Vol. 67

National® Brand

College Ruled White Paper
Single Subject

Dennison National Company, Holyoke, MA 01041

31-987

80 Sheets/11 x 8½

8

3-10-88 HPO Internals (cont)

PMA Implementation - CP uses Single Processor Mode (SPM) & guest can use both processors with real DAS feature. Guest runs in true supervisor state. Guest gets apx. 95% of native throughput.

Needs:

HPO Software

PMA Hardware

CR6 & extended MICBLOK

& GPSW instruction.

Guest controls channels not in DMKRIO. CP owns interval timer & is how CP gets control back. The contents of the other timers are swapped. Guest storage must be V=R. Test Block instructions are trapped to prevent MVS from varying storage online. CP traps SIGPs & won't let MVS stop the processor. Ext interrupts are trapped.

If the guest turns on the PSW wait bit a specification exception is reflected to CP. Absolute Ø is owned by guest & HPO knows not to use it if he is running a PMA guest.

C616 New instruction GPSW causes PSW from MICBLOK to be loaded. PMA does Control Switch back to CP. PMA directs interrupts according to mask in MICBLOK. At control switch, CPs control Regs are loaded & CP dispatched with a pgm chk.

CP processes int w/ DMKPMA -

code 27 - enables for I/O & Ext

06 - MVS in vol. wait

02 - Handles privop

3-11-88

Gary

AP/MP Considerations

HPO only supports 1 or 2 processors.

The prefixed PSAs are protected by convention.

Prefix - if address = ØØØxxx then prefix reg kicks in. If address = FFFxxx then address = ØØØxxx.
↳ matches value in prefix reg

Prefixing allows access to the prefixed PSAs.

Reverse prefixing allows access to absolute PSA.

The other processor's PSA is just an area of real storage. There is no hardware protection.

Absolute PSA - sys wide fields are used here:

'C' Tracstrt -

'1Ø' Tracend -

'14' Next Trace Entry

CPID

Mon Buf ↑

CP Restart Status Word

Prefixed PSAs

Copy of other guy's prefix reg value

AP Stat1 - AP, MP, IO Capability, IPC flag

CPINITD - initialization done

Processor address - IPUADDR(x) ← CPUADDR + Y

Trace Entries are identified with

'ØØ' - IPL Processor

'4Ø' - Non-IPL Processor

Instructions Compare & Swap
" Double & Swap
Test & Set are used to prevent simultaneous updates of the same locations in main store. Example:

```
Trace   L     R3, Traccurr
↑       LA    R4, 16(R3)
        CS    R3, R4, Traccurr
        BNZ   Trace
        MVC   0(16, R3), Data
```

Software Locks = serialization by convention.
Defer Locks
Global System Lock - dispatcher
VMBLOK Lock -
stack a CPEXBLOK for what you wanted to do if you could have gotten the lock & go find something else to do (run user dsp...).

Spin
Free Storage
Run List (TRL)
Timer Request
Dispatcher List - update or de-q
TRQ
I/O - de-q or add IOBLOKs to queues
Real Storage - coretable & paging updates
Private Lock - RDEVBLOK access

Spin Lock Hierarchy - cannot request a higher lock while holding a lock.

SY - Global

RM - Rcd Stor, I/O (I/O)

TRL - Disp, RL - Runlist

TR - Timer

DS - Dispatcher Request Queue Lock

Private

FR - Free

DMKLOK - lock manager for AP/MP.

DMKLOCK - is a UP lock manager for use with the Directory and VMBLOKs. - Occassions where CP routines might suspend an action (wait for a page) or directory updates.

Lock manager holds lock words (4 words ea).

Spin loops are in DMKLOK. - (enabled for Ext. int.)

1st word - Ø = free, Logical ID (CS instr)

2nd - R12 of owner

3rd - MS spent spinning

4th - # spins for this lock.

Lock macro tries to get a lock & if fails, branches to DMKLOK to spin.

LOCK OBTAIN, TYPE=SYS, SPIN=YES

RELEASE VMBLOK NO

Releasing a lock you don't hold will abend the system.

FREE

RL

TR

DS

DSX

IO

RM

TRL → TRLX

PRIVATE

Global System Lock -
- Process <P commands
- Dispatcher holds while working on IOBLOK, TRQBLOK, + CPEXBLOK. - All CP functions
Only FLIHs don't need global.
- Access to all system virtual storage.

VMBLOK - prevents both CPUs from working on the same virtual machine simultaneously (located in VMBLOK).
- DSP holds before dispatching any mch.

tele: 982-4466

Unix for Programmers (AT&T)

5-10-88

Joe Nicoletti

Maximum of 25 background processes.
Terminals are connected to the shell. The shell is a command interpreter.

Workbenches:

Documentors - laser prt, etc. Embed format commands in text

Programmers -

Instructional - CAI: Basic Unix
VI
user supplied material

Writers - analyze text - spelling, usage...

Pipelines permit a building block approach to data manipulation. This increases command efficiency and reduces custom programming tasks.

Documentation comes from:

Customer Information Center
P. O. Box 19901
Indianapolis, IN 46219
1-800-432-6600
Call for catalog of Unix doc.

Terminals: ASCII, full Duplex; screen displays are echoed from host. All communication is character oriented.

Two levels of passwords are available (though optional on dial-up networks). Also, password aging is optional.

stu13 password = Student.

During logon, # is backspace and
@ is line erase.
These change after logon.

The DELETE key terminates a running process.
CNTL-D is logoff.

All oper files will be closed by the system.
Memory contents are not written.
Dial-up lines may disconnect on inactivity.

Mail is read FICO.
End mail entry text with a period (.) or CNTL-D.
syntax:
mail userid
mail userid userid ...
mail userid < file
mail `cat files` - the file contains the userids which are "printed" by the cat command.

mailx is more versatile + is more useful for mass distribution.

mail node!userid

A user can be dynamically notified of mail arrival if a check mail variable is coded in .profile.

The standard shell is the Bourne shell. It functions both as a command interpreter and a pgm language.

By convention, commands are lowercase. Options are usually single characters & are preceeded by - or +. Some options may be strung together with no spaces (-lvc), other require spaces: -l -v -c. (- means on; + means off)
Arguments are usually file or directory names.

Each time the shell scans a cmd line, it removes a level of quoting. This can make for some interesting interpretation. If something doesn't work, add a level of quoting.

I/O Redirection

stdin - standard input - keyboard
stdout - " output
stderr - " error output
These are treated as files (special files). Thus, input & output can come & go to files.

$ pgm < datain >dataout

$ cmd > file → output to file. The file is written from the beginning, thus destroying anything which might already exist.

$ cmd >> file → append.

Pipes

Pipe symbol appears between commands & permits output of first command to be used as input to the next command.

Do not confuse I/O redirection with pipes.

I/O redirection cannot be used. This wipes out the pipe contents. Use tee cmd to save pipe contents into a file.

stty command informs shell of your terminal characteristics.

man - comes with documentor's workbench. It uses the DWB routines to format Unix documentation.

File System

Everything is a file:
- terminal
- printers
- Directories + even
- files.

Three formal types:

Ordinary - data, pgms
Directory -
special - identify hardware devices + software drivers. All special devices are in /dev directories.

Output may be directed to a terminal by using its filename:
/dev/tty32

A fullpath name begins with the root.
A file may be "owned" by multiple directories.

To keep stuff from running off the screen, pipe cmd output to pg.

ls > pg

ls -a - all files, including dot prefixed.
ls -l - long form

Dot Convention -
dot is the current director.
dot dot is one above.
All others must be access by name.

../H → go up 1 dir & down to H
P/S → go down to dir P & then to S
../../C/J → go up 2 directories,
down to C
down to J.

ls
pwd } used together for changing directories
cd - default is home

find very powerful

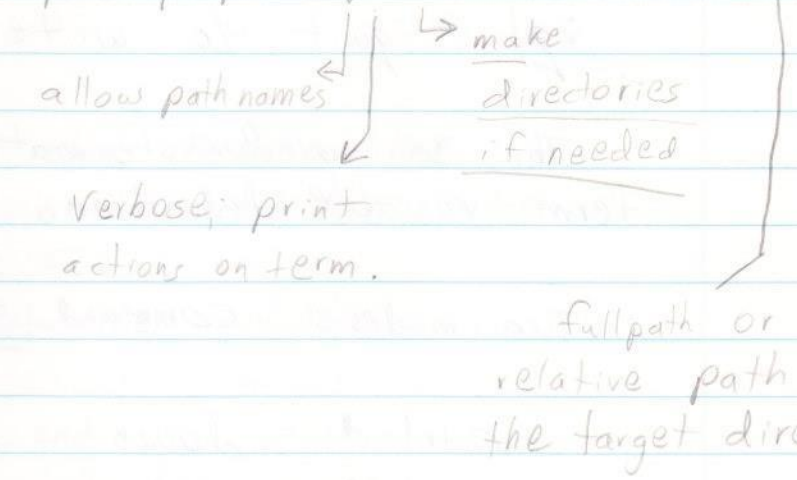
find . -name name-to-find -print
option
↖ starting directory
to display
Returns with full path name.

To copy a sub-directory structure:

1. Make a directory to copy into
   mkdir dirname

2. Goto source dir:
   cd sourcedir

3. enter
   find . -print | cpio -pvd dirname



File Names:
Max 14 chars.
dot (.) masks file from plain ls.

VI Editor

Joe @ UC Berkeley developed vi.

vi filename -

ZZ - write buffer to filename & exit

:q! - quit w/o write

The vi window format depend on the term variable definition.

Two modes: command & input.

ctrl-d - down
u - up
f - forward
b - backward

Cursor control keys position the cursor.
Command mode commands (not echoed):

a - append after cursor
A - " at eol
i - insert before cursor
I - " @ beginning of line
o - open new line below cursor
O - " " " above "
x - delete character above cursor
rc - replace char w/ next char
~ - change case of letter

r<CR> Split line at cursor

[n]J Join line at cursor & next
n = # of lines beneath cursor
to be joined.

u - undo last command

U - undo all changes to current line
These changes are preserved
in the undo buffer

Ctrl-L - refresh screen

Short Cuts

^ - goto first char on current line

$ - last char on current line

nG - Goto line # n
Default = last line in file.

w - next word

e - next word end

b - previous word beginning

5w - skip 5 words

W - skip over punctuation.

E - " "

B - " "

A commandline is not present unless required. Entering a / for search will cause cursor to jump to line 24 for entry of search target.

/ - forward search

? - backward

n - repeat search

d object - delete text up to object
dw - delete up to next word.
d/pattern - delete to next occurrence of pattern.
d$ - delete to EOL.

A line of text may be moved by deleting it into the undo buffer & then using PUT to bring it in elsewhere.

To copy a line, YANK the line into the undo buffer & then put it.

. = repeat the last cmd which changed the buffer.

ex

vi is a front end to ex. All ex commands begin with : These cmds are entered from within vi

:w - write w/o quitting
:w newfile - write to a new filename
:w! oldfile - overwrite existing file
:w >> oldfile - append
:e! filename - edit another file
:f - report filename & line #
:q! - Quit w/o writing
:r filename - read file into current file @ cursor (copy)

ex is necessary for search & replacement.

:s/oldpattern/newpattern/ - on current line, first occurence
:s/ / /g - " " " all " s
:1,$s/oldpat/ /g - all occurences, all lines
:1,$s/ / / - first occurence, all lines.
:5,25s/ / /g - all occurrences, lines 5 through 25

The delimiter can be changed to any character to facilitate changing slashes.

:!cmd - execute a shell cmd.
:r !cmd - read cmd's output into buffer

Metacharacters may be embedded in a string to substitute meaning:

^ - beginning of line anchor
$ - end of line anchor
. - any single character
* - any or no occurences of preceeding
[xxx] - any char in set
[^xxx] - any char not in set

/^time/ - find word time @ beginning of line
/7.7/ - find 737, 747, 757 ...
/[Tt]he/ - find The or the, Their, their ...
[1-5] - find 1, 2, 3, 4, or 5
[3-57] - find 3, 4, 5, + 7

.* - means any character

vi Options

:set - list options in effect
:set all - see option list
:set optionname - turn on option
:set noOption - turn off option

:set num - turn on line #
:set nonu - off
nomesg - block writes while in edit
showmode - display mode
ai - automatic indent
sw=n - shift width for ai

Setting the Login Environment

env - list variable settings. Variable names are capitalized.

userid & password are verified against /bin/login and then executes /etc/profile. Finally, the individual user's .profile is executed. The profiles are shell pgms.

EXINIT = - editor options
PATH = - search order to find commands.
Paths are separated by :
HOME = - Home directory

current directory is not always searched first.

if : comes first in list, search cur dir first
if :: is embedded in the list, search cur here
if : is at end of list, search cur dir last

To use a variable in a shell pgm, use the variable name preceded by $

echo $PATH

When some variables are redefined they won't take effect until an export is performed. Good rule is to export all variables whenever they are changed:

TERM=5420 - a most important variable.
export TERM

LPDEST = default line prt.

Placing calendar cmd in .profile will cause system to look for a calendar file in the login director + can be used to schedule events - date plus one line of message.

Mail

In the home directory, a file .mailrc can contain mailbox groups. For ex -

group students stu1 stu2 ...

+ then utilize in mailx -

mailx students

A single file may be shared via links.

File Maintenance

Shell metacharacters can be used in looking for file names. And they differ from vi & ed.

