below are given in the LSD format, where the parameters in the first pair of brackets are input parameters, and the second set of brackets (if any) contain the output parameters.

4.3.1 Procedures to Plant Entries in Queues (i.e. for Generators)

a) FUNCTION ADDTOQUE (Q,USER,FILE,MARKER)(STATUS,POSITION,TAG)

Function to add an entry to a queue.

Input Parameters

Q	- queue number to receive entry
USER	- address of 4 word array (2 characters per word) containing name of user initiating entry
FILE	- address of 8 word array (2 characters per word) containing name of file to be processed.
MARKER	- marker word (See section 3.3)

Output Parameters

STATUS - status of queue
POSITION - position of entry in queue (relative to start of queue)
TAG - tag number associated with this entry

Function Values

add to queue = Ø entry successfully added
-1 invalid queue number
-2 this queue not to be used
-3 queue full
-4 queue file fails to open

b) FUNCTION CHECKQUEUE (Q,GETLABEL,LABEL)

Function to check state of a queue

Input Parameters

Q - queue number
GETLABEL - 1 to generate a unique label (which can then be used to form a new file name)
- 0 do not generate a new label
LABEL - address of 4 word array (1 character per word) to receive new label if requested.

Function Values

checkqueue = Ø successful
-1 invalid queue number
-2 this queue not to be used
-3 queue full
-4 queue file fails to open

4.3.2 Procedures to Extract Entries from Queues (i.e. for Servers)

a) FUNCTION GET ENTRY (S,FILE,USER)(MARKER)

Function to get next entry to be served from a queue.

Input Parameters

S - server number
FILE - address of 8 word array (2 characters per word) to receive name of file to be served
USER - address of 4 word array (2 characters per word) to receive user name.

Output parameters

MARKER - marker word (see section 3.3)

Function Values

get entry = Ø entry successfully obtained
- 1 invalid server number
- 2 invalid server status
- 3 termination requested (entry to be ignored)
- 4 invalid selection algorithm
- 5 queues for this server empty
- 6 no queue(s) allocated to this server
- 7 queue file fails to open

b) FUNCTION REMOVE ENTRY (S)

Function to remove entry from queue served by server S

Input Parameters

S - server number

Function Values

remove entry = Ø entry removed from queue
-1 invalid server number
-2 invalid queue number being served
or queue has no server
-3 queue file fails to open

c) FUNCTION SERVE QUEUES(S)

Function to identify a server to the queueing system.

Input Parameters

S - server number

Function Values

serve queues = Ø identification successful
- 1 invalid server number
- 2 server already in use
- 3 server not to be used
- 4 queue file fails to open

5. Implementation Note

1. Set the machine dependent values as required in the CLSD text. These manifest constants are:-

DISC BLOCK SIZE	-	word size of disc block
PARITY MASK	-	mask to remove parity bit
CHARØ,CHAR9	-	character constants
CHARA,CHARZ	-	without parity bit
TERMINATE	-	termination bit (set to terminate a server in server descriptor marker word)

Note: it is assumed that characters Ø-9,A-Z have consecutive values for their internal representations (with parity bit removed). If this is not so then the routine "GETNEWLABEL" must be re-written.

2. Select appropriate values for the system constants (only required when creating a new queue file).

MAXQUEUES	-	Maximum number of queues in system
MAXSERVERS	-	Maximum number of servers in system
MAXQENTRIES	-	Maximum number of entries in any queue.

3. Write the disc I/O routines open, read, write, release as required (see section 4.2), adding any extra global variables as required.

4. Select the method to be used in implementing the system (see section 3.2) and write a main controlling routine if needed.

5. Process the CLSD routines to produce the desired code for the target machine.

6. Conclusions

The general purpose queueing mechanism will provide a convenient and flexible mechanism for the matching of tasks to resources, including provision for spooling, priority mechanisms, the pooling of resources and warm restart facilities.

7. Acknowledgements

The concepts underlying this work were developed during study of intelligent satellite systems sponsored by Computer Technology Limited, whose support is gratefully acknowledged.

REFERENCES

1. A General Purpose Task Queueing Mechanism for Small Machines by T LANG and D A FOX. Software - Practice and Experience Vol 4, pages 333-444.

2. LSD Manual by V J CALDERBANK and M CALDERBANK. CLM-PDN 9/71

3. A Portable Language for Systems Development by M CALDERBANK and V J CALDERBANK. Software - Practice and Experience Vol 3, pages 309-321.

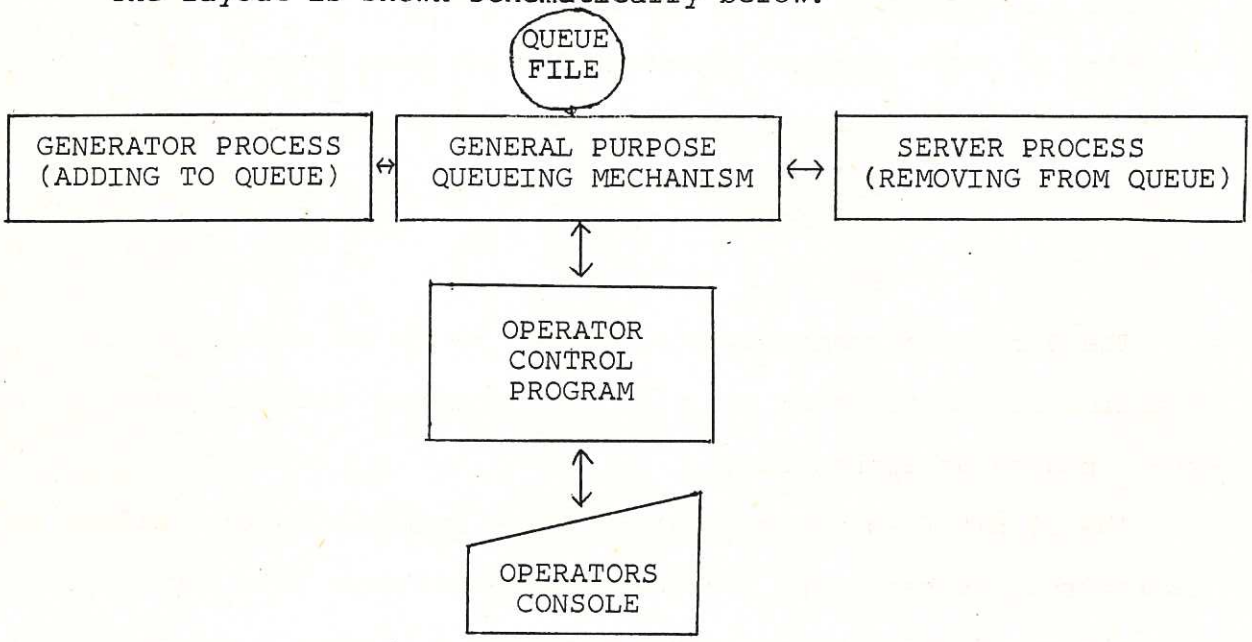
4. MISER - A minimum size executive for the Modular 1 by M CALDERBANK CLM-PDN 1/73

APPENDIX A
The Modular 1 Implementation

Modular 1 Implementation

The queueing mechanism has been implemented as a single core resident program running under MISER[4]. It is activated by other processes, by an exchange of messages, using the MISER message facility. The queue routines access the disc file directly.

The layout is shown schematically below.



The full specification of the MISER message facility is given in the MISER manual.

A1 General Message Format

a) Transmitting Data to Queue System

The A register is set up as follows

bits 2^0 - 2^1 segment containing data area (2 = Y, 3 = Z segment)

2^2 - 2^4 channel on which message is sent (must be 2)

2^5 - 2^{14} 10 bit flag field

2^{15} not used

The B register contains the address of a data

area where word Ø - function number defining queue operation

word 1 onwards - parameters of the corresponding function.

The M register contains the PRN of the core resident queue routines (currently 5).

b) Data returned from Queue System

The A register is set up as follows

bits 2^0 - 2^1 segment specification (Ø as no segment is passed)

2^2 - 2^4 channel on which message is sent (always 2)

2^5 - 2^{14} 10 bit flag field, exactly as transmitted by the caller.

2^{15} set to 1

The B register contains the integer result of the call. In addition the contents of the function parameter list may have been updated as appropriate.

The 10 bit flag field, sent in the A register by the caller, is returned unchanged by the queueing mechanism. This may be used to hold an identification number enabling a process to have more than one call to the queueing mechanism outstanding at the same time.

A2 Functions and Parameters

Function Ø - check queue

Parameters

1 queue number
2 marker = 1 if a new label is to be generated
3 address of 4 word array to receive label (1 character per word)

Message word returned

Ø success
-1 queue number invalid
-2 queue is not to be used (status = -1)
-3 queue full
-4 queue file fails to open

Function 1 - add to queue

Parameters

1 queue number
2 address of 4 word array containing name of user initiating entry (2 characters per word)
3 address of 8 word array containing name of file to be processed (2 characters per word)
4 marker word for queue entry
5 status of queue] set by the queue system
6 position of entry in queue] set by the queue system
7 tag number associated with this entry] set by the queue system

Message word returned

Ø success
-1 queue number invalid
-2 queue is not to be used (status =-1)
-3 queue full
-4 queue file fails to open

Function 2 - serve queues

Parameters

1 server number

Message word returned

Ø success
-1 server number invalid
-2 server already in use
-3 server not to be used (status = -1)
-4 queue file fails to open

Function 3 - get entry

Parameters

1 server number
2 address of 8 word array to receive file name (2 characters per word)
3 address of 4 word array to receive user name (2 characters per word)
4 marker word (set by queue system) - see add to queue

Message word returned

Ø success
-1 server number invalid
-2 invalid server status
-3 termination requested
-4 invalid selection algorithm
-5 queues for this server empty
-6 no queue(s) allocated to this server
-7 queue file fails to open

Function 4 - remove entry

Parameters

1 server number

Message word returned

Ø	success
-1	server number invalid
-2	invalid queue number/queue has no server
-3	queue file fails to open

Function 5 access queue file

This is a special function added to allow the operator control programs access to the queue file. The routine allows either any word of the queue file to be read or any bit of any word to be altered.

Parameters

1	read-write flag (0=read, 1=write)
2	word of queue file to be accessed + 1
3	1
4	zero
5	zero
6	address of area to receive the word (read/write flag=0) or word to be written to queue file (read/write flag=1)
7	mask to allow bit changing of a word (read/write flag=1) all bits zero to replace whole word.

Message word returned

Ø	success
-1	queue file fails to open

A3 System Functions

The operator control programs which carry out system "repairs" on the queue file e.g. reset the system after a crash need exclusive control of the queue file while active. This is achieved by having extra functions giving access to the disc input/output routines. A program can open the queue file, and keep it "locked" during updates so that normal functions such as adding to a queue are prevented. When the update is complete, the queue file is released. These extra functions have values starting at 100.

Function 100 - open q file

Parameters - none

Message word returned

Ø success
-1 file failed to open

Function 101 - read q file

Parameters

1 - disc address (page number)

Message word returned

Ø - read successful
-1 - read failed

Function 102 - set save marker

Parameters - none

Message word returned - not used

Function 103 - release q

Parameters - none

Message word returned - not used

Function 104 - get disc buffer

Parameters - none

Message word returned

address of disc buffer used by the core-resident queue routines.

A4 GENERAL PURPOSE QUEUING SYSTEM - WITH DEBUG CODE

Function

To service queue file

Core Requirements

x:y:z 10:4:Ø

Running

This program must be loaded as PRN 5.

The program uses its own disc I/O. There are two alterations required before running.

a) The disc page address of the start of the queue file (currently 3000). This value is in location 1557 of the X segment.

b) A check is made for illegal disc addresses. The upper limit of the queue file is stored in location 1496 of the X segment in the form 'SBB L n', where n is the upper limit. n should be changed to correspond to the page size of the queue file as given by the initialisation program. (currently n=9).

Termination

This program will only terminate if an error is detected. The most likely failure points are given below:

Program Counter		Reason for Failure
Decimal	Hex	
98	62	invalid function value
235	EB	" " "
1491	5D3	negative disc address) queue file
1500	5DC	disc address too high) corrupt

Note

This program contains debug code. It also prints the queue number being served and the selection algorithm used to select the entry on the monitor console.

A5 GENERAL PURPOSE QUEUING SYSTEM - PRODUCTION VERSION ("SMALL")

Function

To service queue file

Core Requirements

x:y:z 6:2:Ø

Running

This program must be loaded as PRN 5.

The program uses its own disc I/O. There are two alterations required before running.

a) The disc page address of the start of the queue file (currently 3000). This value is in location 1511 of the X segment.

b) A check is made for illegal disc addresses. The upper limit of the queue file is stored in location 1468 of the X segment in the form 'SBB L n', where n is the upper limit. n should be changed to correspond to the page size of the queue file as given by the initialisation program. (Currently n=9).

Termination

This program will only terminate if an error is detected. The most likely failure points are given below:

Program Counter		
Decimal	Hex	Reason for Failure
61	3D	invalid function value
189	BD	" " "
1466	5BA	negative disc address) queue file
1471	5BF	disc address too high) corrupt

A6 INITIALISATION OF QUEUE FILE

Function

To set up a queue file on disc

Core Requirements

x:y:z 6:6:Ø

Running

The program uses its own disc I/O. Word 806 of the X segment contains the disc page address of the start of the queue file (currently set to 3000). Change this word to suit your requirements.

Prompts and Responses

See example.

EXAMPLE OF QUEUE FILE INITIALISATION

```
TINT: RN
PRN: 6;
PTY: 2;
MAXIMUM NUMBER OF QUEUES FOR THIS SYSTEM = 14
MAXIMUM NUMBER OF SERVERS = 6
MAXIMUM ENTRIES IN ANY QUEUE = 64
NUMBER OF QUEUES?3;
NUMBER OF SERVERS?2;
ALPHA NUMERIC 4 CHARACTER LABEL?1234
QUEUE FILE HEADER PARAMETERS

3 QUEUES WITH DESCRIPTORS STARTING AT 9
2 SERVERS WITH DESCRIPTORS STARTING AT 36
QUEUE SERVER MATRIX STARTS AT 46
FILE LABEL IS 1234
OK?Y
INITIALISATION OF QUEUE DESCRIPTORS

QUEUE 1
NUMBER OF ENTRIES?4;
START 1,END 2,ROOM FOR 16 ENTRIES
OK?Y
QUEUE 2
NUMBER OF ENTRIES?16;
START 2,END 3,ROOM FOR 16 ENTRIES
OK?Y
QUEUE 3
NUMBER OF ENTRIES?17
START 3,END 5,ROOM FOR 32 ENTRIES
OK?Y
INITIALISATION OF SERVER DESCRIPTORS

REPLY 1 FOR PRIORITY,2 FOR ROUND ROBIN SELECTION ALGORITHM
SERVER 1
ALGORITHM?1;
OK?Y
SERVER 2
ALGORITHM?2;
OK?Y
SET UP Q-S MATRIX

REPLY 0 SERVER NUMBER WHEN FINISHED
SERVER NUMBER?1;
GIVE Q/PRIORITY
REPLY -1 TO MOVE TO NEXT SERVER1/3,2/2,-1;
SERVER NUMBER?2;
GIVE Q/PRIORITY
REPLY -1 TO MOVE TO NEXT SERVER3/1,-1;
SERVER NUMBER?;
Q SERVERS
   1  2
1  3  0
2  2  0
3  0  1
OK?YWRFTE Q FILE ENTERED BUT SAVE MARKER NOT SET
WRITE Q FILE ENTERED BUT SAVE MARKER NOT SET
WRITE Q FILE ENTERED BUT SAVE MARKER NOT SET
WRITE Q FILE ENTERED BUT SAVE MARKER NOT SET

DISC BLOCK SIZE OF Q FILE = 5
OK?Y
```

Note: No. entries is always rounded up to a multiple of 16.

debug code

A7.1 OPERATOR UTILITY PROGRAM 1

Function

To access queue and server descriptors

Core Requirements

x:y:z 5:3:Ø

Running

The core resident queue routines must be loaded as PRN 5 and must be running before this program is activated.

Commands

The command is typed in response to the query

?

The reply can consist of up to 6 characters terminated by a semi-colon. Command names may be truncated. The commands available are

QSTATUS	Print queue status and number of entries
SSTATUS	Print server status and selection algorithm
QALTER	Alter queue status (to "on" or "off")
STERM	Set termination marker of a server
AALTER	Alter selection algorithm of a server
HELP	Lists the commands available to the console
HALT	Terminate program
SALTER	Alter server status (to "on" or "off")

Prompts and responses

All numbers should be terminated either by a "/" or ";"

The program is sent back into command state if

a) ; is the first character of the reply

b) if a complete range of values has been requested e.g. if the status of all queues has been requested.

Command	Prompt	Response
QSTATUS SSTATUS QALTER	Q/STATUS?	queue number/new status (ON or OF)
STERM	SERVER?	server number
AALTER	S/ALG ?	server/new algorithm PR=Priority or RR=Round robin)
SALTER	S/STATUS?	server number/new status (ON or OF)

A7.2 OPERATOR UTILITY PROGRAM 2

Function

Queue-Server Matrix control program

Core Requirements

x:y:z 6:2:Ø

Running

The core resident queue routines must be loaded as PRN 5 and running before this program is activated.

Commands

The command is typed in response to the query

?

The reply can consist of up to 6 characters terminated by a semi-colon. Command names may be truncated. The commands available are

QLIST	Print entries in a queue
MALTER	alter elements of the queue-server matrix
MLIST	print the queue-server matrix
HELP	lists the commands available to the console
HALT	terminate program

Prompts and responses

All numbers should be terminated either by a "/" or ";
The programe is sent back into command state if

a) ; is the first character of the reply

b) if a complete range of values has been requested, e.g. if the contents of all queues have been listed.

Command	Prompt	Response
QLIST	Q1/Q2?	range of queues e.g. 1/3
MALTER	a)Q/S?	queue number/server number
	b)element?	new value of matrix element
MLIST		

A7.3 OPERATOR UTILITY PROGRAM 3

Function

To reset queue file after a system "crash".

Core Requirements

x:y:z 11:3:Ø

Running

The core resident routines must be loaded as PRN 5 and must be running before this program is activated. The system should be quiescent. Any other program accessing the queue file when this program is active is likely to fail "queue file already open".

Commands

The command is typed in response to the query

?

The reply can consist of up to 6 characters terminated by a semi-colon. Command names may be truncated. The commands available are:

QCB	Print queue control block (all information except queue entries)
DALTER	Alter any word of queue or server descriptors.
CLEARQ	Clear entries from a range of queues.
RESETQ	Reset queue file (for use after system crash).
HELP	List available commands.
HALT	Terminate program.

Prompts and Responses

All numbers should be terminated either by a "/" or ";"
The program is sent back into command state if

a) ; is the first character of reply.

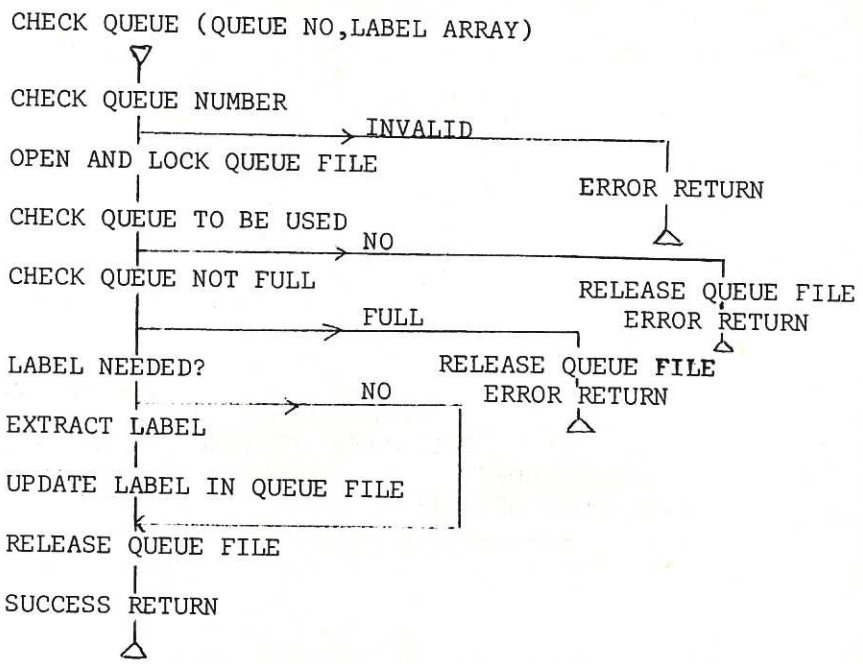
b) if a complete range values has been requested (e.g. all queues cleared).

Command	Prompt	Response
QCB		
DALTER	TN/A/V?	T(type)=S(server) or D(descriptor) N(number)=descriptor number A(address)=word of descriptor to be changed V(value)=new value e.g. S1/Ø/-1; sets word Ø of server descriptor 1 to -1.
CLEARQ	Q1/Q2?	range of queues e.g. 1/3
RESETQ		

APPENDIX B

General Flow Charts for the General Purpose Queueing System

integer procedure

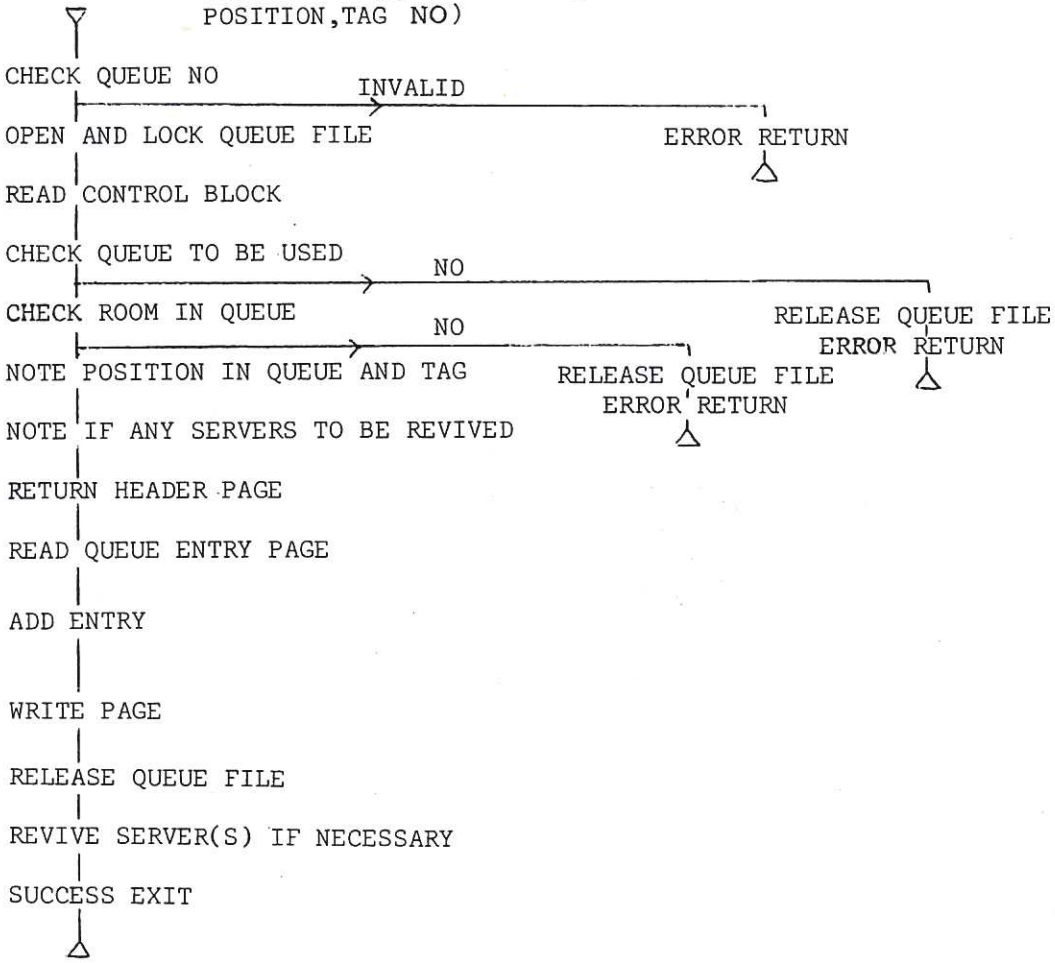


QUEUING PROCEDURE –

CHECK QUEUE

GENERAL PURPOSE QUEUING MECHANISM

integer procedure ADD TO QUEUE (QUEUE NO,FILE NAME,USER NAME,
MARKER WORK,QUEUE STATUS,
POSITION,TAG NO)

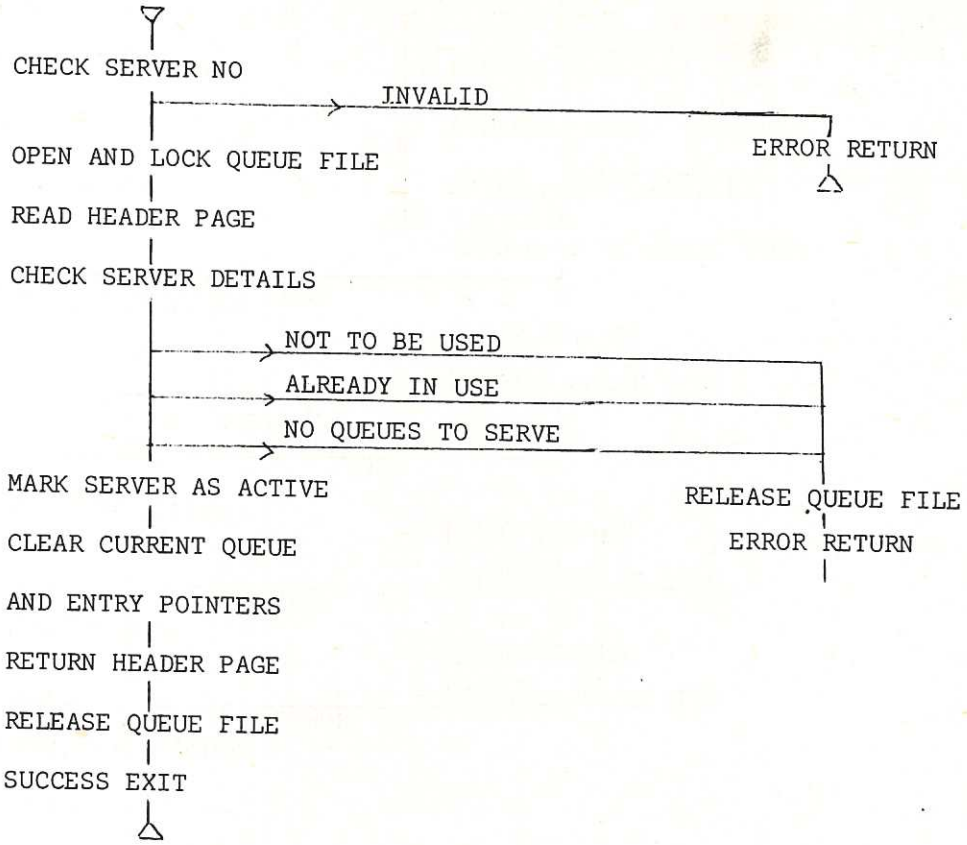


QUEUING PROCEDURE

ADD TO QUEUE

GENERAL PURPOSE QUEUING MECHANISM

integer procedure SERVE QUEUES (SERVER NO)