? - match any single char (may use multiples).
* - match any sequence of chars
[ ] - match any char in set
[0-9][0-9] - searches 00-99

ls * - lists all files in all directories

mkdir - make directory. Requires only 32 bytes to make a directory. If cmds will be placed in this subdir, update PATH in .profile.

cp file1 file2

If you take a file, you own.
If you are given a file, the giver remains the owner.

link command

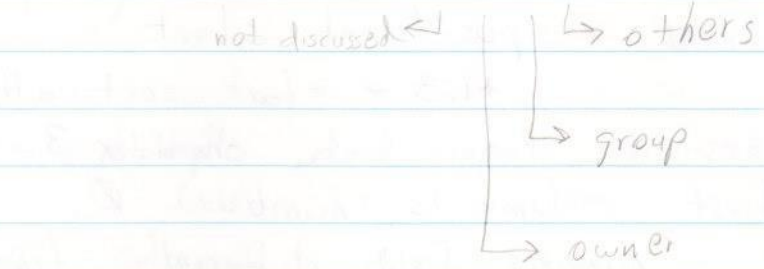
ln - everyone w/ a link has owner permissions. Data security is poor.

File Permissions

/etc/group defines group ids.
write permission is ownership.

chmod - changes permissions
chown - change ownership
Only the owner can execute chown command.

Default file permission mask -
umask cmd → ØØ22



For directories:
777
- 022

755

Determine the permission, subtract from 777 and issue umask cmd.

A user needs read permission on a directory as well as execute permission on a file in order to execute the pgm.

5-12-88

grep - goes into a file looking for a "regular expression" & if found, prints the whole line. The regular expression may contain editor metacharacters.

The target file may be specified using shell metacharacters.

grep 'Tony Bennett' *
↑ search all files in cd

sort - defaults on first character of first column. This is an alphabetic sort.

-n - perform numeric sort (may only be used w/ numerics, -$+.

-f - ignore letter upper/lower case

+pos1 - positional start

+1.3 → start sort after column 1, character 3.

The first column is numbered Ø.

cut - numbers fields differently from sort! permits the contents of a column to be removed. Fields are numbered beginning 1, 2, 3

This command can identify spaces as the column delimiter and may then extract any particular item.

paste - places the contents of multiple files side-by-side.

uniq - eliminates duplicate lines from sorted input.

cat file1 file2 | sort | uniq

tee - I/O redirection cannot be used inside a pipe. The tee cmd permits intermediate result to go on disk and another copy continue down the pipe. I/O redirection may be used at the beginning and end.

; - multiple cmd input on one line.

Command Grouping

Sequential cmds (separated by ;) have their output collected together.

(date ; banner 'HI') > testfile

Background Execution

Type an & at the end of the commandline. The sys returns a PID on the background process.

nohup - if this preceeds the cmd, the background process continues to run even if the user logs off.

Be sure to redirect output of background process. Remember that standard error may need to be redirected.

2> filename

ps - lists the PIDs

kill pid - terminates process

Shell Pgms

Placing executable commands in a file, & making it executable (as well as putting the directory in the cmd search path) results in a file that acts as a cmd.

All items in the cmdline are variables:

$0 is the cmd

1, 2 ... are operands.

To use the value of these variables, refer to them as $1, $2 ...

The variable * (or $*) is the string of the arguments.

Communications

uname - lists node name of machine

uuname - " " names of connected machines.

To send remote mail, simply precede the user id with the node name, separated by !

mail vulcan!spock

Multiple nodes may be specified.

uucp - does a copy. Good only for ASCII

uuto - copies files to a public directory:
usr/spool/uucppublic
Good for binary files.

uupick - retrieves files from the public directory.

uustat - reports status on transfers.

cu - call Unix. permits logging in to a remote machine (assuming you have a userid on that machine). cu can also dial a telephone too!

Programming in C

6-15-88

Stephen G. Kochan
(Indianapolis: Hayden Books, 1988)

Program files must end with the characters .c
To compile & link:

```
cc filename.c
```

The load module may be executed simply by entering its name.

Lowercase and uppercase letters are distinct.

Program Structure

C statements need not be placed in any particular column.
Comments are enclosed in:

```
/*          */
```

pairs (which may span multiple lines).

```
main ()
{
    printf ("Hi \n");
}
```

The main statement indicates start of program execution. It is required. The brackets indicate arguments to be passed at invocation.
Brackets {} enclose all statements in a routine.
All statements end with ;
Blank lines are permitted.

printf is a function call. C language does not have any intrinsic I/O commands. I/O is apparently dependent on the implementation. The standard I/O library contains standardized functions, like printf, which are customized for particular systems. Externally, all functions in the standard I/O library should perform alike.

Declaring Variables:

Integers

int var1, var2, sum, garbage, ... - integer

int var1 = 120

Integer variables may be expressed in either octal or hex in addition to decimal.

Octal - identified by a left-most digit of zero!

int var1 = Ø5Ø → equals 50_8

int var 1 = 5Ø → equals 50_{10}

Hex - two left-most digits are ØX

int var 1 = ØX4EF → equals $4EF_{16}$

Decimal values are displayed in printf with %d

Octal %o

Hex %x

Floating Point

```
float  var1 = 125.8
       = -.ØØ1
       = 6.29e27
```

To display a floating point value with printf, use %f
" " " scientific notation " " " %e

Character

The char variable can only store a single character.

```
char  char_var1 = 'J'
      = '\n'      ← newline character.
```

use %c to display a character variable with printf.

Extended Accuracy

```
long int = var1 = 2579494L;
      or
long = var1 = 2579494L;
```

These values may be displayed in printf with %ld
%lo - octal
%lx - hex

Also, variables may be designated as short:

```
short = var1 = 13
```

An unsigned type also extends accuracy and can be used where the variable will always hold a positive value.

```
unsigned int var1 = ...
        or
unsigned var1 = ...
```

> The accuracy of all stored variables is machine dependent.

Arithmetic

*	multiply
/	divide
+	add
-	subtract
%	modulus - give the remainder

```
a % b = c
25 % 10 = 5
```

Blanks may be placed around operators for readability. A program statement may be continued on another line at any point where a blank is permitted.

If a floating point value is assigned to an integer variable, it is truncated.

If either value in an operation is floating point, then the operation is performed as a floating point operation.

8-3-88

To print a % sign within a printf statement, use %%. A % used within an expression is the modulus operator and may only be used with integer variables.

8-10-88

Looping

initial value

Continue looping until this is false

```
for (init_expression; loop_condition; loop_expression )
{
   program statements;
}
```

bump counter after program statement.

Braces are optional if only one program statement is used.

```
for ( n = 1; n <= 200; n = n + 1)
    triangle = triangle + n;
```

Order of evaluation

1. Set initial value.
2. Check loop_condition + if false, terminate.
3. execute program statement.
4. perform loop_expression.
5. Goto step 2.

Loop conditions (beware)

n == 200 checks to see if n is equal to 200.
n = 200 assigns the value 200 to variable n.

Loop Expression

++n is equivalent to n = n + 1

--n is equivalent to n = n - 1

Printf

The number of characters expected for a variable may be specified to preserve right-hand justification of lists:

```
printf ("%2d    %d\n", x, y);
```

would result in:

8	10
9	9
10	8

Scanf

The scanf function reads input from standard input.

syntax:

```
scanf ("%d", &number);
```

The & is required

for loops may be nested to any required depth. Loops may be explicitely terminated with a return, break, or goto.

While Loops

syntax:

```
while (expression)
{
    program statements;
{
```

So long as the expression is true, the program statements will continue to be executed.

Do Loops

syntax:

```
do
  {
    program statements
  }
while ( expression );
```

The program statements are executed prior to the test of the expression. If the expression tests true, the program statements are executed again.

The break statement immediately terminates a loop. If the loop is nested, only the innermost loop ends.

The continue statement ceases execution of the program statements found in a loop, but loop execution continues with another iteration. Continue is used to bypass code within a loop.

8-11-88

The variable type may be temporarily modified for the purpose of a calculation:

```
int   x, y;
float z;

z =   (float) x  / y ;
```

Only one variable need be floating point in order for the operation to be carried out as floating point.

If x and y were left integers the result would be integer even though z is defined as floating point.

The number of significant digits to be displayed in a printf statement for a floating point variable can be controlled via:

```
printf ("The result is %.2f \n", x);
```

This is the precision modifier and indicates the number of digits placed to the right of the decimal point.

If Statements

```
if (expression)
   program statement;
```

```
if (expression)
   program statement;
else
   program statement;
```

Note semicolons

The program statement can be any valid C statement including another if.

```
if (expression)
   if (expression)
      program statement;
   else
      program statement;
else
   program statement;
```

An else is associated with the immediately preceeding if. The else & if are paired. If an inner if does not have an else, then it must be bracketed in order to isolate it from a preceeding if.

```
if (expression)
{
   if (expression)
      program statement;
}
Else
```

pair because of brackets

Compound Relations

The relationship tested by an if expression may be compounded by logical ANDs and ORs.

```
&&  = AND
||  = OR
```

```
if (x >= 70 && x <=79)

if (x >= 70 || x== 0 )
```

These may be quite complex:

```
if ((x == 0 && y != 0) || z == 0 )
```

Note the use of inner parentheses to clarify the logic. The order of evaluation is arithmetic, AND, and lastly OR. Parentheses are highly desireable.

The program statement following an else may be any valid C statement, including another if! In this case, it is usually written as:

```
if (expression)
    program statement;
else if (expression)
    program statement;
else
    program statement;
```

The else-if construct permits three way choices to be made concisely.

Multiple choices can be performed even more concisely using the <u>switch</u> statement.

```
switch (expression)
{
    case value1 :
        program statement
        program statement
        break;
    case value2 :
        p/s
        break;
    case value3 :
        ↓

    default :
        program statement
        program statement
        break;
}
```

The value in expression is successively compared against each case, and when a match occurs, the program statements under it are executed down to the break.

The program statements need <u>not</u> be enclosed within braces!

← A default clause exists to catch a variable that does not match any case.

Flags

1 = True
∅ = False
True ≠ ∅

Flag variables may be tested implicitely:

```
if ( x)
   ⋮
```

is equivalent to

```
if (x != ∅)
   ⋮
```

```
if (! x)
```

is equivalent to

```
if (x == ∅)
```

The not operator may be used to toggle a flag:

```
x = ! x
```

Conditional Expression

condition ? expression1 : expression2 ;

example:

```
s = (x < 0) ? -1 : x * x;
```

If condition is True,
then s = expression1.

If condition is False,
then s = expression2.

The parentheses around the condition help readibility. They are not required.

```
max_value = (a > b) ? a : b;
```

The expressions can be any C statement, including another condition expression!

```
sign = (number < 0) ? -1 : ((number == 0) ? 0 : 1);
```

will result in the variable sign having a value of -1, 0, or 1. The parentheses are optional. Evaluation is peformed from left to right.

UTS System Administration

Brian Jenks
8-16-83
Columbia

System Configuration

/etc/devicelist is input to sysgen shell script which includes a pgm config which does most work. Dozens of files are output including kernel + dev. directory.

Then script newsys installs this stuff. /uts + /dev/* are created + backups are created. Backout is possible.

/etc/devicelist - just a plain file + is easily modified. This drives the config process. Some things have nothing to do with I/O, like XA/370, time zone, uucp nodename, etc.

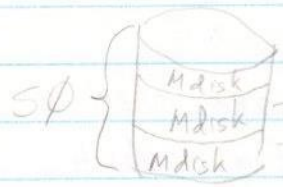
sysgen script - does the work.

newsys - script to archive current kernal + dev + installs /nuts (kernal) + /dev + re-IPLs. Executes in superuser mode.

Read Shell Command + Programming.

/etc/devicelist - uses a lot of VM terminology.

A DASD device as a whole is slice 0 + can be subdivided into minidisks S1, S2, ...



A filesystem occupies a slice. Having multiple filesystem is good for
- reliability (they can break)
- moving + backing up
- tuning

Striping is optional.

range
hexadr [-hexadr] dasd [mountdir

110 dasd /usr

decimal
dskadr mdisk blk-offset blk-count mountdir
4405 1 mdisk 0 10000 /usr/fun -r

↑ zero, meaning the first block that UTS can use.

0 for length means until the end of the pack.

Supported Devices:

6280	120 blks/cyl;	833 cyl,	409 M/vol
6380	150	885	543
6680			
2305-1	24	48	5
2305-11	24	96	9
3825 - in 2305 emulation			
3350	120	555	273
3380	150	885	543

The device must be defined in the list before it is sliced.

tubes - local

hexadr [-hexadr] 3xxx tube runlevels

620-63f 3278m2 tube 2

↑ this is a label which is used in creating another file.

remotes are more complicated:

hexadr line ← must come first

slctadr [-slctadr] r3xxx tube runlevels

Ø22 line

6ØØØ-6Ø1f r3278m2 tube 345

32 devices on 47Ø5

Determines polling, not adr!

6ØØØ-6Ø18 is only 9 devices!

Polling Seq.

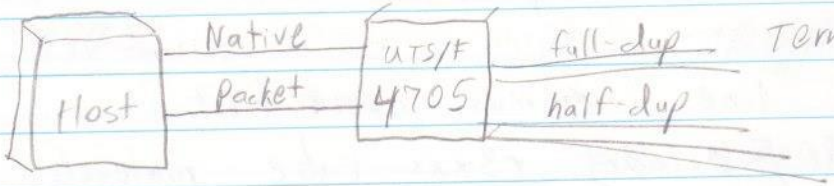
4Ø	5Ø
C1	D1
⋮	⋮
C9	D9
4A	5A
⋮	⋮
4F	5F

Cluster printers are treated like tubes:
3287, etc.

Chan att printers: 1403, 3203-5, 3211.
00e printer

half-duplex - 1 line, bi-directional
full-dup - 2 lines, 2 addresses.