QUEUING PROCEDURE

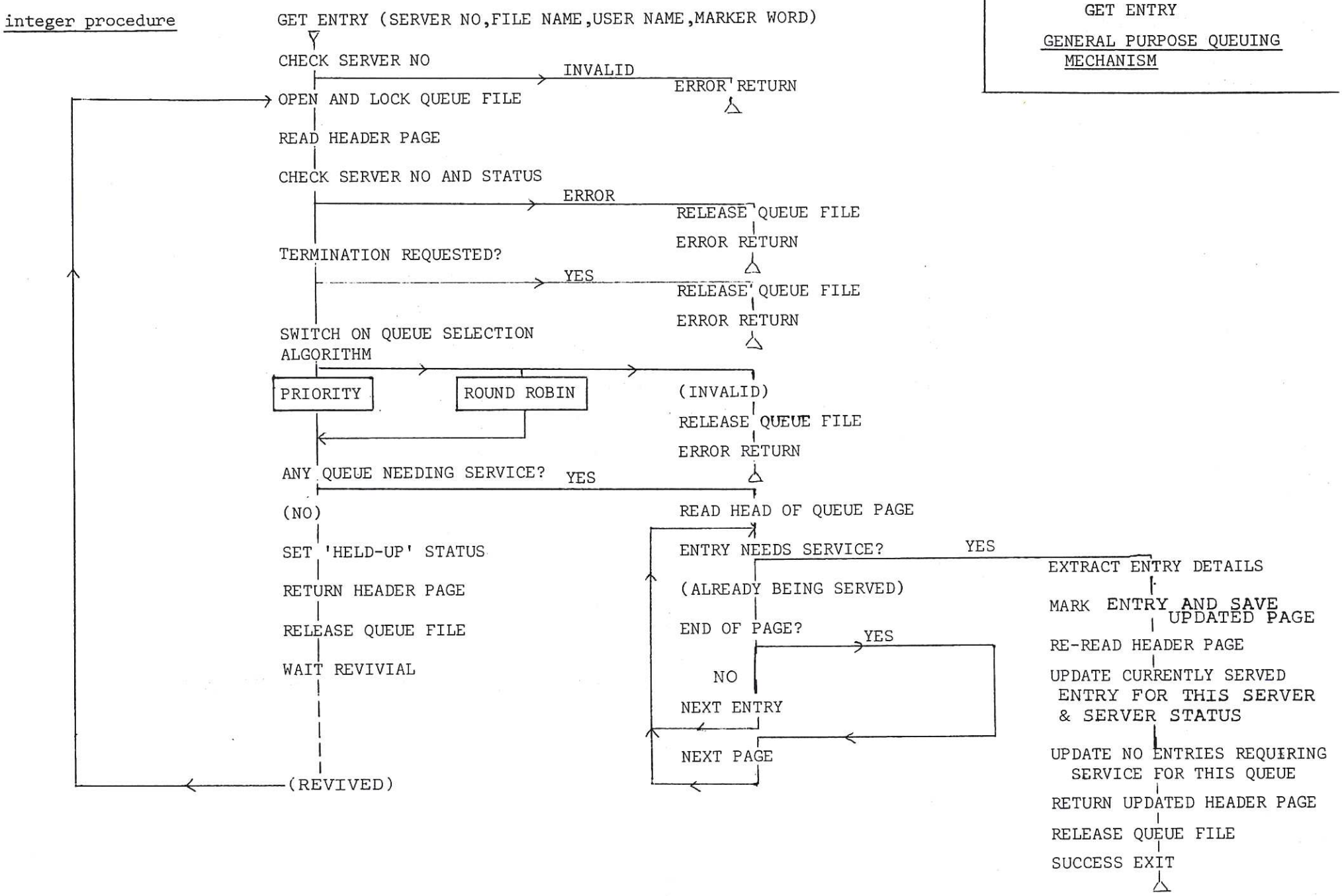
SERVE QUEUES

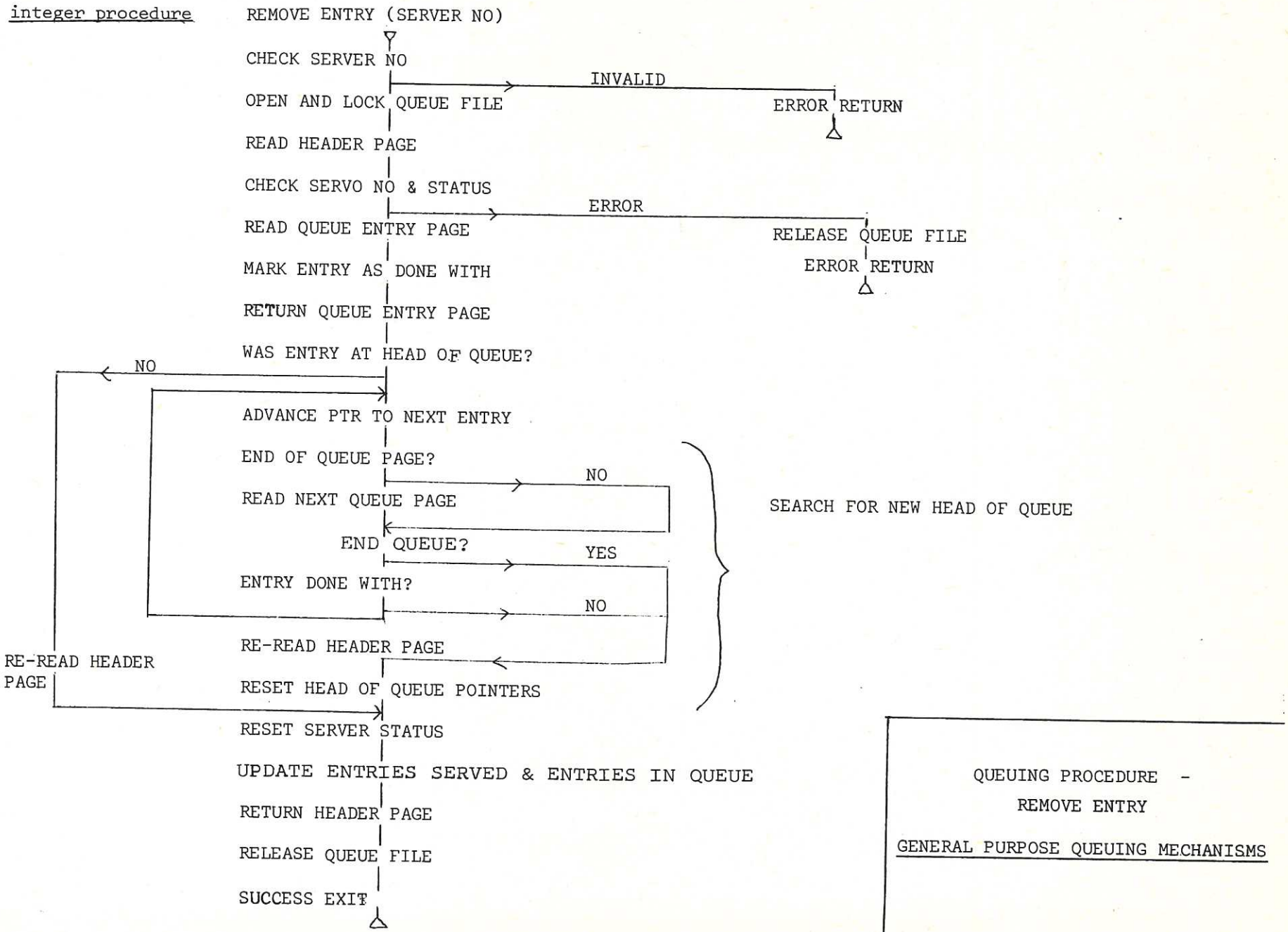
GENERAL PURPOSE QUEUING MECHANISM

QUEUING PROCEDURE -

GET ENTRY

GENERAL PURPOSE QUEUING MECHANISM





QUEUING PROCEDURE -
REMOVE ENTRY

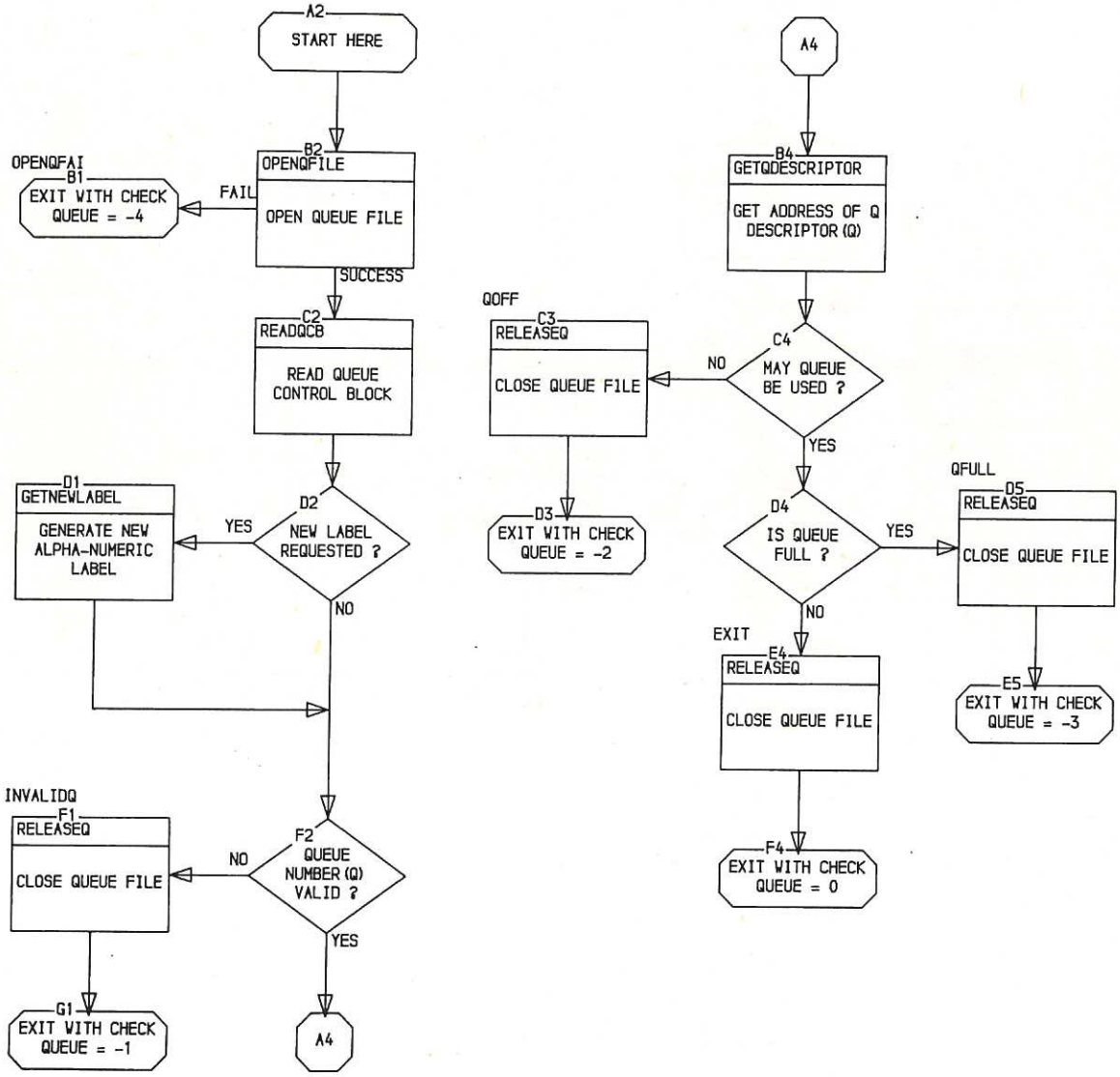
GENERAL PURPOSE QUEUING MECHANISMS

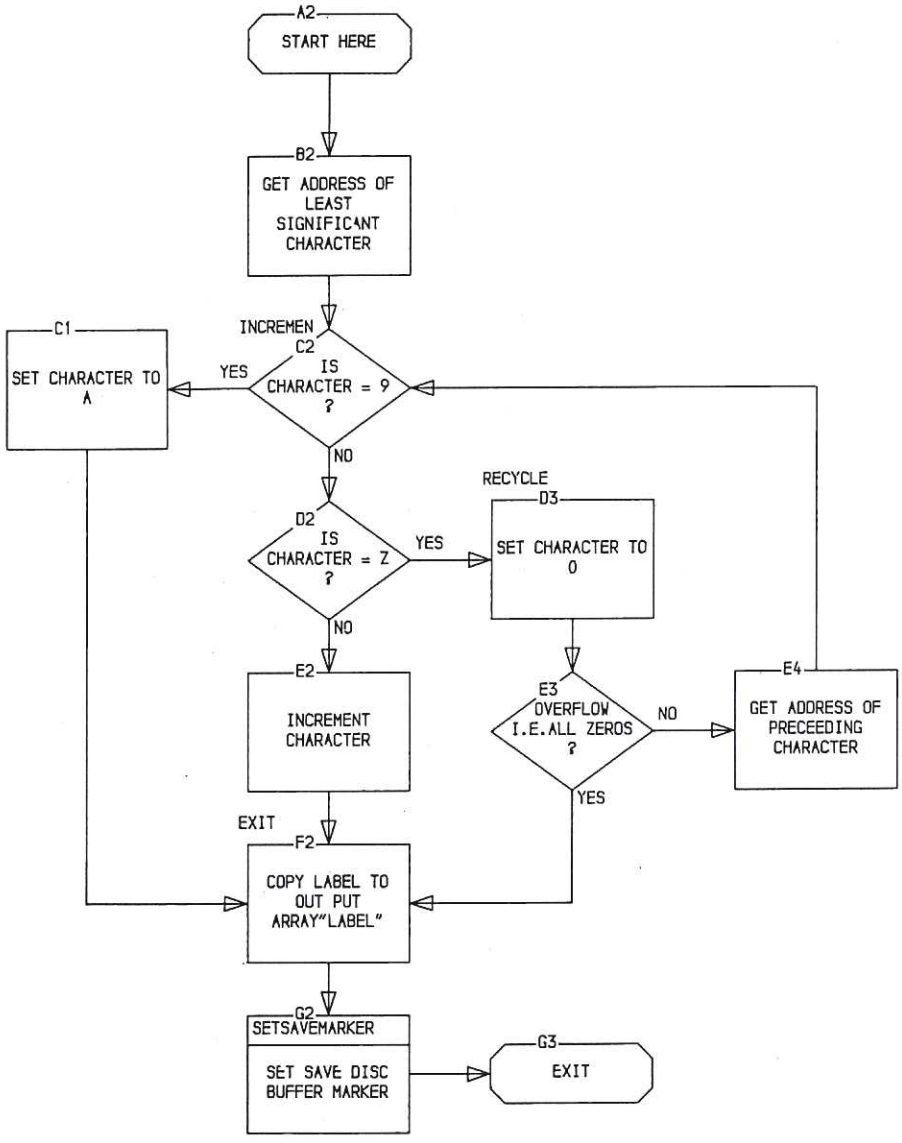
APPENDIX C

Detailed Flow Charts of the General Purpose Queueing System

GENERAL PURPOSE QUEUEING MECHANISM. D A FOX, T LANG, CULHAM LABORATORY, ABINGDON
, FUNCTION CHECK QUEUE (Q, GET LABEL, LABEL) -MODULE LSD.CHECKQ (S)

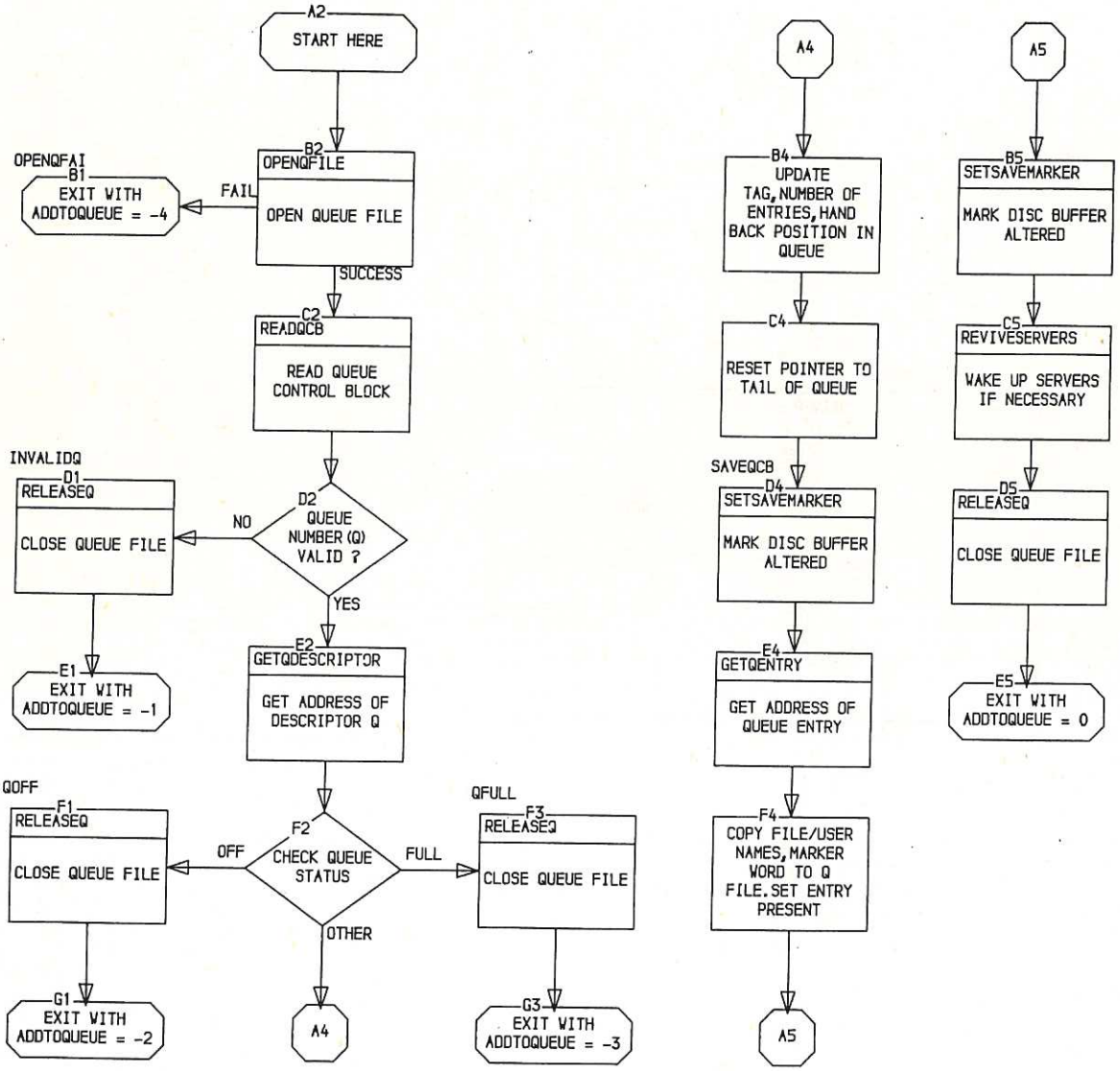
CHART P1

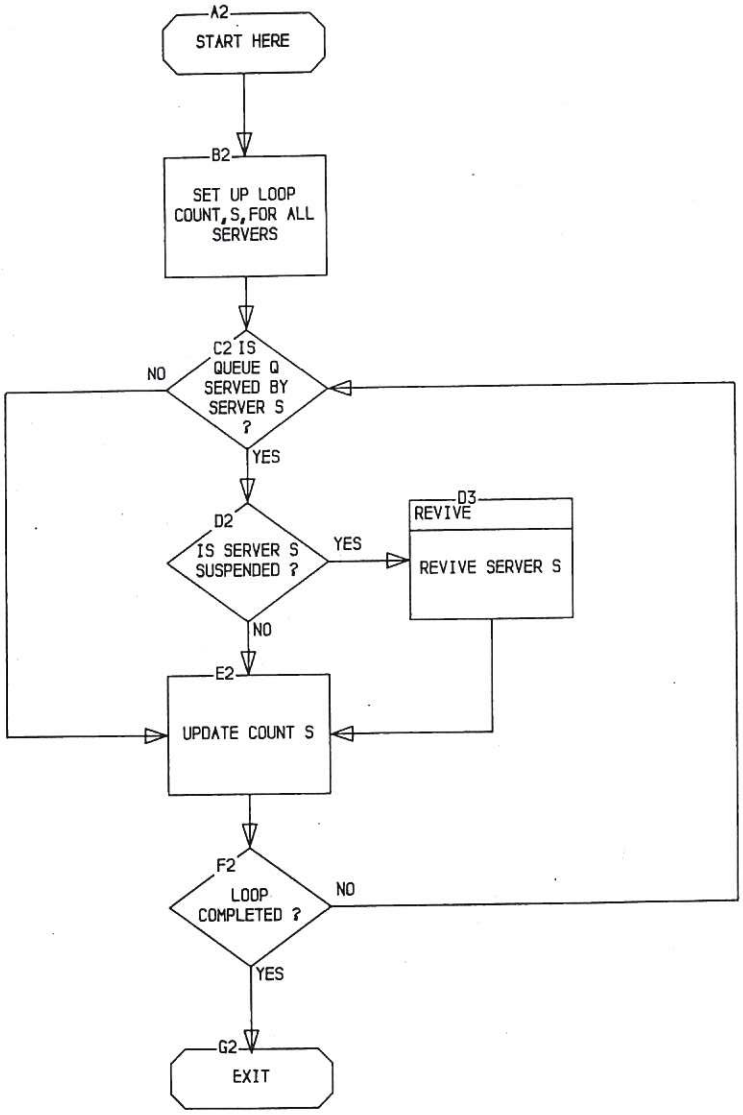


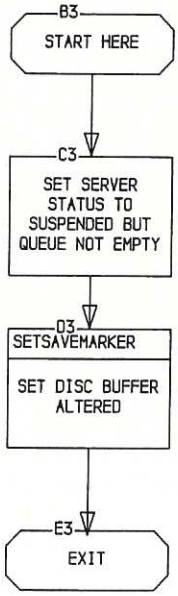

A2
START HERE
B2
GET ADDRESS OF LEAST SIGNIFICANT CHARACTER
INCREMEN
C2
IS CHARACTER = 9 ?
YES
C1
SET CHARACTER TO A
NO
RECYCLE
D3
SET CHARACTER TO 0
D2
IS CHARACTER = Z ?
YES
NO
E2
INCREMENT CHARACTER
E3
OVERFLOW I.E.ALL ZEROS ?
NO
E4
GET ADDRESS OF PRECEEDING CHARACTER
YES
EXIT
F2
COPY LABEL TO OUT PUT ARRAY"LABEL"
G2
SETSAVEMARKER
SET SAVE DISC BUFFER MARKER
G3
EXIT

GENERAL PURPOSE QUEUEING MECHANISM D A FOX, T LANG, CULHAM LABORATORY, ABINGDON
FUNCTION ADD TO QUEUE (Q, USER, FILE, MARKER) (STATUS, POSITION, TAG) -MODULE LSD.ADDTOQ