Native subchannel for 4705 - to communicate with the box as a device itself - for loading, msgs...
Data travels on the packet subchannels. These are defined separately.
010 fep # Native SC
062 fep # Packet SC
They are defined identically



UTS/F builds packets from byte terminal input & relays packet to host w/ added tag info.

Tapes

180 - 184 tape

↑ by tradition + bootstrap assumption.

For 580, a ACS erep interface is defined:

091 acs

IOCDS definition:

0 mssf

Config finds the driver for each type of device + then builds a file (special) to connect to the driver when each device is opened. For DASD, a special driver exists to format whole disks (not slices). Only one format driver exists, + it doesn't know about slice sizes.

Pseudo-Devices

vio - defines virtual files in main store.
Memory-resident virtual disk.
0 vio /tmp } multiples
1 vio }

sxt - shell layering + multiple sessions.
tbs(1) + shl(1)
Permits logging on to multiple sessions.

ucblock - Ingres - tape management for bkup/restore
needs pseudo device.

diag - permits use of cp diag commands from UTS. See vipl (1m)

After a device is defined, its useage is declared:

Can all be slices.

- root -
- dump - must be formatted w/ make dump command - automatic
- page - not a file system
- swap -
- pipe - holds pipe files. Root is used if pipe not def.

shared - range of addresses on cu that share a subchan
unshared - " " " do not " " "
altch - alternate chan for 370 mode.
altcu - alternate cu for 370 mode.

System Table Parameters - the non-I/O stuff in devicelist.

nusers - max users. Default is # of terminals. Can be two low if multiple sessions used.

maxuprc - max # of concurrent processes for any one user. 25 should be ample (is default). Causes kernel to increase.

Inter Process Communications (IPC) - facilities can be turned off. Should be on.

- mesg - messages
- sema - Semaphores
- shmem - shared memory

Environmental

dstflag - Daylight Savings Time flag (rules)
timezone - offset from GMT in minutes
localtod - is system using local (1) or GMT (0).
nodename - system's name on uucp network.
sysname - named returned by uname.
version - to override default.
dlimit - max filesize permitted to user. (ulimit)
autoup - # secs before flushing delayed-write buffers. Default is 20 secs.

Several VM-type parameters can be used.

When the filename is specified on the DASD and MDISK statements, config generates a full name & uses it to mount the device (via mount list).

```
440S1  mdisk  0  10000  /usr/fun  -r
```

read only

causes /usr/fun/440S1 to be placed in the mount list. Sysgen sorts these so the highest level gets mounted first.

Entries in mountlist can be added or deleted after sysgen. Remember to update devicelist too.

edusdØ3 d edutsv3
edutsv3 ussa Ø3 -login
PF1Ø unix, +

IOCDS

macros:

ID - msg1 & msg2 strings

CHPID -

CNTLUNIT -

IODEVICE -

CHPID - Path -

type - BL/BY

CNTLUNIT - cuNumbr -

↳ path

- protocl - D/S DCI or D/S
- shared - Y/N
- unit - type
- optional [unitadd - ØØ-FF range of devices (adr, number)

IODEVICE - address = (adr, number)

- cumumbr =
- optional { model -, path -, timeout = Y/N, unit, unitadr -

The command IOCP compiles the input. The utility dd writes the IOCDS to /dev/mssf. Use output block size of 1280

The command iocpdump < iocdsfile reads the IOCDS hard-disk console file & produces a report.

In XA-mode, UP, DP, & MP may also be specified in an environmental parameter.

Memory Management

Paging - brought-in on demand.
- paged out when required - LRU

370-mode
Users have 64meg } may have changed
Runs DAT off → Kernel has 16 meg } w/ 1.2.2

XA - no distinction between user & kernel.

V.io blks are kernel pages.

Swapping - if paging is insufficient, all pages of a process are swapped out. Swap goes to space defined as swap if available & if not, will go to page. Paging will also spill to swap if required.

The Working Set Scheduler isolates the active working set & permits other pages to be paged out first.

Tune parameters

quantum - how often to take CPU Timer interrupt. Range: 10ØµS - 1 sec
Default is 10Øms which may be too long. Seems like this should be model-dependent.

Scan - how often (in quantums) to check working set. (Reset Ref bit).

Window - how long (in scans) before pages lose working set status. Default = 5

Leaving the working set simply makes it eligible to be paged out.

A process may be considered runnable even when swapped out.

Badness is a measure of how long you have been where you are. If in mainstore, badness increases. If out, badness decreases.

$$\text{Badness} = \text{time} - \frac{\text{wss}}{\text{factor}}$$

The factor is tuneable. The priority of the process does not influence. Swapping indicates a stressed system.

Paging is crudely distributed among the paging devices. For swapping, the wss is kept together if possible.

At swap-in, a subset of the wss is brought in, the "preload". The pages that are only three scans old are brought back.

Process Time Slice - 100μs - 1s range. Determined by CPUTimer interrupts (EXT). Specified in # quantums. Default for slice in 1μs.

I/O int may interrupt the time slice. If an interrupt occurs during the last moments of a slice (min-quantum), the process either loses the rest of the slice or gets another whole quantum. Instructor doesn't know.

Tune parms:

quantum	-q	
min-quantum	-m	
window	-w	
preload	-p	
scan	-s	→ only scan a portion of memory
slice	-S	
select	-c	→
factor	-f	
lowmem	-l	→ thresholds to awaken pager daemon.
highmem	-h	→ " " " " " 1/sec
limit	-L	→ how much of mainstore can 1 process control.
steal	-t	# of pages to steal from a user when that user wants pages!
maxmem	-M	

These knobs won't do much.

Improvements can be wrought through buffer management. Blocks coming & going pass through a buffer. A page may be used right in the buffer. Buffer is cache & is efficient for reading.

The nbuf parm in devlist (see config) watch buffer hits in sar & adjust this number. Default is # of terminals. Delayed write buffers are part of the kernel. Don't make it too big, especially in 370-mode.

8-17-88

Filesystem Administration

UTS blocks are 4K. This is larger than most unix systems. Blocks are numbered from Ø. Block Ø is boot block but is not used in 370-land. Therefore it is unused. This maintains compatibility.

Blk 1 - "superblock" - control block for whole file sys.

Next "I-list" - blks containing I-nodes which acts a file labels - type, date, permission, pointer to file. Name of file is not in I-node. I-nodes are 64 bytes.

Data blocks hold files. No structure is imposed. A record is usually ended with \n but it can be anything the pgm wants.

Superblock -

isize - # of blocks in I-list
fsize - total # of blocks
flock -
ilock - } locks + flags used when superblk is in storage.
fmod - } "dirty bit" sched. superblock for writing.

When filesys is mounted, the superblock comes into storage + is updated. The disk copy is occasionally updated.

state - set active when read into storage + set off when placed back onto disk. If found active at mount, then filesys possibly corrupt.

ronly - read only; used in storage - set when explicitely mounted read-only. Superblock will never flush out. Might be used if filesys is corrupt & trying to salvage. Also, can be a way of sharing a filesys between multiple UTSs (since both would hold in core copies of the superblock). It can be R/W from one of the systems.

time - TOD of last superblock update.

dinfo - device dependant info. Tracksize...

tfree - # unused data blks.

tinode - # free inodes.

label { fname - filesys name. Matched at mount cmd. If ≠ issues warning!
fpack - volser for record keeping. Not used.
fill - filler

magic - a number

type - how big are the blocks. Will be 4

The superblock has an array of 900 entries containing entries of free data blocks (not owned by inode).

nfree - pointer to next entry in array where the next free data block number can be written.
If more than 900 datablocks exist free, then the array is placed in the datablock and the entry Ø points to that list. nfree is set to 1. Entry Ø always points to a freelist in a datablk or = Ø (since Ø is not a valid pointer).

To satisfy a request, decrement nfree by 1 & give the user the data block number found there. If up to entry Ø, read in the data block & then give it away.

ninode - index into free inode list
inode - array of 200 2-byte entries. (offset)

Inode

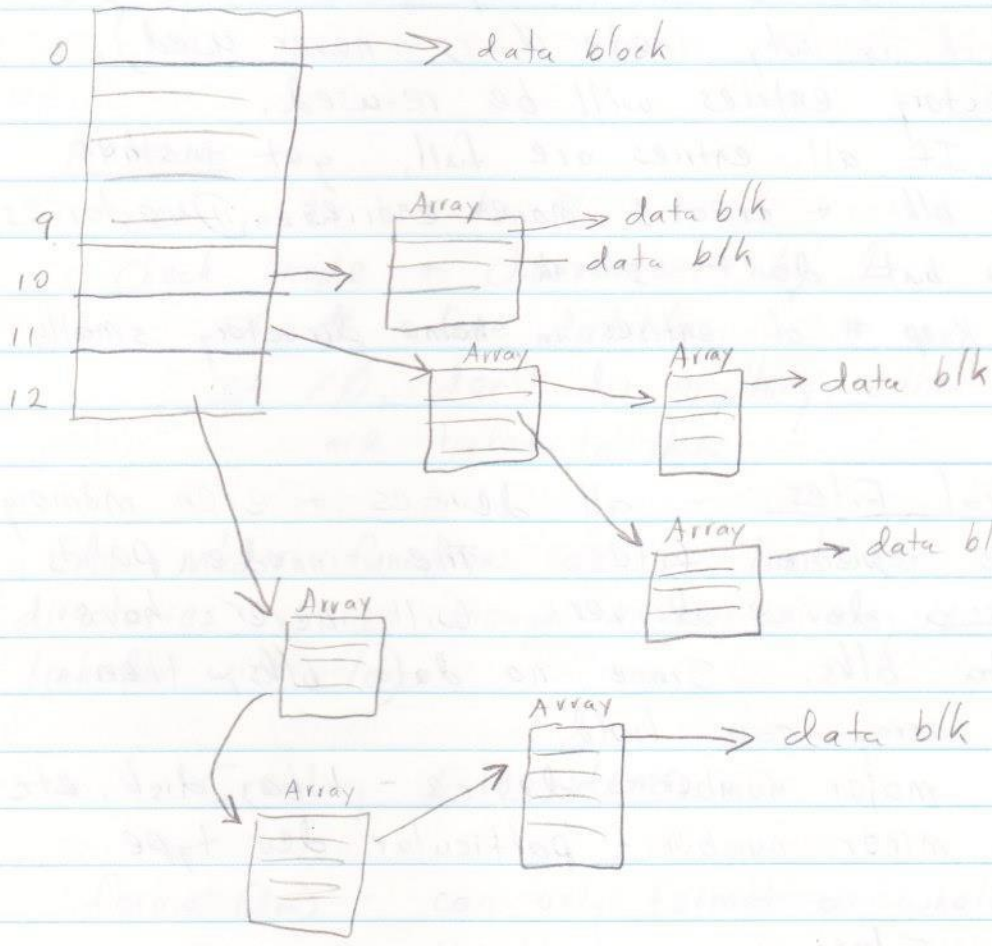
mode - protection mask & file type
If all off, I-node is free.
nlink - # links to the file. # of directory entries pointing. If Ø, then free.
uid - owner id # (Is looks this up in password file)
gid - group id #
size - # bytes in data file. 4 bytes thus limits maximum filesize of 4,294,967,296 which is larger than any disk drive now made. Physical size of the pack is the limit.
atime - last accessed TOD for file.
mtime - " modified " " "
ctime - " " for inode.
addr - array of 13 words. Holds numbers of the assigned 4K datablks. After Ø-9 are allocated, entry 10 points to a data block holding pointers to other data blks.

A recycled # gives new user ownership of old user's files!

special files are Ø

unsigned long

0-9 are direct data blks.
10-12 are indirect datablks.
Entry 11 points to an array of indirect data blocks.
Entry 12 points to three levels of arrays (but is not used because filesize is too big:
4,402,345,713,664 bytes !).



inode 1 is not used in Unix. UTS can use it. When a bad block is found it is placed in inode 1 & appears owned. Max of 10. badblock command is used. fsck will give you the bad block #.

Directories

Are simply files maintained by the operating system. Contains 16 byte entries. 2 byte inode number + 14 byte name. First two entries are . and ..
If the inode pointer for . and .. are equal, then directory is root for filesystem.
If file is removed, inode ↑ is ∅ (which is why inode ∅ is never used). Directory entries will be re-used.
If all entries are full, get another data blk + create more entries. Directories grow but don't shrink.
Keep # of entries in home directory small.

Special Files - all devices + even memory are special files. The innode points to a device driver. Will never have data blks. Since no data blks, then the array can hold
major number - device - tube, disk, etc
minor number - particular dev type

Data Files

Linking Files

Link cmd permits 1 file to be linked to more than 1 user (in same file sys). Also, the same file can have a second name.

```
/n  /usr/louie/xyz  abc
```

link to me + change name

and increment link count.

Remove

set inode # to Ø.
Check inode + decrement by 1.
If Ø, free datablks.
If ≠Ø, don't do anything, other users are linked to this.

An mkdir functions as a link for directories (even though directories may not be linked with /n).

Creating Filesystems

step 1. format (1m) - can only format a whole pack. It writes 4K blocks + verifies that the tracks are ok.

2. mkfs (1m) - make filesystem according to default specs or info produced by
step 1½ mkproto (1m) which checks the characteristics of the raw disk specified. Mkfs uses the output.

It figures out how many data blks will fit on the device + then gives 2X number of inodes.
mkproto also specifies a directory w/ 777 permission.
Also creates a lost+found directory for files found floating around.

mkfs does not read stdin. Therefore, cannot pipe mkproto to mkfs. Use I/O redirection instead. Perhaps edit mkproto output.

mkfs /dev/rdsk/140S0 /tmp/140
target output of mkproto

File sys should not be mounted when mkfs executes since any open fs has a superblock in storage.

labelit (1m) - writes to fields within superblock (fname + fpack). It waits for ten seconds (like mkfs) for typos to change their minds. labelit.now is immediate.

step 3. mount (1m) - Any directory may be a mount point, but it should be an empty directory. Set aside empty directories for mounting file systems. Don't forget to update the mountlist.

file systems exist in the hierarchy, but the mount point is where . and .. are equal (always inode 2).

/etc/mount /dev/dsk/1415Ø /usr/src

The mount point would be at /usr/src.

/etc/umount /dev/dsk/1415Ø - unmounts a file system.

Integrity

sync (1) - flush buffers to disk.
fsck (1m) - filesys checker. Audits & repairs
blocks & sizes.
- cross checks sizes
- checks paths & connectivity
- checks ref counts.
- check free list to account for all data blocks.
- May attempt a repair.

If fsck can't fix it, the filesys is lost.

Note that root cannot be unmounted but fsck will run against it. fsck will create a new superblock on disk and request a reboot no-sync! It does not want the root superblock to be updated by incore copy.

dfsck - will check multiple file systems concurrently. Hopefully the file systems will be on separate packs to avoid head contention.

df -t - shows all mounted file system + total free data blks, + # free inodes.

fsdb - file sys debugger - a raw utility to run against mounted (not recommended) or unmounted file systems.
It can do many things, including displaying a formatted inode:
2i - display inode 2
Syntax of fsdb commands is horribly cryptic

lab: ✓ format disk (2 cyls) 143
✓ - mkproto /dev/rdsk/14350 > protout
✓ - mkfs
✓ - Have Dale mount it.
✓ - create a file under new file sys - readme = inode 4
- run fsdb - inode ✓
- directories 2i.fd
- data blks 4i + fc8
↳ #8
Use cd to get into new file sys.

Shuffling the Filesystem

- out of free blks
- " " inodes
- I/O perf rotten

Create a new filesystem & mount it to the generic mount point /mnt.
Shuffling is easy if not the root.

volcopy - exact replica. Great for backups.
dcopy - reorganizes as copies. Files occupy contiguous blks.
cpio - slow. Moves files one at a time. Good for writing to tape. Recommended.
fscp - allows multiple copies. why??

Root - the root must be expanded to hold enough free space to do maintenance. So, copy root & change device list. Make new vol IPLable. Run newsys & point to new pack. Boot new pack. Format old pack (backup before doing anything), enlarge old root, & copy back.
Root should have, c. 10K free blks.

Plan ahead & give it a larger slice when building first devlist.

Administration Commands

When running as superuser several commands have much more power. Some cmds are even superuser only!

Superuser Only

umount
init & telinit - run limit
mvdir -
dsksize - report size & block numbers for disk.
dpasswd - attach passwd to shell or tty
chroot - change location of root.
Establishes a fake root for a particular application. Points to an alternate director which the pgm sees as root.
su - change superuser to be any other user on the system & doesn't ask for password.
passwd - change passwords.
group - purge or assign passwords to groups.
mount -

Superuser Enhanced

fuser - kill process.
date - set TOD
wall - broadcast messages
chown - assign file ownership
chmod - " uid & gid.
nice - increase process priority
kill - signal process

General Commands

find - filename locator

xargs -

file - classify a file by type - shell script, executable, text ...

disku - report ownership of 4K blks

chgrp - change group affiliation

Site Management

8/18/88

A fork starts an independent process.

An exec begins a new pgm under the same process.

IPL -

The UTS IPL pgm waits for an interrupt from any local tube to find a console. Now, a file called conslist.h can hold addresses of eligible terminals. As distributed, this is empty. Have at least two entries, preferrably on separate 3274s.

The first prompt is:

Enter kernel name:

reply is usually UTS.

Then UTS locates the root dir on slice 0 of the IPL device, reads the kernel in, & passes control to it. The scheduler process becomes pid 0. It forks to /etc/init (pid 1) which is the parent of all other processes. The file /etc/inittab is used by init. Check inittab manual entry for syntax of commands.

Runlevels determine which terminals can be active. Runlevels are arbitrary. In inittab, each tube is associated with one or more runlevels. Runlevel 0 is single-user mode. Level 1 is rarely used. 6 is max.

When the inittab is changed, the cmd init q will cause init to rescan the inittab.

Other scripts called at initialization:

/etc/brc - clears mount table & page dev table
/ /bcheckrc - builds vio; invokes /etc/checkall
/rc - starts daemons
/local.rc - runs site specific tasks, like tape daemon.

Inittab forks pgm getty for each terminal. The pgm getty reads a file /etc/gettydefs. Then writes system nodename & /etc/issue, & login prompt. Then waits for user login. When a person arrives getty execs to login which queries for password. This encrypts the password & matches against /etc/passwd (where all stored passwords are encrypted).

Login then does a cd to the home dir specified in the pwd file & does an exec to the shell, bin/sh (bourne shell). Therefore, the fault shell is whatever is in bin/sh.

When the shell goes away, pid 1 respawns the getty as specified by inittab.

Config generates inittab entries based on the devicelist.

The gettydefs file is also output of config. One entry per term dev type. The label in device table is used to set up gettydefs & is used in the inittab statement as a parm to pass to getty so it knows what to use out of gettydefs. The gettydef string includes the literal "login" prompt.

Strings may be placed in devicetable to pass into inittab. Read inittab and config documentation.

Shutdown - /etc/shutdown

Sends messages & terminates selected daemons & user pids, unmounts the file systems, and ends up in single-user mode. Don't use init s, use shutdown to get into single-usr mode.

The header of inittab controls shutdown & the contents ship as a separate file & config cats them together.

News & stuff

/etc/issue - message before login.
/ /motd - message of the day after login
/usr/news - Directory for news items
/etc/wall - Broadcast msg to all logged on users.

If the director is writeable, any user can put a news item out.

Ned has a limit on line length & if file doesn't have a /n it appears as one record.

Userids

/etc/passwd - 7 fixed fields. Don't directly edit, use utilities.

/etc/identity - variable fields defined by /etc/identity.defs

Adding User - add, mod, & delete are all the same module. Adduser is screen driven.

- assigns an empty uid number (which may match files already in existence.)
- password
- shell
- home directory (does mkdir)
- mailstop ...

Adduser even sents a letter. Customize this: /usr/lib/model/letter
- copies model .profile

Deleting User - two pass

Pass 1 - password is replaced by EXPIRED mmdd... All files are preserved. The userid # still exists.

Pass 2 - Removes home directory & all below it. Removes /usr/mail and entries in /etc/passwd, identity, and group.

Files owned by this user which exist in other directories are not deleted.

Userid # 0 is superuser & is slipped in via ned.

41
Normandy
Astronomer
cars
banker
pgmmer
relgez

pwck - checks the password file for gross errors.
grpck - checks /etc/group

Security Considerations

Since PF keys can be downloaded from the host, it is possible to send a msg to a user which contains the esc sequences to program these keys. The screen looks like garbage, but "most be line noise." Then the unsuspecting victim presses a newly redefined PF keys.

Use mode bits.

With setuid on, when a pgm is exeated the owner is the real owner regardless of which process executes it. With setuid off, the executing process is temporarily attributed as the owner. The same applies to gid.

PF10 retrieves the last command which includes the superuser password. Don't leave terminal unattended.

Keep cron's directories secure. Otherwise, if the cron field separator is set to slash, when it connects to your directory it will execute /bin/xxx which will now appear as a cmd called bin which may be in your dir & is now run w/ superuser power.

Never run a user pgm from a superuser session.

Don't use PATH :: /bin/local
                ↑ current working dir.
since you might cd to a user & then do ls which the user may have a pgm called ls which now executes under superuser.

Tape Management

This is an add-on to Unix.
UTS will respect OS vol labels. It reads the volser and checks for a match but does not even read the dsn.

Under VM, the tape daemon can communicate with a tapeopr id under VM.

User issues tape mount command to master tape daemon which selects a drive and starts another daemon. This, in effect, creates a special file /dev/tape/volser which the user now references instead of having to know the physical device selected.

tpdaemon - started by /etc/local.rc
- allocates drives in response to requests
tapevary - display status or vary
tapemt - " active & pending mount req.
tape - mount or unmount. Use -c
tm - positioning
tapecp - copy
tapecmp - compare

Cron

Cron handles unattended batch processing.

Crontab tables live in /usr/spool/cron/crontabs/logname. Be sure to clean out any from ex-employees. Carry this file forward to each new UTS release. The crontab cmd adds or replaces a user file.

cron.allow & cron.deny govern ability to use cron tables.

format: 6 fields

1. minute (0-59)
2. hour (0-23)
3. day of month (1-31)
4. mouth (1-12)
5. day of week (0-6)
6. shell command to execute

Splat (*) is valid as wildcard for any field.

Cron activity is recorded in /usr/lib/cron/log.

at - queues jobs for a one-time execution. at.allow & at.deny control access

batch - queues job for "later" execution when system is not "too busy". Job goes into queue B.

Jobs reside in /usr/spool/cron/atjobs/

Twenty-six queues are available (A-Z). Queue C is for crontab jobs. The characteristics are specified in /usr/lib/cron/queuedefs.

/usr/lib/cron/.proto.? holds environment of queued jobs. Prepended to every job run on the queue.

8-19-88

Dale Sansing

Tape Backup

Uses the public-domain Ingres relational database in /usr/ingres directory.

Files written in finc format and read with frec.

/usr/lib/backup directory holds administrative cmds + data files, logs, reports.

The db holds all info about the file - when, where, etc.

Admin Cmds:

backup - can run on mounted systems (even w/ users on). Updates the DB. Msgs are sent to userid "backup." This can be changed.

restore - queries DBMS, requests tapes + handles mounts, + restores via frec.

Full backup - whole filesystems.
Incremental - backs up only files updated since last full backup. Usually runs longer than full!

Customize /usr/lib/backup/back.full.sh for full backup script. Customize /usr/lib/backup/back.incr.sh to control incremental backups. This is where the default userid is changed.

Test out backup + restores (using frest) before you have a failure. Get hardcopy of the backup report + keep it.

Ingres even backs up its data base - probably /usr/lib/backup ... as its last action.

dsklist specifies the disks and file systems to be used plus the tape drives to use.

bkupcr - creates backup database - only run once.
bkupadm - add tapes to pool, delete, report, + alter retention.
bkuprpt - reports on full backup DB. Contains the info for manually recovering a file.

Also consider running volcopy as a supplimental backup method. These could be placed in the same scripts.

recover (1) - specify file to restore
Used by user to schedule
Schedule the backup scripts using cron.
Schedule /etc/restore several times per day
to do restores scheduled by users w/ recover cmd.

Installation:
- ; run bkuper to create DBMS.
- label (lm) tapes
- create file of tape names
- add tapes to bkupadm
  bkupadm -f /usr/lib/backup/ TEXTFILE
- designate disks & file systems to back up.
  specify /usr/ingres/data/base/backup last
- create filesystem containing list files
  /usr/lib/backup
- modify backup scripts - full + incr
- specify cron

Brian Jenks

Accounting

Data captured per-process:
- resource utilization
- connect time
- disk util → only physical I/O (not out of buffer)
- Bourne script writes to /usr/adm/fee for any special charges.

Acct data is written at the end of the process.

Tape mounts are not captured.

Files:

/usr/lib/acct/ - scripts
/etc/wtmp - temporary acct records
/usr/adm/pacct - reduced data files
dtmp
fee
sulog - who has been superuser
acct/nite/
sum/
fiscal/

Commands (scripts): - reformat from temporary areas & writes to stdout.
runacct - daily
monacct - monthly
acctwtmp - writes utmp(4) to stdout and can append records to /etc/wtmp
chargefee - script for recording special services

Many fields are low because the 5890 is so fast & the system was designed for much slower machines. Therefore, many tasks have Ø resource utilization. Amdahl is working to improve this.

ckpacct - will turn off acct if /usr/ gets too low on free blocks.
startup - starts acct daemon. This may be removed to turn off acct.
dodisk - performs disk acct & is controlled by /etc/checklist

turnacct - turns acct on/off. Saves and starts a new pacct file.
This is called by ckpacct.

Report cmds:
prdaily - daily rpt
- IPLs
- usage by user
- command summary
acctdusg - disk usage
acctcom - reports by process

/usr/src/cmd/acct/holidays contains the specification of prime time and non-prime times including a list of holidays which are non-prime.

Accuracy of acct is somewhat poor. Userids do not have an associated account number for project charging.

SAR

System Activity Reporting - this reports on system performance.
Cmds:
sadc - periodic data collector according to set intervals & number of samples.
Don't use too long an interval.
sa1 - shell script that executes sadc. Stores data in /usr/adm/sa/sa??
Invoked via cron.

sar reports realtime or formats a file recorded by sadc. Watch %usr and %sys (TVratio) for resource problems (thrashing).

sadp - disk file report
- cyl access
- seek distance

sag - sys activity graph of sar output.

Setup cron entries for reporting intervals.

Other Sys Activity Cmds

iostats - I/O requests
- % I/O by dev
- Average service time
- Reports on DASD, tubes, tape, & rio

timex - times a cmd while it executes

sfats - over system stats. Like a sar report.

Erep

UTS needs IBM Erep to format IPL, TSR, EOD, OBR, & MDR. UTS supplies scripts to download the IBM Erep (loaderep) and ossific to execute. Ossific handles SRCs (& is also used to run gen4705).

Data is collected by errdemon(1m) and reads from /dev/error & places them in /usr/spool/adm/errfile.

/etc/errdead will extract kernel error records from the LRB after a crash.
/etc/errstop - terminates daemon. Usually issued only by shutdown.