CHART P1

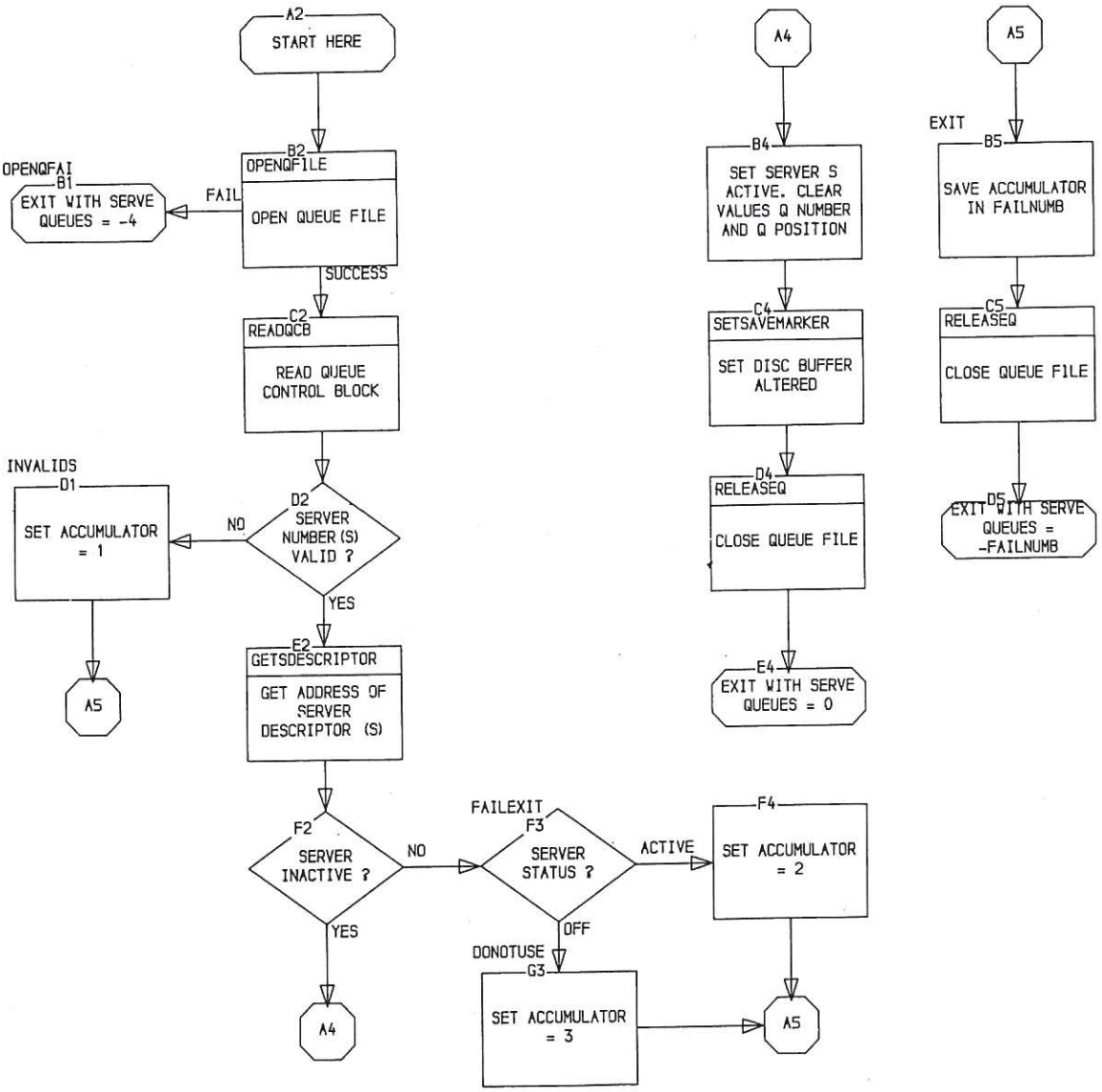



A2
START HERE
B2
SET UP LOOP COUNT, S, FOR ALL SERVERS
C2 IS QUEUE Q SERVED BY SERVER S ?
NO
YES
D2
IS SERVER S SUSPENDED ?
YES
D3
REVIVE
REVIVE SERVER S
NO
E2
UPDATE COUNT S
F2
LOOP COMPLETED ?
NO
YES
G2
EXIT


B3
START HERE
C3
SET SERVER STATUS TO SUSPENDED BUT QUEUE NOT EMPTY
D3
SETSAVEMARKER
SET DISC BUFFER ALTERED
E3
EXIT

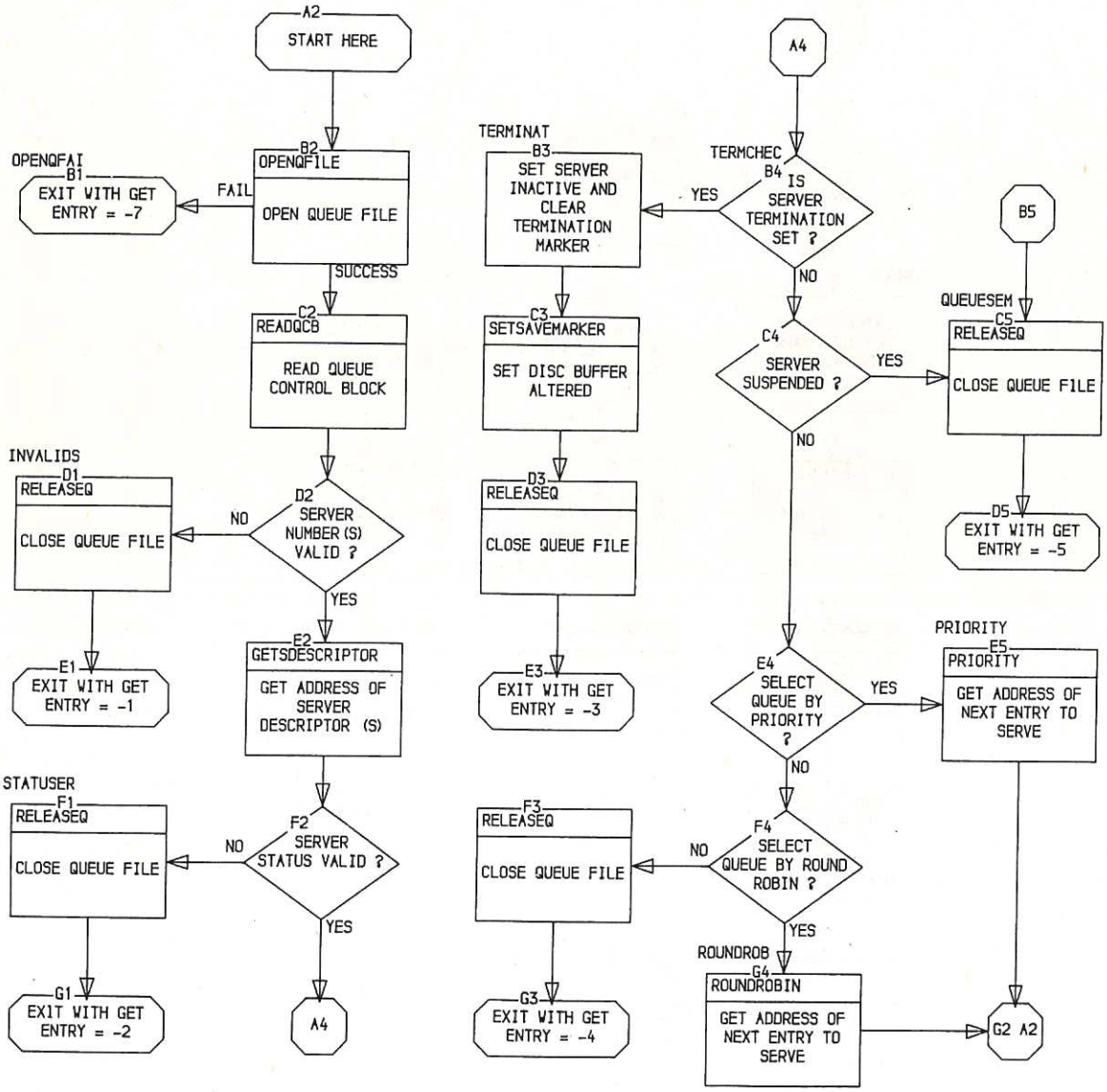
GENERAL PURPOSE QUEUEING MECHANISM. D A FOX, T LANG, CULHAM LABORATORY, ABINGDON
FUNCTION SERVE QUEUES (S) -MODULE LSD.SERVEQ (S)

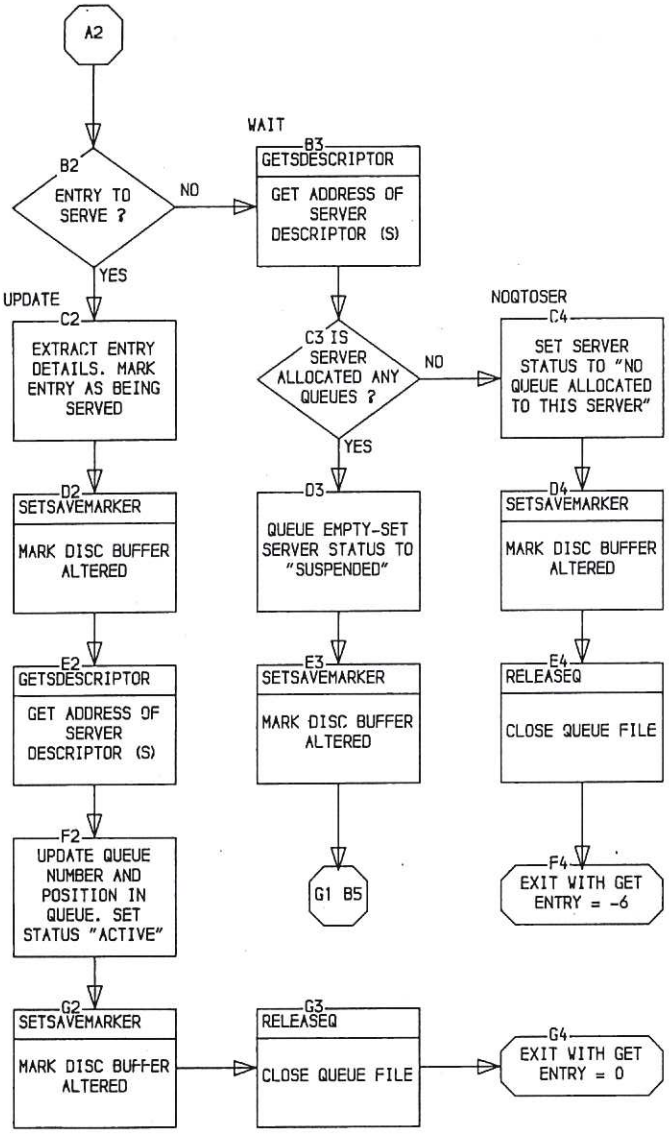
CHART S1



GENERAL PURPOSE QUEUEING MECHANISM. D A FOX, T LANG, CULHAM LABORATORY, ABINGDON
FUNCTION GET ENTRY (S, FILE, USER) (MARKER) -MODULE LSD.GENTRY (S)

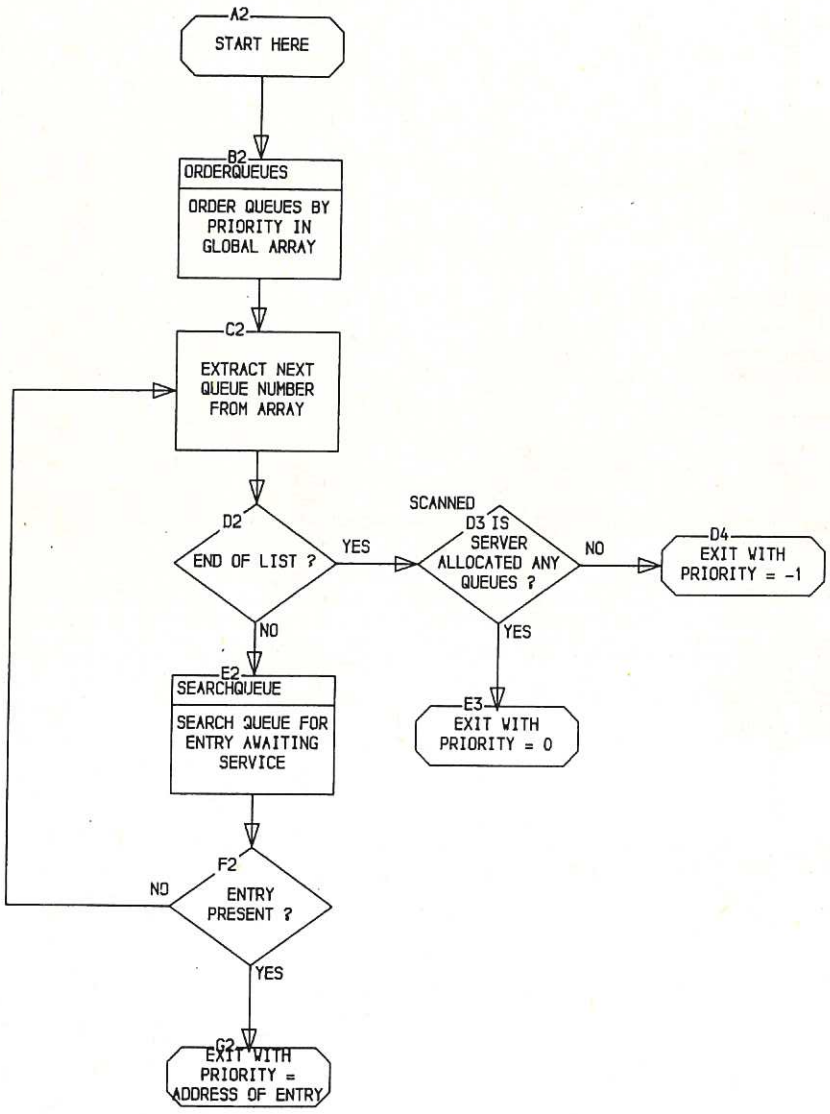
CHART G1




A2
B2
ENTRY TO SERVE ?
NO
YES
WAIT
B3
GETSDESCRIPTOR
GET ADDRESS OF SERVER DESCRIPTOR (S)
UPDATE
C2
EXTRACT ENTRY DETAILS. MARK ENTRY AS BEING SERVED
C3 IS SERVER ALLOCATED ANY QUEUES ?
NO
YES
NOQTOSER
C4
SET SERVER STATUS TO "NO QUEUE ALLOCATED TO THIS SERVER"
D2
SETSAVEMARKER
MARK DISC BUFFER ALTERED
D3
QUEUE EMPTY-SET SERVER STATUS TO "SUSPENDED"
D4
SETSAVEMARKER
MARK DISC BUFFER ALTERED
E2
GETSDESCRIPTOR
GET ADDRESS OF SERVER DESCRIPTOR (S)
E3
SETSAVEMARKER
MARK DISC BUFFER ALTERED
E4
RELEASEQ
CLOSE QUEUE FILE
F2
UPDATE QUEUE NUMBER AND POSITION IN QUEUE. SET STATUS "ACTIVE"
G1 B5
F4
EXIT WITH GET ENTRY = -6
G2
SETSAVEMARKER
MARK DISC BUFFER ALTERED
G3
RELEASEQ
CLOSE QUEUE FILE
G4
EXIT WITH GET ENTRY = 0

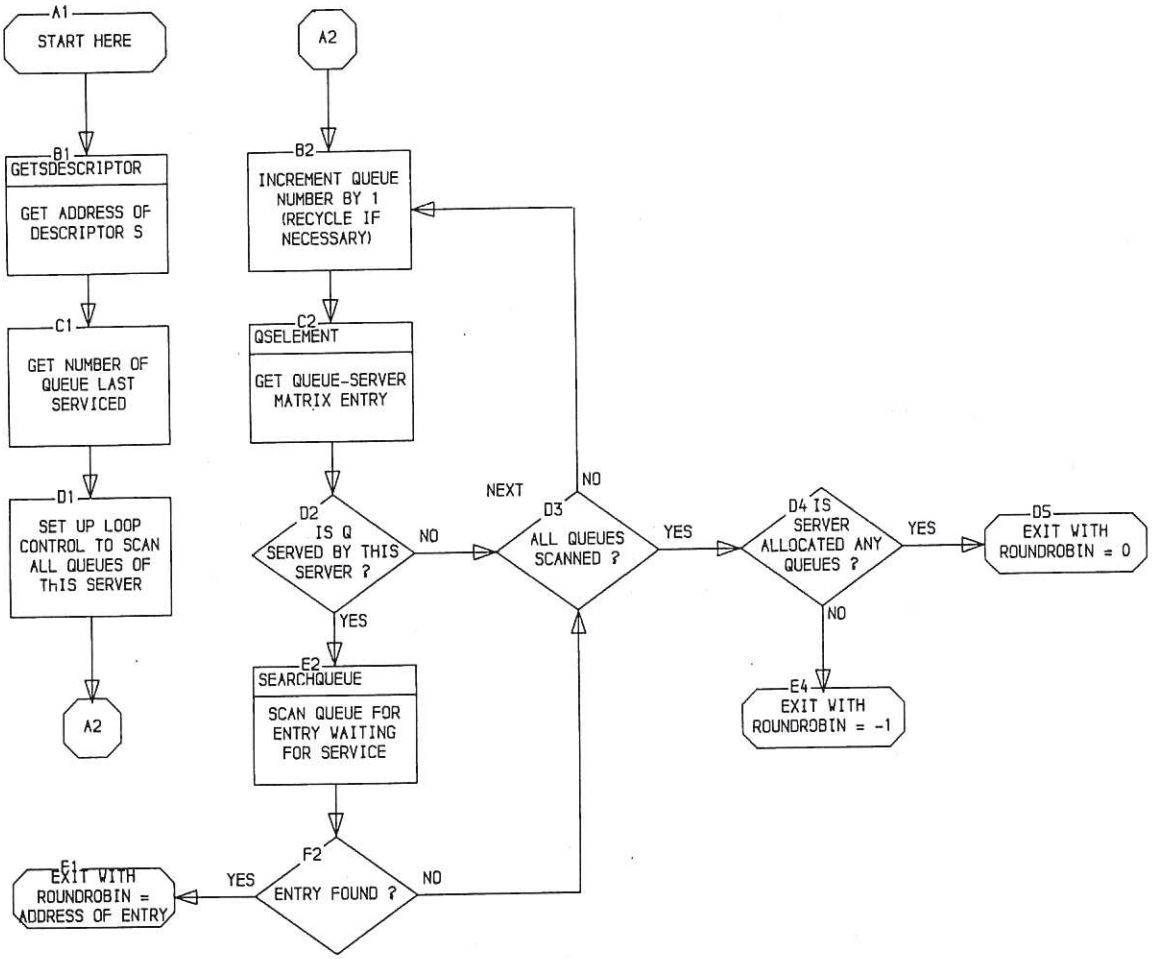
GENERAL PURPOSE QUEUEING MECHANISM. D A FOX, T LANG, CULHAM LABORATORY, ABINGDON
PRIORITY (S) (Q,P) -MODULE LSD.GENTRY (S)

CHART G3



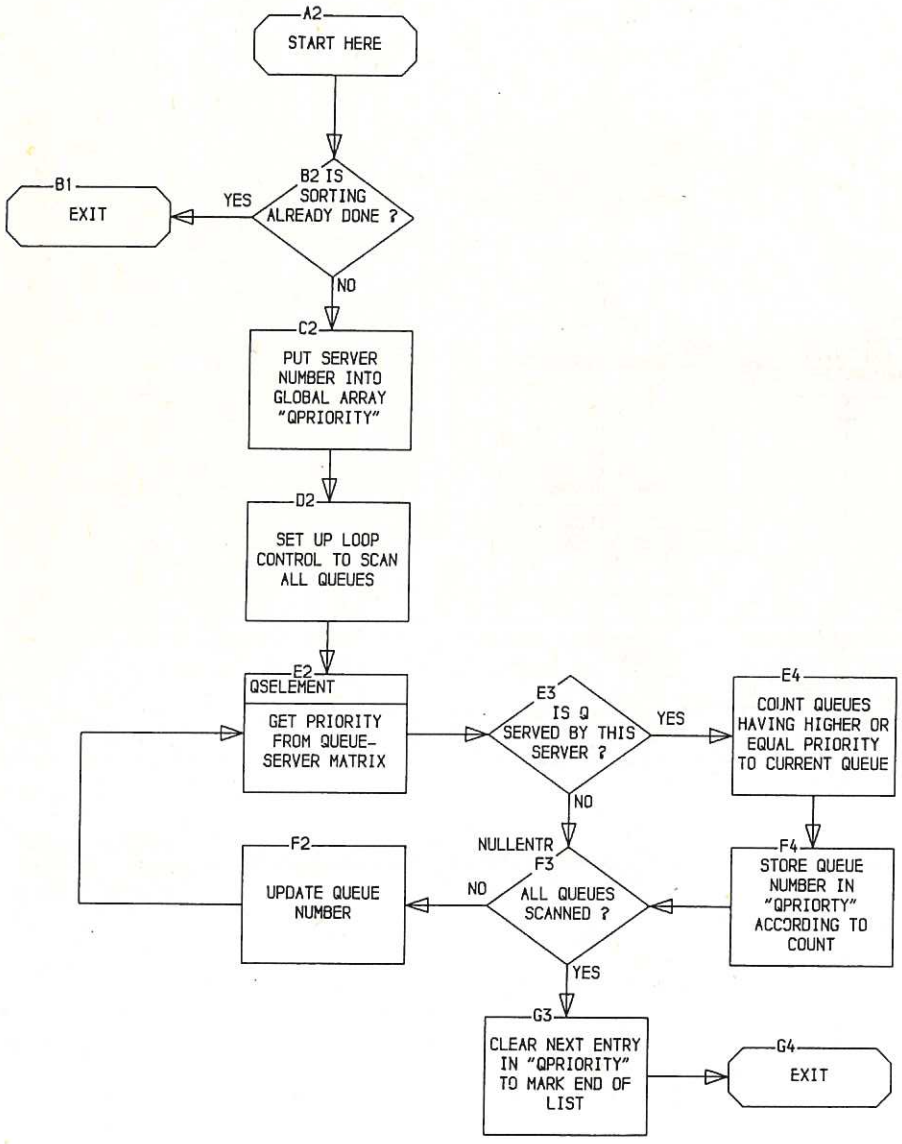
GENERAL PURPOSE QUEUEING MECHANISM. D A FOX, T LANG, CULHAM LABORATORY, ABINGDON
ROUND ROBIN (S) (Q, P) -MODULE LSD.GENTRY (S)

CHART G4



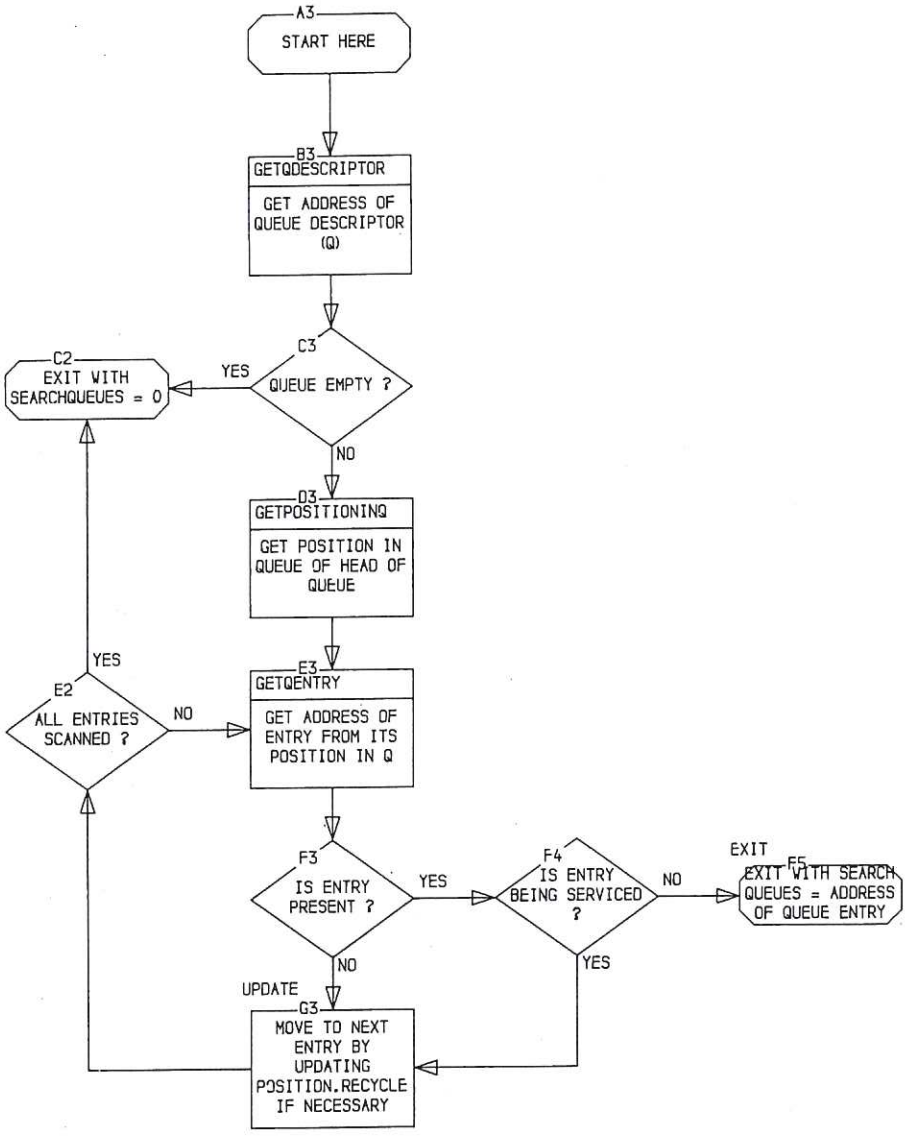
GENERAL PURPOSE QUEUEING MECHANISM. D A FOX, T LANG, CULHAM LABORATORY, ABINGDON
ORDER QUEUES (S) -MODULE LSD.GENTRY (S)