Checklists

Daily activities:
- operations log book
- console log
- check that all file systems are mounted
- daemons up (& no duplicates!)
- check motd
- backups
- acct
- file sys free space
- network up & ok

Weekly
- lost+found (via script)
- clean up /usr/news

fsck just found an inode & does not know the name! The name in lost+found is the inode #. Check the owner & have him put it back in a directory.

- run sweeps for setuid & large directories.

Monthly

- review perf
- reorg filesystems if req.
- change root passwd: when a person knowing the root passwd leaves, consider all passwords compromised.
- clean out core files (dumps are written to active directory).

UUCP

Each system in the net is defined to its neighbor & must have a map (i.e. usenet) A full path (of nodes) can be specified or a system takes a guess (maybe via map) or just ship it to the nearest backbone.

There is no standard protocol. Traffic is passed over whatever connects to a neighbor.

full-duplex
X.25
4635

Directories

/usr/spool/uucp - spooling area
/uucppublic - incoming

/usr/lib/uucp - conf files & commands

Restrict dial-ins.

Rm 312
x5378

Raj K. Gulati
9-12-88

UTS/F 1.2.2

Steve Swinkles - course adm (old) - Wed.
Brian Jencks - (new)

UTS/F supports full-duplex in FEP & is downloaded via superuser.

UTS/F is an extension of EP - adds macros.

UTS capabilities:
- Async - Full & half dup; 8 bit ASCII
- Bisync - chan att 3270 (3271, 3274)
  - Remote 3270
  - RJE (connect to JES via BSC line)

{ 3270 supported in full-screen or line mode.

- Ethernet (TCP/IP)
- Hyperchan

EP w/ UTS/F supports:
- full dup
- half dup
- ACU (801 type) - Auto Call Unit
- RSCS
- RJE
- 3270 (polled BSC)

Native Subchannel (NSC) - down loads EP
Packet " - talks to multiple lines.
↳ read & write

Protocols

full dup - dedicated r/w lines
half - same line
simplex - dedicated r/w but no response. All one way. - console prt.
echoplex - full-duplex w/o host echo. Terminal echos itself.

S/S (start/stop) - 10/8 or 11/8

1 start bit | 8 data bits | 1 stop bit

| 2 stop bits

BSC - blk check chars at end.

Full Dup (FDX)

Non-Canonical (raw) - char echo by
application pgm
vi
uucp

Canonical - echo by driver. Most popular.
May use I/O redirection. More dev. independent.

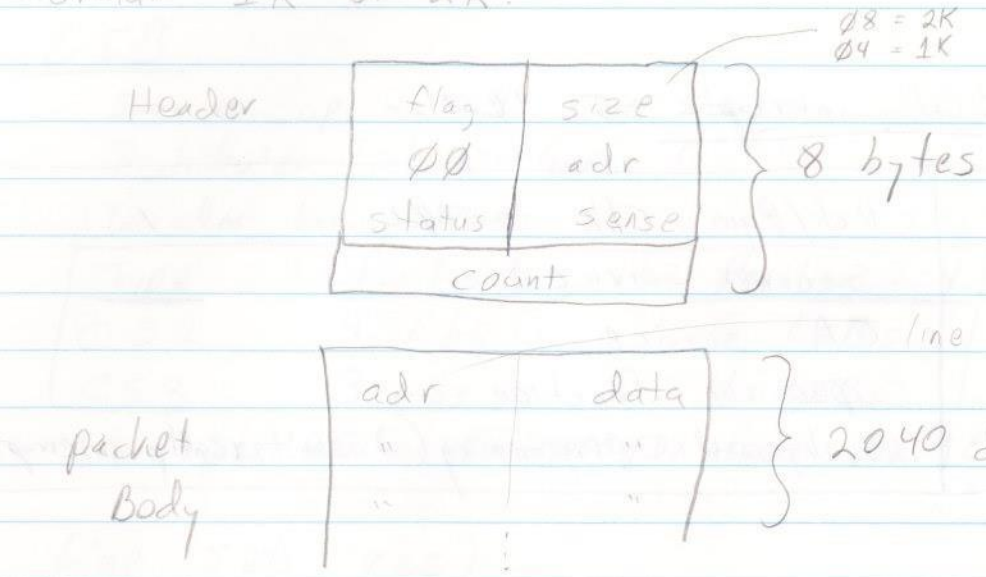
256 subchans into one 4705 chan adpt.
Max 2 CAs.

The 4705 w/ UTS/F buffers the character oriented devices and the host by building packets of incoming data and passing them to UTS in packets over packet subchannels.

Packet Operation

forward to host every 100ms if data ready. (per chan adaptor)

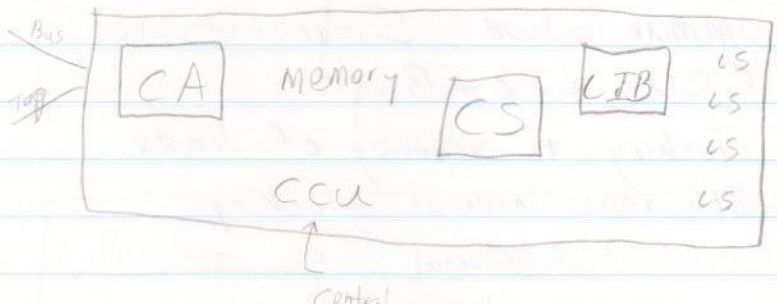
format 1K or 2K.



Dev attributes set by EP gen (termio options).

- Baud (cbaud)
- Char size (csize)
- local/dial-up (clocal)
- stop bits (cstopb)

4705 Architecture



write = even adr

read = odd adr

4705 will Run:

- EP
- PEP
- NCP
- UTS/F

Using Protocols:

- SDLC → not sup. by UTS/F
- X.25
- Bisync
- Async

CA - types 2, 3, 4
CS - " 2, 3

CCU interrupts - 5 levels

1	Mch/Pgm Chk + IPL
2	Scanner service
3	CA service
4	Supervisor function
5	Background programming (doesn't really interrupt

CA 4
EP + NCP
UTS/F supports max 2 (although 4705 can have 4)
Byte chan only for EP.
CA types 2/3
Not supported w/ UTS/F
used w/ NCP
Only native subchan is used.

CS - Communication Scanners
Connects CCU + CIB
Provides clocking + scanning of lines.
2400 BPS max internal clocking
64K " " external
↓
modem to modem synchronization

CS type 2 - max 4 in a 4705.
Async, BSC, + SDLC supported
64 adr in position 1 (of 4)
96 " " positions 2, 3, + 4

CS type 3 not supported. No S/S.

LIB

Services up to 4 line sets.
Distributes clock from CS
Decodes line addresses

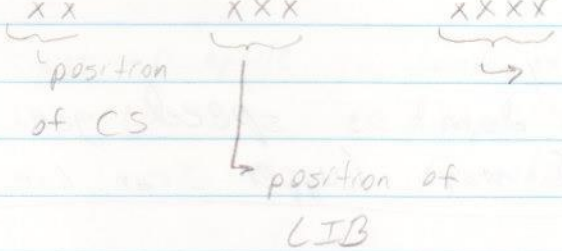
Type	Position 1	Positions 2-4
CS2	4 Libs	6 LIBs
CS3	3 Libs	4 LIBs

Two types: 2E for 4800+9600 FDX ASCII; and 1E for all others.

Line Sets (LS)

interface to modems
4 adr per LS
16 types of LS! - Half dup, FDX, NCI (autocall) . . .
One bit buffer!

Line address generation:



Position	Line Address CS2	CS3
1	020-05F	020-04F
2	0A0-0FF	0A0-0DF
3	120-17F	120-15F
4	1A0-1FF	1A0-1DF

4705 Configuration

Can have up to 512K of mainstore, but UTS/F needs only 256K.

CIB

Interface for h/s int-clocked async dev.
Multiplies 3rd + 4th scanner osc by 4. (in 2E)
Can mult 1 or both.
Max speed 9600 BPS
CSB Macro
Line speeds in CSB macro get used as labels + cannot be repeated.

2E - high speed, Business Machine clocking

HD1/FD1 - half+ Full-dup, low to medium speed.

NC1 - auto dial connection to device type 801 ACU.

Scan Modification

1.) w/o scan:
max ext = 4800 BPS
max int = 2400 BPS

2.) Upper Scan Limit - used by UTS/F
Reduces scan period for all lines on a CS.

3) Other - addr substitution for FEP.

Scan some lines more often by ignoring others completely. Limits the # of LGBs per CS.

External Clocking

adr goes down as speed goes up.
64K BPS max (Upper Scan Limit)

Internal -

9600 BPS max

An RPQ provides Extended Connectivity which increased # CIBs to 4. (from 3)

EDUSDØ3 PF1Ø

password - utsfoø dial EDUTSV3

login: usbiø3

utsf

/usr/lib/4705 - to process EP gen file.

do - add /usr/lib/4705 to PATH

PATH = $PATH:/usr/lib/4705

export PATH

or change .profile

check out /usr/local/games

gen4705 is a shell script. Pass this the name of the epfile.

net -l epfile - loads epfile to NSC.

-d - dumps 4705 to dumpfile. (crashes box)

Transmission techniques:

Async - clocking occurs in the DTE. Async devices will use internal clocking.

Sync - clocking occurs in the modems through an initial handshaking. This is used in BSC, HDLC, + SDLC.

/usr/lib/4705/*
/usr/staff/dacvkg/USBI/bx25.asm3705

EP Generation

9-13-88

EPgen builds the 4705 load module. Macros describe hardware options and line configuration.

Macros (EP)

Build - address, CA types, 3705 model, buffer definition . . .

CSB - logical position of scanner, type, speed, & wraparound test line. CSB will appear for each scanner.

Group - defines lines. Leased & switched must specified separately. Usually group by protocol. Protocol must match LS hardware - FD1, HD1 . . . Speed = labels

Duplex = Full simply keeps Clear-to-Send high at all times.

LINE - one coded for each physical line. Codes line adr, subchan adr, and chan adptor position. Also, clocking, speed, & duplex mode.

Generd - upper scan limit (usl) high speed select w/ CS3.

Macros (UTS)

LINES - list even (write) subchan for each FDX pair & associates with a packet subchannel.

/bin/prt
SBI DIST + export

UTS/F Installation

Seven files on a tape:

1 - stage 1 mac lib
2 - Object Lib (UTS/F)
3 - EP Object Lib
4 - JCL
5 - VM/CMS 3705 assembler
6 - " pgms + sample EP
7 - UTS/580 files + pgms

UTS/580 Considerations

Device drivers in /usr/src/uts/io linked into nuc during sysgen.

The file directory entry for a device (in /dev) contains in the inode field a major and minor number. The major number is used to locate the appropriate device driver. The minor number identifies the individual device within the major number.

The major + minor numbers can be displayed with ls -l or file /dev/... command.

The character device switch table contains addresses of entry points into the open, close, read, write, and ioctl routines of <u>each</u> device driver. The cdevsw structure is order by major number.

Device drivers generally consist of two parts: top half (or base level) handles open, close, read, write, and ioctl. The bottom half handles device specific interrupts.

To locate a device driver, get the major number from the inode in /dev. Use this to index into cdevsw and pick the appropriate pointer for open, close, ...

inittab

The inittab is used by the init pgm at system initialization (or run level change) time. The table associates each terminal with a process to execute.

/etc/getty is the process that opens the terminal line and displays the login prompt. Init processing invokes getty based on info in inittab.

The file /etc/gettydefs is consulted by getty to establish initial terminal settings. The user can change these settings after login with the stty command.

9/14/88

Steve Swinkles

steve @ uts. amdahl. com
amdahl! steven
x 6198

UTS + 4705 communications guru.

4705 Diagnostics

see:

3704/3705 Program Reference Handbook
GY30-3012-06

see the UTS man page for systrace.

LNVT - Line Vector Table
CHVT - Channel Vector Table w/in CHCB
CHCB - Channel Control Block
CCB - character (pointers)

CHCB - 1 per Channel Adaptor
x'68' lochan
hichan
+ pointers into line vector table by subchannel #.
(Subtract x'800' from entry ÷ 2 for line #)

CCB -
holds data buffers 0 + 1 (2 - 4byte buffers)
data service q ↑
CFLG - line type

Packet (UTS)
Header is not sent to host.

Emulator CCB listed for each subchannel (which appears at +C into control block).

Programming in C

9/19/88 Chapter 7, Arrays.

A linear array is declared simply by specifying the maximum number of elements:

```
int items[100];
```

Array elements are addressed beginning with 0. Thus, a 100 element array would contain items 0 - 99. The subscript may itself be an integer variable. C does no checking to see if the subscript is valid!

Array elements may be initialized during declaration:

```
static int items[5] = {2, 4, 6, 8, 10};
```

similarly, when an array is completely initialized (which is not required), the maximum subscript may be omitted.

```
static int stock[] = {1, 2, 3};
```

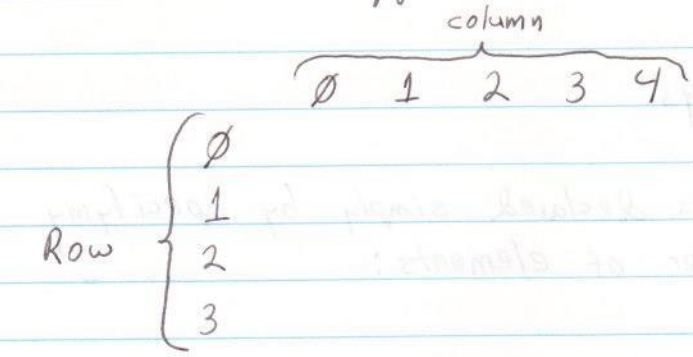
Two-Dimensional Arrays:

A double subscript denotes a two-dimensional array.

```
int M[4][5];
```

row — column

The matrix appears as



Note that both row and column begin with 0.