CHART G5



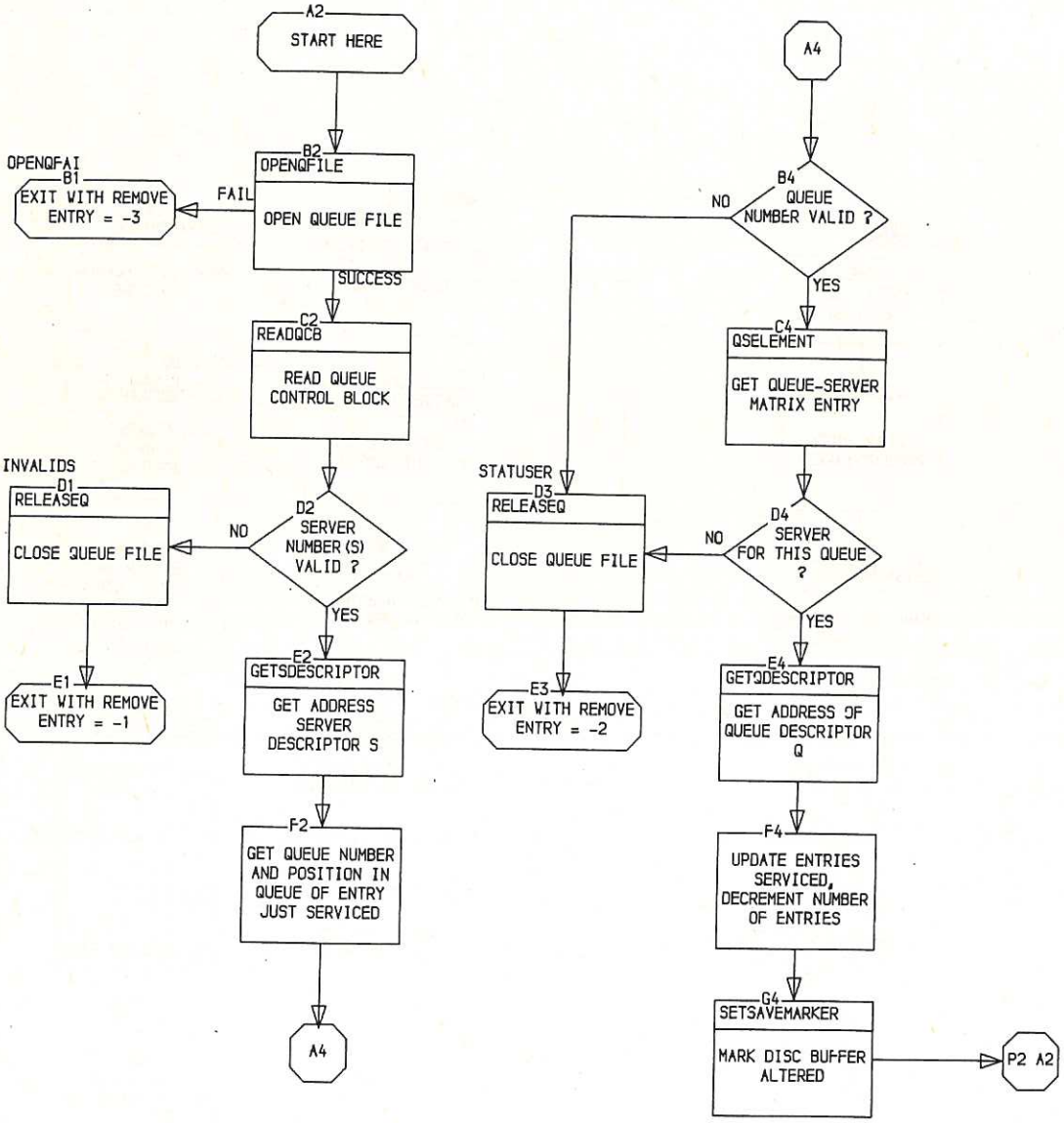
GENERAL PURPOSE QUEUEING MECHANISM. D A FOX, T LANG, CULHAM LABORATORY, ABINGDON
SEARCH QUEUE (Q) (POSITION IN Q) -MODULE LSD.GENTRY

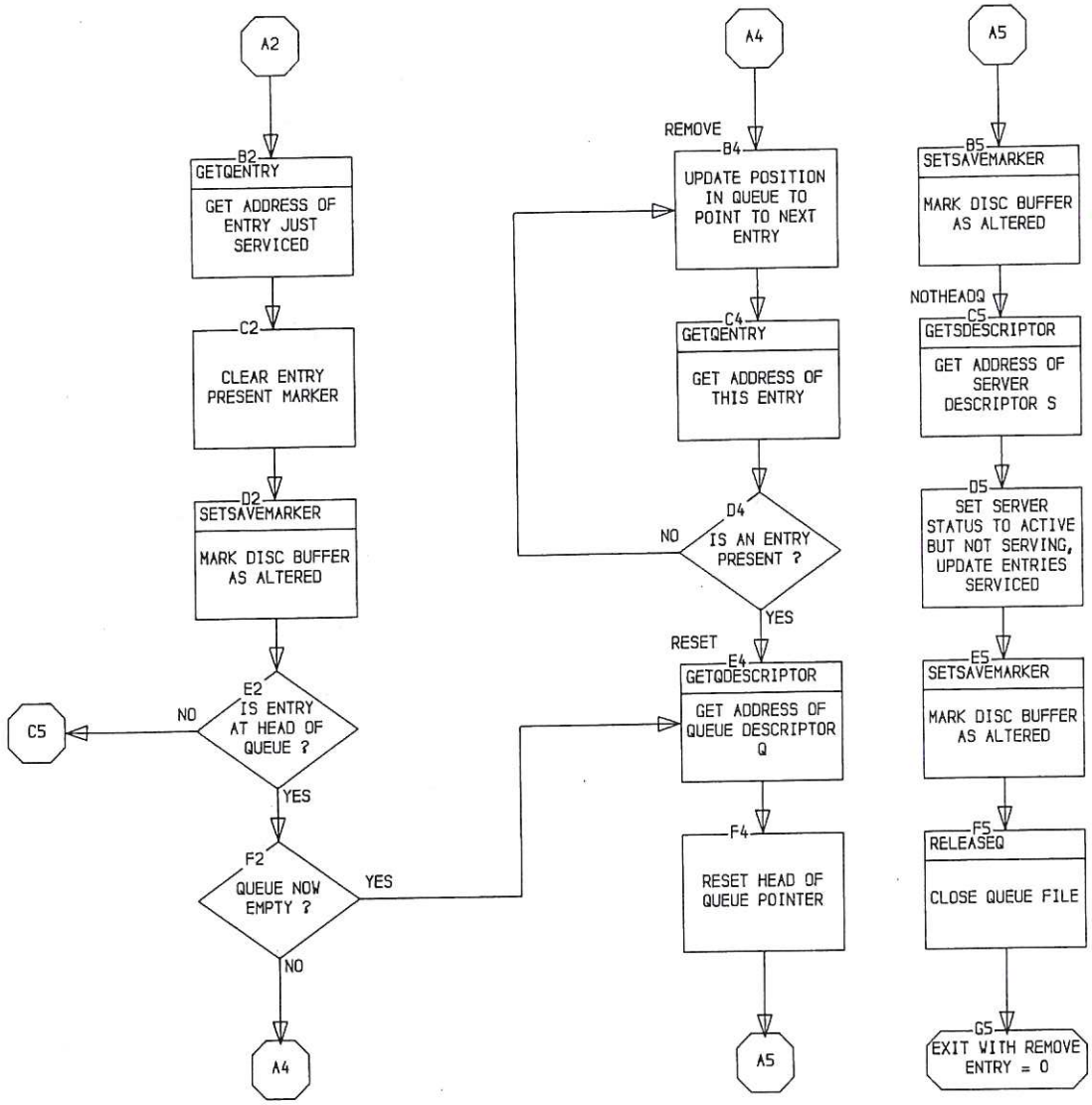
CHART G6



GENERAL PURPOSE QUEUEING MECHANISM D A FOX, T LANG, CULHAM LABORATORY, ABINGDON
FUNCTION REMOVE ENTRY (S) -MODULE LSD.REMOVE (S)

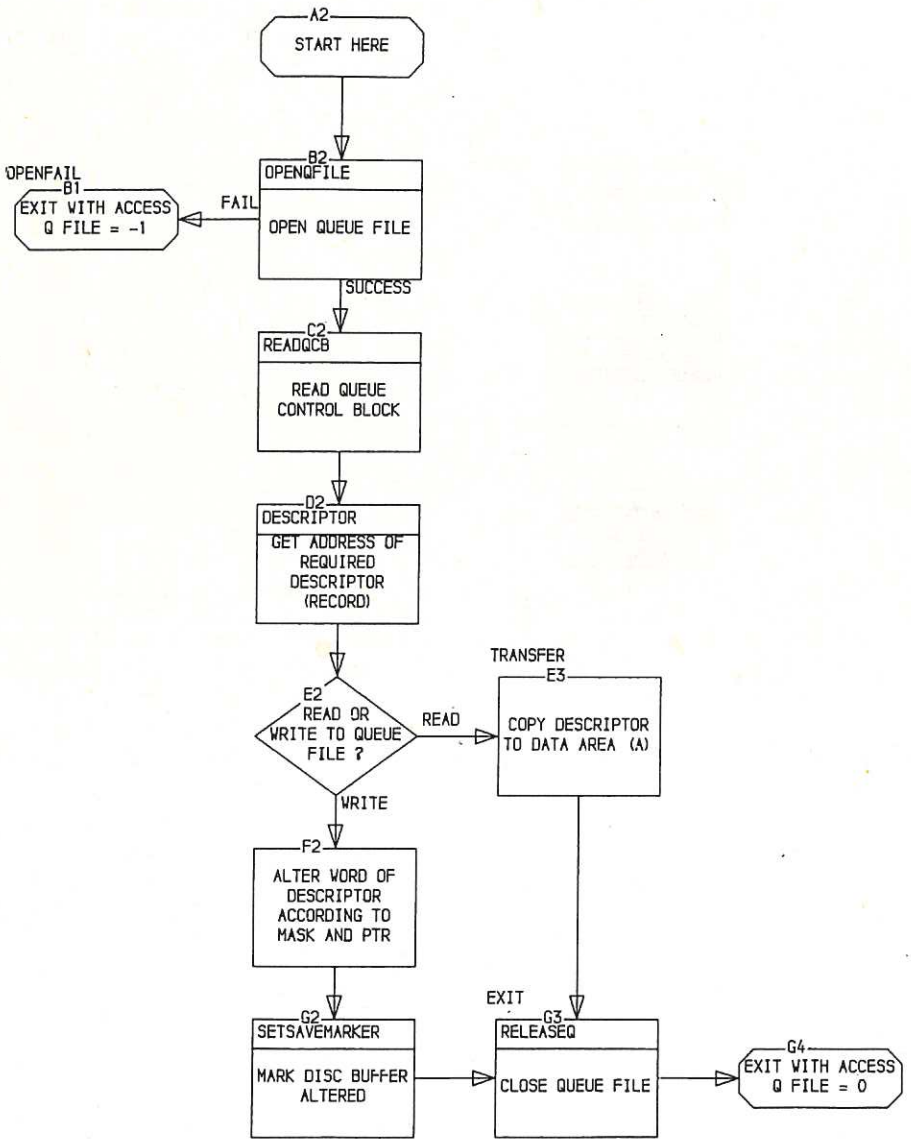
CHART P1

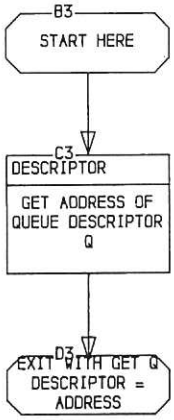


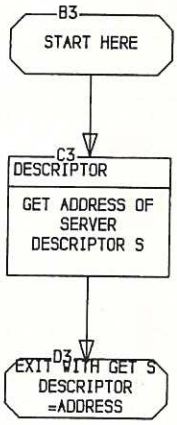

A2
B2
GETQENTRY
GET ADDRESS OF ENTRY JUST SERVICED
C2
CLEAR ENTRY PRESENT MARKER
D2
SETSAVEMARKER
MARK DISC BUFFER AS ALTERED
E2
IS ENTRY AT HEAD OF QUEUE ?
NO
C5
YES
F2
QUEUE NOW EMPTY ?
YES
NO
A4
A4
REMOVE
B4
UPDATE POSITION IN QUEUE TO POINT TO NEXT ENTRY
C4
GETQENTRY
GET ADDRESS OF THIS ENTRY
D4
IS AN ENTRY PRESENT ?
NO
YES
RESET
E4
GETQDESCRIPTOR
GET ADDRESS OF QUEUE DESCRIPTOR Q
F4
RESET HEAD OF QUEUE POINTER
A5
A5
B5
SETSAVEMARKER
MARK DISC BUFFER AS ALTERED
NOTHEADQ
C5
GETSDESCRIPTOR
GET ADDRESS OF SERVER DESCRIPTOR S
D5
SET SERVER STATUS TO ACTIVE BUT NOT SERVING, UPDATE ENTRIES SERVICED
E5
SETSAVEMARKER
MARK DISC BUFFER AS ALTERED
F5
RELEASEQ
CLOSE QUEUE FILE
G5
EXIT WITH REMOVE ENTRY = 0

GENERAL PURPOSE QUEUEING MECHANISM D A FOX, T LANG, CULHAM LABORATORY, ABINGDON
FUNCTION ACCESS Q FILE (RWFLG, N, LENGTH, START, PTR, A, MASK) -MODULE LSD. ACCESS (S)

CHART P1

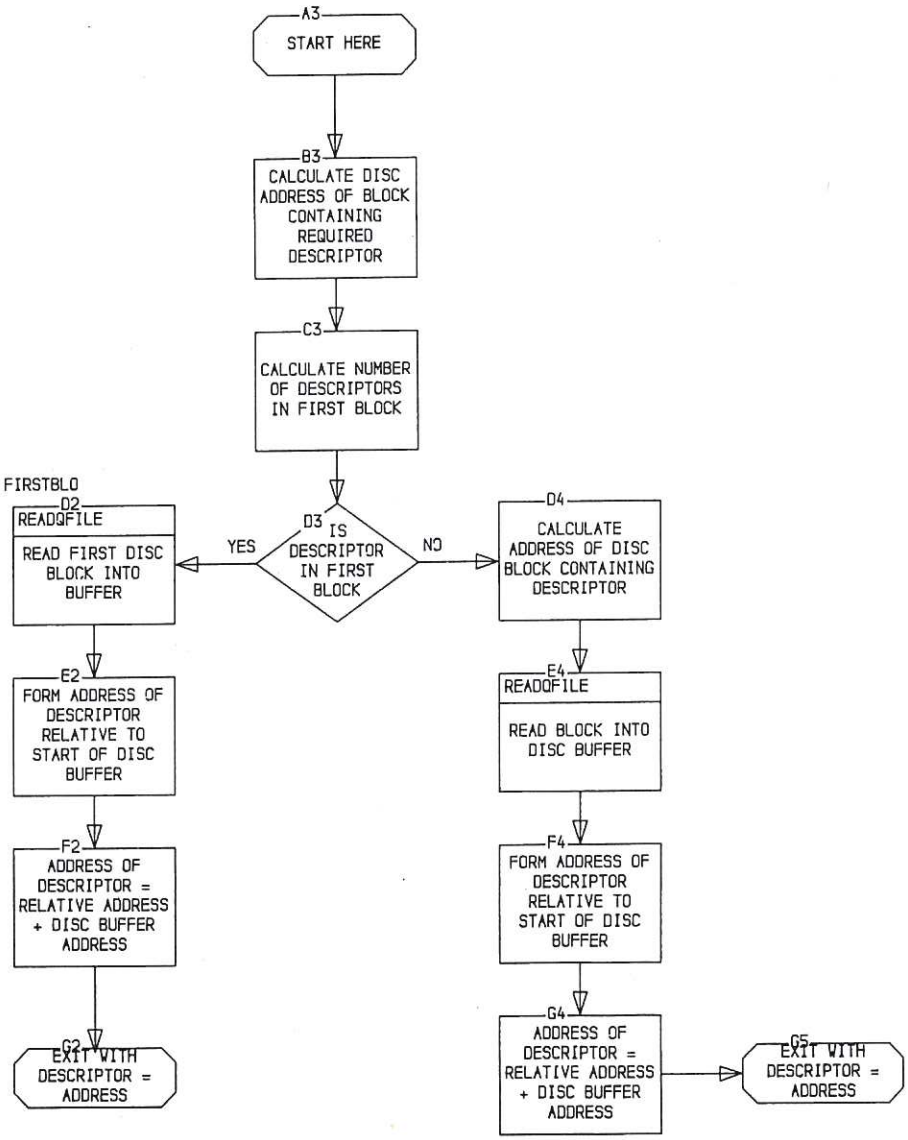



B3
START HERE
C3
DESCRIPTOR
GET ADDRESS OF QUEUE DESCRIPTOR Q
D3
EXIT WITH GET Q DESCRIPTOR = ADDRESS


B3
START HERE
C3
DESCRIPTOR
GET ADDRESS OF SERVER DESCRIPTOR S
D3
EXIT WITH GET S DESCRIPTOR =ADDRESS

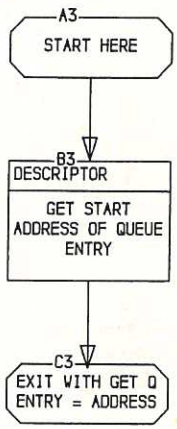
GENERAL PURPOSE QUEUEING MECHANISM D A FOX, T LANG, CULHAM LABORATORY, ABINGDON
FUNCTION DESCRIPTOR (N, LENGTH, START) . MODULE LSD. DESC (S)

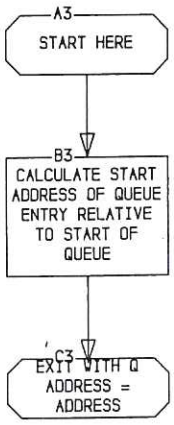
CHART P3

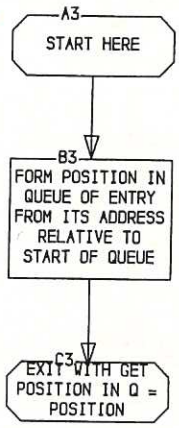


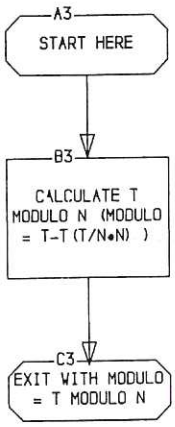
GENERAL PURPOSE QUEUEING MECHANISM D A FOX, T LANG, CULHAM LABORATORY, ABINGDON
FUNCTION GET Q ENTRY (DISC ADDRESS, POSITION) -MODULE LSD.DESC (S)

CHART P4



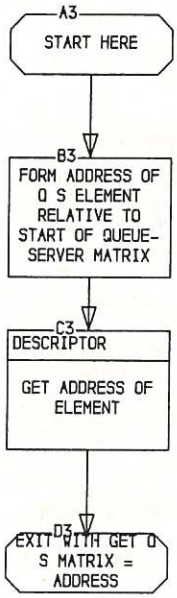

A3
START HERE
B3
CALCULATE START ADDRESS OF QUEUE ENTRY RELATIVE TO START OF QUEUE
C3
EXIT WITH Q ADDRESS = ADDRESS

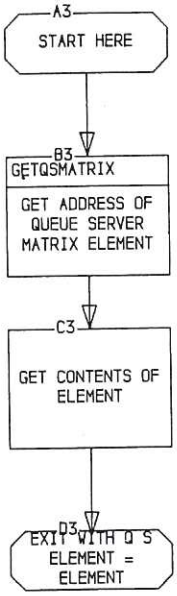

A3
START HERE
B3
FORM POSITION IN QUEUE OF ENTRY FROM ITS ADDRESS RELATIVE TO START OF QUEUE
C3
EXIT WITH GET POSITION IN Q = POSITION

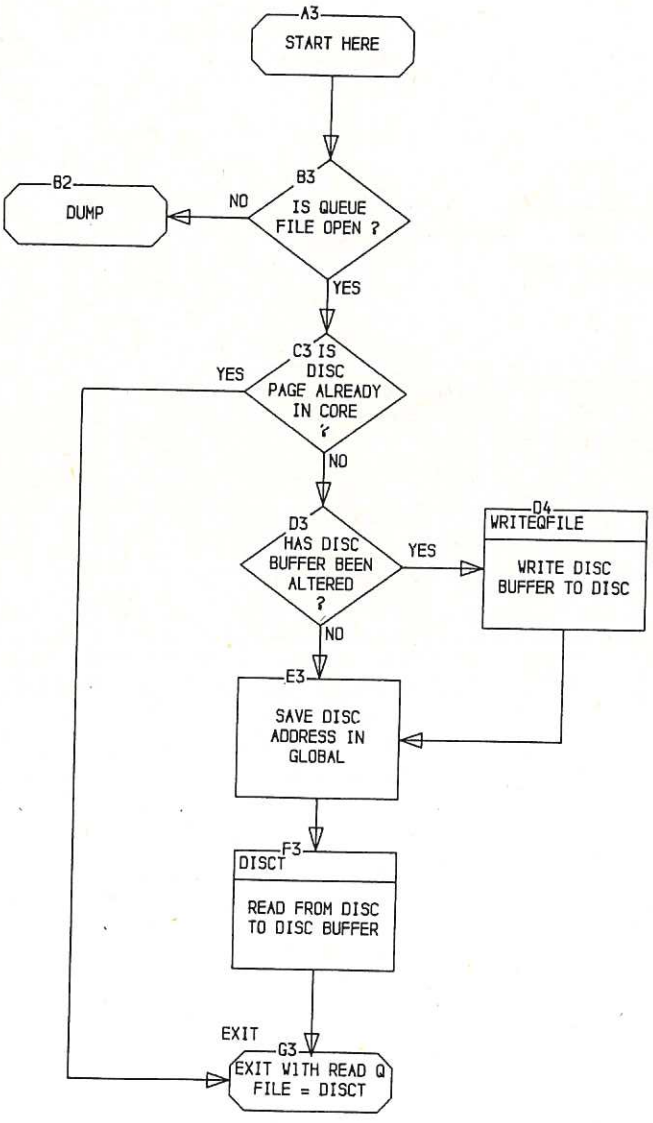

A3
START HERE
B3
CALCULATE T MODULO N (MODULO = T-T (T/N•N))
C3
EXIT WITH MODULO = T MODULO N

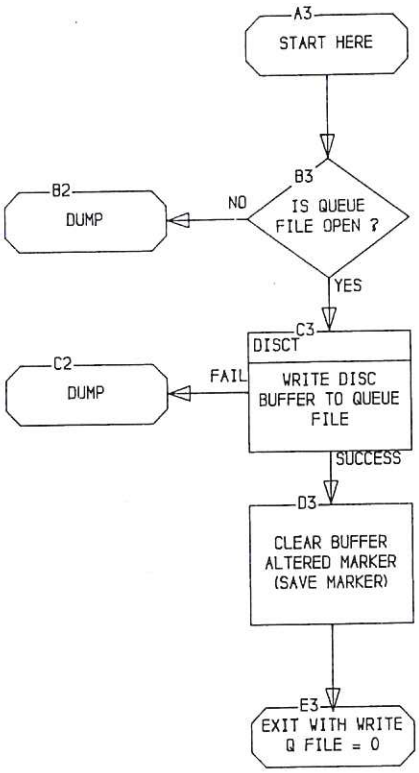
GENERAL PURPOSE QUEUEING MECHANISM D A FOX, T LANG, CULHAM LABORATORY, ABINGDON
FUNCTION GET Q S MATRIX (Q, S) - MODULE LSD.DESC (S)

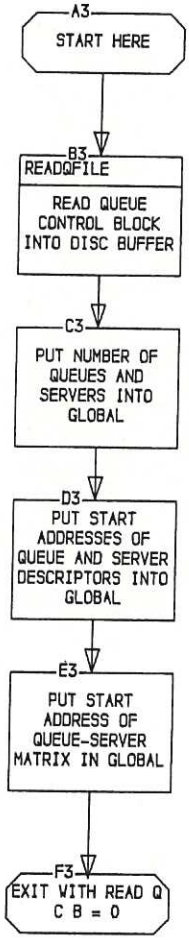
CHART P8

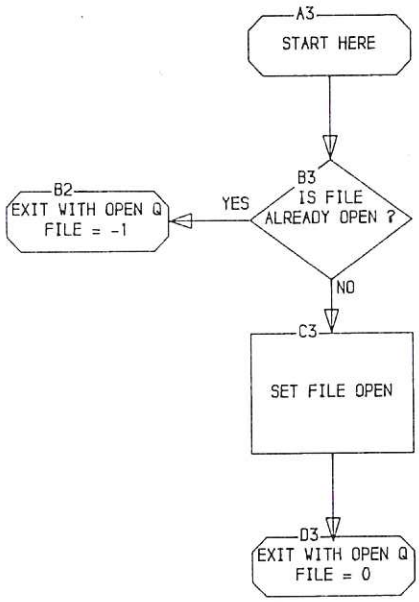


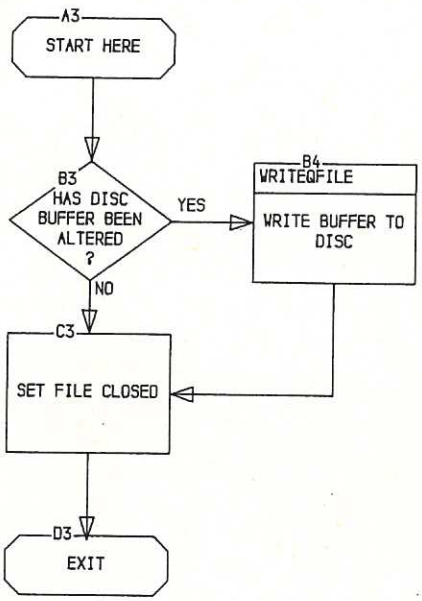

A3
START HERE
B3
GETQSMATRIX
GET ADDRESS OF QUEUE SERVER MATRIX ELEMENT
C3
GET CONTENTS OF ELEMENT
D3
EXIT WITH Q S ELEMENT = ELEMENT


A3
START HERE
B3
IS QUEUE FILE OPEN ?
NO
B2
DUMP
YES
C3 IS DISC PAGE ALREADY IN CORE ?
YES
NO
D3
HAS DISC BUFFER BEEN ALTERED ?
YES
D4
WRITEQFILE
WRITE DISC BUFFER TO DISC
NO
E3
SAVE DISC ADDRESS IN GLOBAL
F3
DISCT
READ FROM DISC TO DISC BUFFER
EXIT
G3
EXIT WITH READ Q FILE = DISCT


A3
START HERE
B3
IS QUEUE FILE OPEN ?
NO
B2
DUMP
YES
C3
DISCT
WRITE DISC BUFFER TO QUEUE FILE
FAIL
C2
DUMP
SUCCESS
D3
CLEAR BUFFER ALTERED MARKER (SAVE MARKER)
E3
EXIT WITH WRITE Q FILE = 0


A3
START HERE
B3
READQFILE
READ QUEUE CONTROL BLOCK INTO DISC BUFFER
C3
PUT NUMBER OF QUEUES AND SERVERS INTO GLOBAL
D3
PUT START ADDRESSES OF QUEUE AND SERVER DESCRIPTORS INTO GLOBAL
E3
PUT START ADDRESS OF QUEUE-SERVER MATRIX IN GLOBAL
F3
EXIT WITH READ Q C B = 0


A3
START HERE
B3
IS FILE
ALREADY OPEN ?
YES
B2
EXIT WITH OPEN Q
FILE = -1
NO
C3
SET FILE OPEN
D3
EXIT WITH OPEN Q
FILE = 0


A3
START HERE
B3
HAS DISC BUFFER BEEN ALTERED ?
YES
NO
B4
WRITEQFILE
WRITE BUFFER TO DISC
C3
SET FILE CLOSED
D3
EXIT


B3
START HERE
C3
UPDATE GLOBAL MARKER (SAVE MARKER) TO SHOW THAT DISC BUFFER ALTERED
D3
EXIT