Matrices may be either partially or fully initialized:

```
static int M[4][5] = {
                        { 2, 4, 6, 8, 10 },
                        { 3, 6, 9, 12, 15 },
                        { 0, 1, 1, 2, 3 },
                        { 1, 3, 5, 7, 13 }
                      };
```

How does UTS handle an out of range subscript?

UTS will display an out-of-range subscripted element without protest. It will also store into and subsequently retrieve an out-of-range element. The program itself may be corrupted, but execution simply stops. No error condition is reported.

Functions - Chapter 8 9/20/88

Functions are identified by a pair of parentheses following the function name and the type declaration.

main ( ) - is a special function indicating where program execution begins.

The function name should be preceded by a type declaration indicating the type of value which is returned by the function.

Examples:

```
int    gcd (u, v)
float  square_root (x)
void   display_line (result)
```

The type void is a special declaration. It means that the function will not return any value to the calling routine. If no type is specified, the default is integer. Always specify a type.

The variables specified within the parentheses are the values passed to the function. These are called arguments, and are positional with respect to the values presented by the calling routine. The argument name is called the formal parameter name. Each argument must also have a type definition prior to the brace which begins the function. The type declared by the function must agree with the type passed. This is because the variable is simply copied from caller variable name to function variable name.

```
/*  Square root function  */
float  square_root (x)
int x;
{
    float y;
    code to do square root ...
    return (y);
}
main ()
{
    sqroot = square_root (5);
    printf (" The Square Root of 5 is %f\n", sqroot);
}
```

Passed by caller

formal parameter name

Automatic Local Variable

Used only by the function

calling the function.

Multiple variables can be passed to a function:

```
int mean (x, y, z)
int x, y, z
{
    :
}
```

The individual variable names used by the function are independent of the variable names used by the calling routine.

The formal parameter names identify only those values passed to the function. If variables are used within the function these are called automatic local variables and exist only for the life of function execution. Automatic local variables are defined after the opening brace of the function.

A function can only return one value!

```
return (x);
```

Whenever a return is executed within a function, control returns to the caller. Thus, multiple returns may be used to handle contingencies. For example, return (-1) may be used to signal abnormal completion.

An array ~~may~~ may be passed to a function. Simply pass the array name without any subscript. Of course, the function must be expecting an array.

The statement

```
minimum (scores)
```

← Array called scores in the caller

passes the array scores to the function minimum. The function must also type declare the array:

```
int minimum (values)
int values[];
```

← Array called values in the function.

Note that the array size is not specified in the function.

This points out a major difference in how functions handle arrays and individual variables. When an individual variable is passed, its value is copied from the caller to the function. The function cannot alter the value of the caller's variables.

When an array is passed, the function really receives a pointer to the original array! Thus, any changes made by the function effect the real array.

The size of the array need not be specified because the compiler is not actually dimensioning an array. If a size is specified, the compiler will ignore it.

Also, a function working on an array might appropriately be type declared as void since its output is preserved as alterations to the array.

Multidimensional arrays are yet different. The number of columns must be specified in the formal parameter name.

columns

```
int array_values[][75];
```

rows

Global Variables

A program's global variables are available to all functions. These are type declared first in a program, prior to any functions.

```
/* convert positive integer to another base */
/* Global Variables                         */
int converted_number[64];
long int number_to_convert;
int base;
int index = 0;
void get_number_and_base()
{
    :
```

Globals

function

Global variables somewhat limit the versatility of a function since it now references something outside of itself. It loses independence.

Automatic local variables (those referenced solely within a function) are reinitialized each time the function is called. If the value is altered during function execution, then that alteration disappears on the next function call.

However, the changes may be preserved by prefixing the type declaration of the local variable with the keyword static.

```
{
  static int times_called = 0;
          :
++ times_called;
}
```

Statics may also be useful for a variable that does not change in value. Function execution may be improved by avoiding reinitialization at each call.

Recursive Functions

Functions may be called from within functions. Indeed, a function may even call itself! The recursiveness must have a limiting value to avoid an loop. During each call the function gets its own set of formal parameters and local variables. In other words, it does

not know that it is being called recursively.

Structures - Chapter 9

Structure definitions, in effect, create a new type of variable. First, the structure itself must be defined:

```
struct date
{
    int month;
    int day;
    int year;
};
```

This info is used as a pattern by the compiler.

This example simply defines a structure called date which contains three elements. To use this structure it must be assigned to a variable:

```
struct date today;
```

← And this actually reserves storage.

```
struct date today, purchase_date;
```

To address a component of a structure separate nodes with a dot:

```
today.day = 21;
```

If a structure is defined outside of a function, then all functions can format particular variables according to that structure. Of course, the function must be expecting a variable of the given structure. All that needs to be passed to the function is the variable name.

Since only the format of the structure is defined globally the function works only on a copy of the data in the variable. Note that this is different from the way functions handle arrays.

To initialize a structure:

```
static struct date today = {7, 2, 1988};
```

Some compilers permit the static keyword to be omitted.

Arrays of structures as well as structures of structures may be defined.

```
struct time experiments[10];
```

defines an array of ten elements. Each element is in the format of a previously defined structure called time.

```
experiments[3].hour
```

accesses a particular component of an element.

Structures may contain structures:

```
struct date_and_time
{
    struct date sdate;
    struct time stime;
};
```

The date_and_time structure contains two previously defined structures.

Once a variable is defined to the structure

```
struct date_and_time event;
```

an individual component can be addressed:

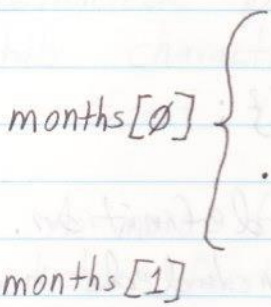
```
event.sdate.month = 10;
```

Structures may also contain arrays!

```
struct month
{
    int number_of_days;
    char name[3];
};
```

The second member of the structure is an array containing three elements.

Nesting can continue to absurdity:



These elements can be addressed as:

```
months[0].number_of_days
months[0].name[1]
```

Addressing procedes left to right from most inclusive to specific element.

Character Strings - Chapter 10

The data type char may store only a single character delimited by single quotes:

```
char greeting = 'H';
```

an array may be defined to hold a string:

```
static char greeting[] = {'H', 'I', '\0'};
```

The last character is the null character and defines the end of a character string in C.

Character arrays may be more straightforwardly defined using double quotes:

```
static char greeting[] = {"HI"};
```

This is equivalent to the previous definition. The null character is implicitely included in the character array.

Each element in the character array may be addressed individually. Alternatively, printf can handle the whole array by passing simply the variable name:

```
printf ("%s \n", greeting);
```

Printf will halt at the null character (which is why it is important). The array holding the character string can be greatly larger than the actual string. All that matter is the position of the null character. When concatenating string arrays, be sure not to embed null characters.

To compare strings, each element of each array must be individually compared.

Scanf may be used to input strings.

```
scanf ("%s%s%s", s1, s2, s3);
```

For this statement, scanf expects exactly

three items to be entered. Input is delimited per variable by a blank space, tab character, or end of line.
Scanf is not very flexible.

A more flexible alternative is to use the getchar function which is implemented in most installations.
Getchar returns one character at a time as it is entered (including spaces) which may be loaded into a character array. At the end of a line, getchar returns the newline character '\n'.

Another function, usually called gets reads an entire line from the terminal, up to, but not including, the newline. If this array is to be handled as a string, add the null character.

To use these functions, access their library at the beginning of the program with the statement

```
#include <stdio.h>

do
{
        character = getchar();
        buffer[i] = character;
        ++i;
}
while (character != '\n');
```

Don't forget to store \0 into the array in place of newline.

Escape Characters

\b	backspace
\f	formfeed
\n	newline
\r	carriage return
\t	horizontal tab
\v	vertical tab
\\	backslash
\"	double quote
\'	single quote
\ carriage return	line continuation
\nnn	ASCII equivalent

```
printf ("\"Hello\", he said.\n");
```

"Hello", he said.

The null character has an ASCII value of ∅. Thus, it may be used in looping:

```
while (string[count])
    ++count;
```

The value of string[count] will be ≠ ∅ until the newline is reached.

Value comparisons against characters are made on the bases of their ASCII numerical value (integer). Thus, the statement

```
c >= 97 && c <= 122
```

would determine if the character in variable c was between lower case a (ASCII 97) and lower case z (ASCII 122).

```
printf ("%d \n", 'a');
```

will display the integer ASCII representation of the literal lower case a (which is 97).

Other common functions:

- strlen - determine length of string
- strcmp - compare strings
- strcat - concatenate strings
- strcpy - copy string

9-21-88

Pointers - Chapter 11

```
int   count = 10;
int   *int_pointer;
int_pointer = &count;
```

← Defines an integer variable

← Defines a pointer to an integer variable

Assigns the address of variable count to the int_pointer

```
x = *int_pointer
```

→ This is the pointer in use:
$x = 10$.

& is the address operator.
* is the indirection operator.

The definition of the pointer itself is an ordinary integer definition (in this example) and could have been written

```
int count = 10, *int_pointer;
```

The type declaration of a pointer must match the type of variable it will point to.

```
char   c = 'Q';
char   *char_pointer = &c;
```

Pointers of the same type may be equated:

```
pointer1 = pointer2
```

assuming that both pointers had previously been defined.

Pointers may point to structures. The type of pointer must match the variable type.

```
struct date
{
    int month;
    int day;
    int year;
};
```

Defines structure type

```
struct date todays_date;
struct date *date_pointer;
date_pointer = &todays_date;
```

creates a structure

creates a pointer

Assigns a value to the pointer.

To use the pointer to access an element:

```
(*date_pointer).day = 21
```

Parentheses required

or

```
date_pointer->day = 21
```

Structures may contain pointers.

```
struct entry
{
    int value;
    struct entry *next;
};
```

Note self reference

Defines format

```
struct entry n1, n2, n3 . . . ;
n1.next = &n2;
```

Creates structures

Creates a linked list

:

Linked lists permit queues of sorted structures to be manipulated.

By convention, a linked list is terminated with a null pointer:

```
n3.next = (struct entry *) Ø ;
```

(struct entry * is the structure type name)

This is the formal way to declare a null pointer and the syntax may be used to detect one also:

```
while (list_pointer != (struct entry *) Ø )
```

Pointers and Functions:

When a pointer is passed to a function the function cannot alter the original pointer, but it can alter data using the pointer.

Pointers and Arrays:

An array pointer must be type defined as are the ~~the~~ elements of an array.

```
int values[1ØØ];
int *values_pointer;
values_pointer = values
```

Note that the & is omitted. An array name without a subscript is always treated as a pointer. An equivalent statement is:

```
values_pointer = &values[Ø]
```

Adding and subtracting from an array pointer has the effect of indexing to other array entries.

Thus, using the previous example,

```
*(values_pointer + 3)
```

would index to

```
values[3]
```

The ++ and -- operators are readily used when using array pointers.

Array pointers may be compared:

```
values_pointer > &values[99]
```

indicates if the pointer has exceeded the limit of the array.

Constant character strings are always handled as pointers:

```
text_pointer = "I want to go home.";
```

Thus, a constant character string may be handed to a function expecting a pointer

```
copy_string ("Invalid input.", string2);
```

Pre and Post Increments and Decrements

```
i = 0;
j = ++i;
```

→ Pre-increment

i is now equal to 1.

```
i = 0;
j = i++;
```

→ Post-increment

j is now equal to 0
i is now equal to 1

The same principle applies to the -- operator.

Pointers to Functions:

The type of the pointer must match the type of the value returned by the function.

```
int (*fn_pointer) ();
```

↑ Parentheses required

```
fn_pointer = lookup;
```

↑ Compiler knows this is a function.

```
entry = (*fn_pointer) (arg1, arg2);
```

is equivalent to

```
entry = lookup (arg1, arg2);
```

Pointer to arguments are particularly useful when passing a function as an argument to another function. This permits a sort of user exit for some functions.

Operations on Bits - Chapter 12

Bit Operators

&	AND
|	Inclusive OR
^	Exclusive OR
~	One Complement
<<	Left Shift
>>	Right Shift

Bit operations may be performed on short, long, and unsigned integer values, and on characters, but <u>not</u> on floating point variables.

<u>AND</u> - often used for masking.

```
x = x & 15;
```

or

```
x &= 15;
```

Inclusive OR

```
x |= 07;
```

Exclusive OR

```
x ^= y;
```

Exclusive ORing a variable with itself produces 0. This can also be used to exchange two values without using an intermediate:

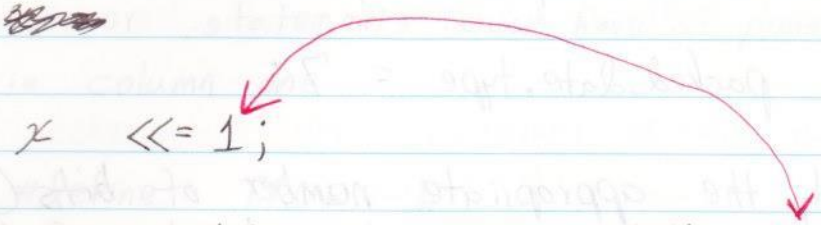
```
i1 ^= i2;
i2 ^= i1;
i1 ^= i2;
```

Ones Complement

This is a unary operator which simply flips each bit.

When bit operations are performed on variables of different lengths (long int and short int) the operands are aligned on the right.

Left Shift



```
x <<= 1;
```

results in the bits in x being shifted one bit to the left. Zero is shift in from the right.

Right shift

```
x >>= 1;
```

If the sign bit is Ø, then zeros will be shifted in from the left. If the value is negative and the sign bit is 1, either Ø or 1 will be shifted in from the left depending on the specific implementation and hardware.

Preserving the sign bit is an arithmetic right shift.
Abandoning the sign bit is a logical right shift.

Bit Fields

Individual bits and collections of bits within structures may be named.

```
struct packed_struct
{
    unsigned int f1:1;
    unsigned int type:4;
    unsigned int index:9;
};
```

field name

number of bits

These structure fields can be referenced in the same way as any structure field.

```
packed_date.type = 7;
```

Only the appropriate number of bits (low order) will be moved into the structure field.

The expression

```
if (packed_data.f1 )
   :
```

tests if the bit flag in true (1) or false (Ø).

Bit fields may be combined in a structure with other data types.

A bit field with a length of Ø can force the alignment of the next field on a word boundary.

The Preprocessor - Chapter 13 9-22-88

Preprocessor statements must have a pound sign # in column one.

```
#define  TRUE  1
```

#define functions like an equate. Literal substitution occurs before the program is compiled.

Everything to the right of the name is equated to the name. This is why preprocessor statements do not end with semicolon.
A #define may be continued to another line with the backslash character.

Once a #define occurs it will be used anywhere in the program, functions or main.

```
#define  NULL  Ø
#define  PI    3.141592654
```

By convention #define names are written in upper case. They may thus be readily distinguished from variable names.

Use #define whenever possible.

```
#define  MAXIMUM_DATA_VALUES  1ØØØ
```

will dimension an array and then serve in all tests of boundaries. To enlarge the array, one #define statement need only be changed.

Program readibility may also be greatly enhanced:

```
#define NULLPTR (struct entry *) Ø
```

could then be used in the statement:

```
if ( list_pointer == NULLPTR )
```

Anything may be equated in a #define, even C operators:

```
#define AND &&
#define OR ||
#define EQUALS ==
#define PI 3.141592654
```

a #define can even reference a previously defined name

```
#define TWO_PI 2.Ø * PI
```

Arguments may also be passed to #defines. These truely function as macros.

```
#define SQUARE(x) (x) * (x)

y = SQUARE (v + 1);
```

will evaluate as:

```
y = (v + 1) * (v + 1);
```

Note the use of parentheses. This is good programming practice.

Macros have an advantage over functions in that type declarations do not have to match. The substitution takes place in source language.

Header files containing definitions may (and usually do) exist in separate files which are included prior to compilation with the #include preprocessor statement.

```
#include "metric.h"
```

The quotes around the filename instruct the preprocessor to search the current directory first.

```
#include <stdio.h>
```

The <> marks instruct the preprocessor to avoid searching the current directory.

Many #define statements exist already and may be used by #include-ing the appropriate library.

Conditional Compilation -

Source statements may be conditionally compiled using conditional statements.

```
#ifdef  IBMPC
#       define  INT_SIZE  16
#else
#       define  INT_SIZE  32
#endif
```

Thus, in the preceding if IBMPC is 1, then INT_SIZE is set to 16. IBMPC may be set with

```
#define IBMPC 1
```

or

```
#define IBMPC
```

or

```
cc -D IBMPC pgm.c
```

Debugging statements may also be included conditionally:

```
#ifdef DEBUG
    printf ....
    for ....
#endif
```

Then just compile with the keyword DEBUG.

Further control may be exercised with #if and #elif.

```
#if MACHINE == 1
    :
#elif MACHINE == 2
    :
#elif MACHINE == 3
    :
#else
    :
#endif
```

A definition may be dynamically removed with the statement

```
#undef name
```

Data Types - Chapter 14

Specific variables may be defined as having only a limited permissible set of values. These are known as enumerated data types.

```
enum flag {true, false};
```

braces signify the definition of the type.

This defines only the data type flag and its permissible literal values.

no braces or parentheses.

```
enum flag end_of_data, match_found;
```

defines two variables of type flag.
Both statements may be combined:

```
enum {north, east, south, west} location;
```

In this example, the data type is not even named, but the variable location is of that type.

The values permitted must not be the same name as another variable.

typedef Statement

An alternate name may be assigned to a data type to improve program readibility.

```
typedef int COUNTER;
```

defines the name COUNTER as an integer which may then be used to define other variables.

```
COUNTER j, z;
```

This is equivalent to

```
int j, z;
```

Definitions may get elaborate:

```
typedef char STRING[81];
STRING text, input_line;
```

The variables text and input_line are now character arrays containing 81 elements.

```
typedef struct
        {
            int month;
            int day;
            int year;
        } DATE;

DATE birthdays[50];
```

Typedef names are uppercase by convention to indicate that they are not integral to C.

Working with Larger Programs - Chapter 15

Multiple files may be compiled together:

```
cc mod1.c mod2.c
```

Any errors discovered would be attributed to the appropriate source deck.

The source decks are compiled separately and then linked to form a single load module. When multiple decks are passed to cc, the resultant object decks are preserved with the .o suffix. A source deck may be combined with an existing object deck

```
cc mod1.c mod2.o
```

if changes had been introduced in mod1.c only.

Regardless of the number of source files, the program can have only one main() function.

Communications Between Modules

Unless specifically indicated, global variables (those defined outside of any function) are accessible from other modules. For example,

```
int move_number = 0;
```

would define a global variable in the owning module.

Other modules wishing to use this global variable would specify:

```
extern int move_number;
```

The external global variable may only receive an initial value in the owning program.

```
extern char text[];
```

references a global character array.

```
extern int matrix[][10];
```

references a global two-dimensional array of ten columns. Like formal parameters, the number of columns must be specified.

A variable may be globally defined for all functions within a module but not made available to other modules. Simply precede the definition with the keyword static.

```
static int move_number = 0;
```

Likewise, functions may be restricted within a module:

```
static int cube(x)
int x;
{
    :
}
```

Common data structures, external variables, typedefs definitions, and functions are excellent uses for #include. This enhances program readability and simplifies program maintenance.

Input and Output - Chapter 16

C language does not have inate I/O capability. All operations are performed via function calls. These functions are contained in the Standard I/O Library.

```
#include <stdio.h>
```

Standard I/O Functions

`putchar (c);` - place character on standard output.

`putchar ('\n');` - print a newline

`getchar ()` -

`printf %[flags][width][.prec][l]type`

- type → type of variable.
- l → lower case L, to display long integer
- .prec → precision
- width → Minimum size of field

Printf type conversion characters:

d	-	integers
u	-	unsigned integers
o	-	octal
x	-	hex using a-f
X	-	hex using A-F
f	-	floating point
e	-	exponential using e
E	-	" " E
g	-	floating point in f or e format
G	-	" " " f or E "
c	-	single character
s	-	null terminated character strings
%	-	percent sign

scanf - for reading a line of terminal input.

```
scanf ("%d:%d:%d", &hour, &min, &sec);
```

↑ Colons are expected in the input string as field separators.

```
scanf ("%d%%", &percentage);
```

↳ Indicates that a % sign is expected as input.

Scanf must have pointers to variables.

The end of file condition may be checked while reading files with the EOF flag.

```
while ( (c = getchar()) != EOF)
    putchar (c);
```

Files must be opened and closed. The fopen function returns a unique file pointer. This function is invoked with the file name and the mode - read, write, or append.

If the open fails, NULL is returned.

```
FILE *input_file;
input_file = fopen ("data", "r");

if (input_file == (FILE *) NULL)
    printf ("open error \n");
```

A maximum of 20 files may be open at one time.

```
fclose (input_file);
```
- closes file.

getc reads a single character from a file.

```
c = getc (input_file);
```

↑ should be type int

Getc returns EOF at end of file.

putc ('\n', output_file); - writes a newline to a file.

feof - tests for end of file.

```
if ( feof (input_file) )
printf of 4
    printf ("End of data.\n");
```

fprintf and fscanf perform printf and scanf operations on designated files.

```
fprintf (out_file, "Hi There.\n");

fscanf (in_file, "%f", &fv);
```

Read a floating point value into variable fv from file in_file

fgets and fputs are used to read and write entire lines.

```
fgets (buffer, n, file_pointer);
```

n - number of characters to store read data. Null is automatically appended after each line.

fgets will read n number of characters or until a newline is read, whichever occurs first. The newline does get placed in the buffer.

```
fputs (buffer, file_pointer);
```

fputs writes until it reaches a null character.

Program Termination - may be forced at any time:

```
exit (n);
```

where n is an integer return code or exit status.

Miscellaneous and Advanced Features - Chapter 17

```
goto out_of_data;
    :
out_of_data: printf ...
```

A label is simply a name followed by a colon.

A variable may be defined to hold different types of variables.

```
union mixed
{
    char c;
    float f;
    int i;
};

union mixed x;
```

The variable x now holds only a single value (unlike a structure) but that value may be either character, floating point, or integer.

Addressing is like structures:

```
x.c = 'K';
x.f = 3.14159265
x.i = 2
```

x can only hold one of these values at a time.

Registers

The compiler may be encouraged to place a frequently referenced variable into a register to improve performance. Simply prefix the register definition with the keyword register:

```
register int index;
register char *text_pointer;
```

It is not guaranteed that the variable will actually reside in a register.

Command Line Arguments

Variables may be passed to main at initiation of program execution. Two arguments are given to the main function if the routine is prepared to use them. These are a count of the arguments and a pointer to an array containing the input pointers. To utilize:

```
main (argc, argv)
int argc;
char *argv[];
```

The array contains elements of type pointer to character.

argv[0] is a pointer to the name of the program itself.

argv[1] is a pointer to the first operand.

Dynamic Memory Allocation

calloc and malloc reserve storage.

calloc needs the number of elements and the size of each element in bytes. It returns a pointer to the beginning of the area.

malloc needs only the total number of bytes. It does not zero storage.

sizeof (x) - returns an integer indicating the number of bytes in x.
x may be a variable or structure.

free (data_pointer); - does a freemain of the storage pointed to by data_pointer. This pointer must be set to the beginning of the area; it must be the pointer value returned by calloc or malloc.

AT+T BX.25 Network Architecture 9/27/88

Vince Rendenna (N2CLR)
AT+T Corporate Ed + Training
Piscataway, NJ
(201) 457-7194

Paul Baron, RAND 1964

Packet switching originated in the 1960's. The CCITT issued the X.25 standard in 1976. ISO issued an OSI 7-level structure of which X.25 is the first 3 levels. BX.25 adds some of these other layers above CCITT X.25. UTS like AT+T supports up to the session level. AT+T will conform to ISO OSI.

Telenet was the first public packet network. ARPANet was the first packet network. (1969)

DCE - the nodes
DTE - the customer: originates message and gives it to the network (DCE) which packetizes + routes.

Level 1 - physical level - transmits/receives binary
2 - controls modems
3 - controls packets
Level 3 FEP - recovery
Mainframe - Packet Driver: the bulk of the work.
4 - Session Driver (same as 3B)

A byte is called an octet to further separate it from IBM.

Packet sizes are 128 + 256 bytes. UTS also supports 512 byte packets. Messages are broken up to enhance reliability.

Level 3 divides the message into packets and creates headers with packet id, routing, and sequence of packet (to reassemble complete message).

The network only deals in packets. It doesn't know about messages. The level 3 software on the receiving mainframe does the reassembly. Then the message is given to the application.

X standards - public network protocol
U " - telephone network protocol

Next Unix + UTS releases will implement 7 layer ISO.

Terms

DXE - combines DTE + DCE ← cloud!
- Data Exchange Equip.
MAD - Msg Assem/Disassem (used by DTE)
PAD - originally async interface; now avail for anything
PSE - Packet Sw Exchange (4410) - a node
VC - Virt Circuit (user term, through network to host).
Two types - PVC - permanent (log chan # predefined + always used.

Dedicated to a specific host.
SVC - Switched Virt Circuit - user can connect anywhere in network.
LC - Logical Chan

Packet nets permits differing technologies to be connected. All conform to protocol; even SNA can come off DCE.

Level 1 implements V standards.

BX.25 will allow point to point. DTE to DTE. No DCE is absolutely required. Better than bisync for this. Multiple msgs before ack. All this + still X.25 compatable applications.

Packet is not great for bulk data transfer. Too much overhead for large amounts of data. But it is really great for rapid exchange of short messages.

X.1 Standard

Packet 2400 bps
4800 "
9600 "
48K "

This is official. Others exist in practice.

X.2

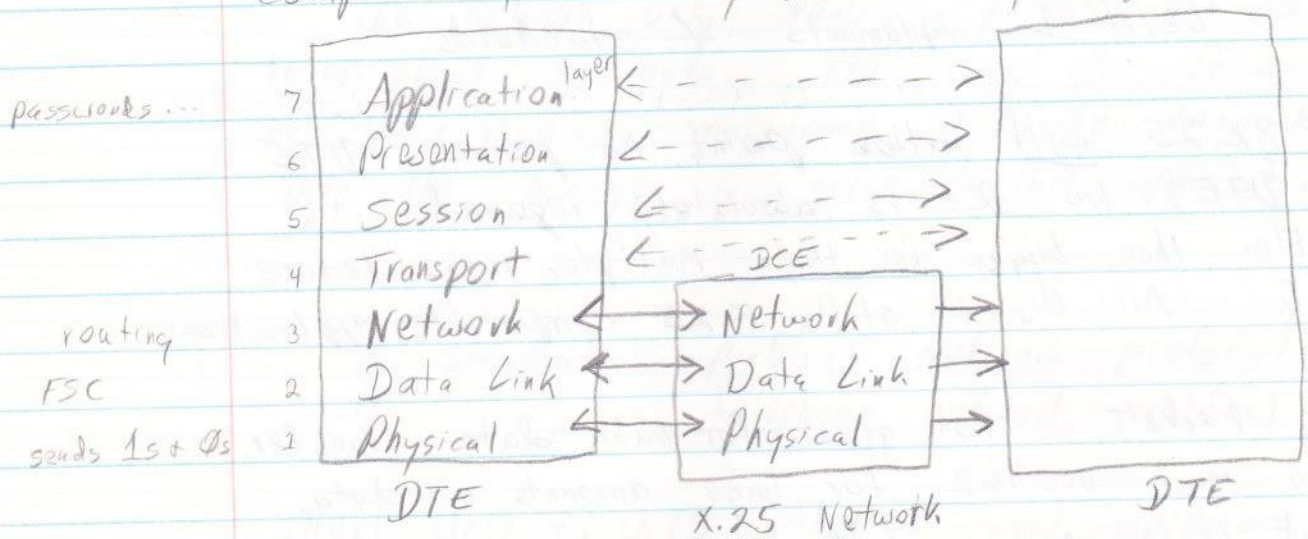
- SVC (or Virtual Call)
- PVC

CCITT is under the UN. ISO is the organization to which ANSI belongs.

AT+T's net is Accunet Packet service (APS) & came out in 1984.

Seven Layer Reference Model

Do things in the com software to relieve the application of the chore (like data compression, encoding, formatting ...).



Each layer adds a header which means something to the corresponding layer in the target DTE.

Layer 1 - provides bit streams
- activate/deactivate physical connections
- pin voltages & connector type

Layer 2 - monitor modem
- Provides flow control (buffering ...)
- UTS/F
- adds flags to denote beg/end of frame

Layer 3 - determines routing and switching
- provides billing info
Layer 4 - address end user machine
- Verifies end-to-end packet reception
5 - Session Level: Dialogue control between two applications.
- Provides expedited data exchange.
- Provides session sync + recovery (if one app. fails, halts data transfer while recovery takes places + then continues).
6 - Protocol conversion as req. (perhaps graphics...).
- Req. session estab. + termination
7 - interfaces with real application.

AT+T	X.25
Level 1 Physical	1 Phy
2 Data Link	2
3 Packet	3 network
4 Session	4 Session
5 AT+T File Transfer	6 + 7

AT+T does not currently implement the exact seven layer reference model.

Link-Layer Procedures

High Data Link Control (HDLC) is used by Level 2.

Window - the number of blocks which may be transmitted before ack. Various levels have different windows + these correspond to different increments. For level two the window is the frame.

ADR - address (each layer has its own)
CRC - Cyclic Redundancy Check-algorithm
FCS - Frame Check Seq.
LAPB - Link Access Procedure Balanced
RR - Rec. Ready
RNR - " Not Ready
REJ - reject
SABM - Set async Balanced Mode
Disc - disconnect
DM - " mode
UA - unnum. Ack
FRMR - frame rej

- Phy Layer Connectors -

RS-232-C -
- 50 ft max
- 20 K BPS
- 25 pin connector

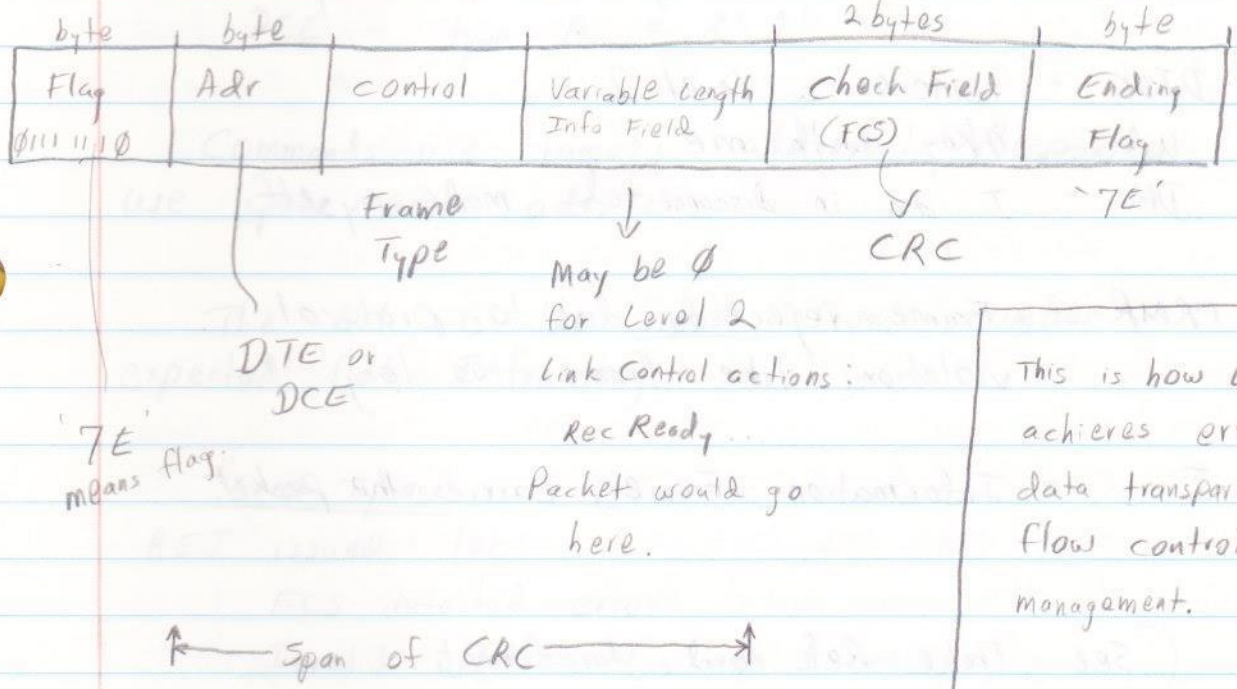
RS-449
- 200 Ft +
- 2 Meg BPS
- 37 pin connector

X.21 - standard for Digital connection to PDN.

X.21 bis - analog con. to PND. Eq to RS-232

Link Layer Frame (Level 2)

HDLC - 127 blks in a window. Will transmit any combination of bits. Allows both full & half duplex. 20 to 100 x faster than bisync.



This is how Level 2 achieves error detection, data transparency, and flow control (buffer management.

Zero Bit Insertion:

"Bit stuffing" is done to all info between flag bytes. If the bits contain five consequtive bit set on, it inserts a zero bit. This zero bit is removed on the receiving end by level 2.

CRC is created before bit stuffing and is applied on the receiving end after the stuffed bits are removed. As transmitted, only the flags will possess six consequtive 1 bits.

Contiol Flow - frame types

Supervisory Format:
- RR - ready to receive I-frames (into frame containing a packet)
- RNR - Rec. Not Ready - when buffer shortage occurs
- REJ - Reject: send I-frame over, data not valid.

SABM - handshake initiation
UA - ack to SABM
Must be exchanged before any I-frames.

DISC - removes a Level 2
UA - okay with me
DM - I am in disconnected mode myself

FRMR - Frame rejected due to protocol violation, like frame too long.

I - Information Frame; carries the packet.

See Frame Ref card for format.

An ack can be piggybacked with other frame types. When would a receiver ack I-frames:

1. Pole bit = 1
2. Window is full (K parameter - between 1 & 7)
3. Piggyback an ack on data that is ready to go back (this is full duplex).
4. Timeout (T2 timer expires)
5. REJ -

If you receive a frame with your own address specified, then bit 5 is a P bit. If not, then it is an F bit. The F bit can be on! F bit is final bit.

The pole bit forces an immediate ack. The P bit is normally set in the last I-frame that is less than the window amount.

SABM + DISC always set P bit.

S frames (RR, RNR, + REJ) normal have P = Ø unless status is required. The response would be an RR or RNR.

Address Field: two types:

DTE - type A = X'3'

DCE - type B = X'1'

Commands use target's address! Responses use your own adr.

The ack is the next frame number expected (of I-frames).

Error Types

REJ issued for

FCS detected error

Invalid address (is not 1 or 3)

Bad CTL field (NR + NS invalid) - like count goes backward. } software error in level 2

REJ returns bad control field + what was expected for VS + VR. Also, bits show type of detected protocol violation.

Not many FRMR frames nowadays.

Timers:

T1 - send timer. Expect ack (20-30 sec)

T2 - response delay - wait to see if more is coming. short

T3 - idle link disc (started when data set ready drops). or continuous carrier (15 consecutive 1s). Then initiate recovery.

T4 - BX.25 only. When T1 is not running, start T4 timer & upon exp. send supervisory frame - just to know link is up. Link assurance timer.

SABME - extended SABM. This is for satellite. NR & NS are two bytes.

Level 2 & 3 Communication

They pass registers:

- is link up or down (2 to 3)
- 2 passes adr, length & portid of input queue.
- 3 passes adr, length, & portid of output queue.

A link level window will contain multiple logical channels (each LC being a packet communication).

Level 3 Packet Layer

9/28/88

This level controls each logical channel. The packet layer is X.25 and is embedded in the HDLC frame of level 2.

The DTE selects the highest ready logical channel and issues a Call Req packet to the DCE. This is moved across the net to another DCE which selects the lowest ready log chan and passes the Incoming Call packet to its DTE. The reply is a Call Accepted packet which is received as a Call Connected packet.

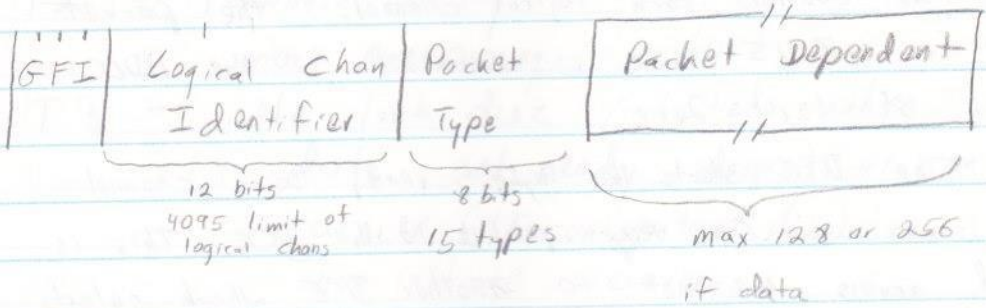
Call Req Packet:
- Calling/Called DTE addresses.
- Logical Channel Identifier (LCI).

The DCE keeps a table of the Log chan used on input + output. Likewise for the receiving DCE and DTE. Thus, each handler knows how to route the packet through itself. It does not know (or care) about entire routing. Once tables are built the DTE calling/called addresses are not needed.

Terms

GFI - General format Id
LCI - Log chan id
LC - Log chan
P(S) - Packet send Seq #
P(R) - Next pkt expected
M-Bit - More Data Bit
Q-Bit - Data qualified pkt (flag for Session Layer)
D-Bit - Delivery confirmation bit (like a return receipt requested).

Packet Header:



UTS supports 512 data packets. APS max is 256.

The M-bit will be on for all packets in a message except the last. Also, all but the last should have the maximum packet size. The last may be maximum size.

Packet Types:

Interrupt (x'23') - the only priority packet. Handled LIFO & can hold 32 bytes of data.

Rej (x'x9') is no longer valid.
If a packet is missing, now, reset the logical channel:

Reset Request ('1B') - retransmits all unack packets after reseting the logical channel. Counters are reset. So the missing packet arrives as Ø.

Restart ('FB') - usually used by the network to knock everybody off.

Diagnostic ('F1') - issued by the network containing specific codes to gather status.

Call Req ('0B') -
- calling / called adr length
- " " adresses
- Facilities Sections: (variable)
  - length
  - facility codes
- User Data (16 bytes)
  - the Fast Select offering permits 128 bytes.

Call Accepted ('0F) - permits no user data unless Fast Select is in use.

Logical Channel Numbers:
- 0 is reserved for restart & diagnostic pkts.
- PVCs are assigned next.
- Outgoing SVCs
- Incoming calls.

DTE Network Address: (IDN) International Data Number

DNIC = 3134
NTN = 212565 - 2574
SRSA = 212565
Area code + Node
EPN = 2574
EPN Customer

NTN = Network Terminal Number
SRSA = Serving Regron Serving Area
EPN = Ending Point Number

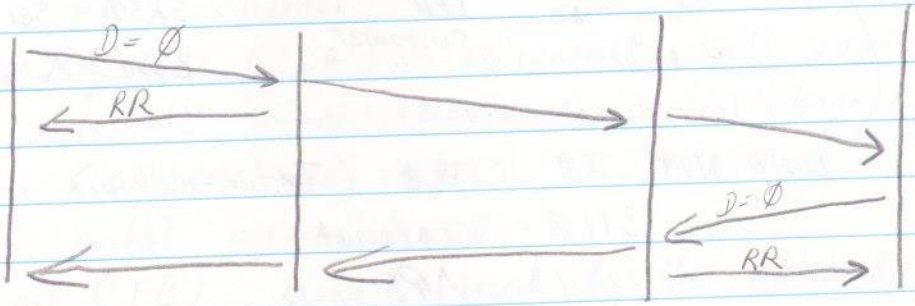
Data Net Id Code (Inter-network id)
- 3110 = Telenet
- 3134 = APS

Supervisory Packets: (3 bytes only)
- RR
- RNR
- REJ (not used in BX.25)

D-bit request ack from destination DTE (not the network). RR comes back.

Level 3 acks:
1. Window Full
2. Timer Expires (T22 timer)
3. Error Detected by Level 3.

Without the D bit on, only the local nodes (on each end) will RR to Level 3. With bit on the local nodes remain silent & the destination DTE returns RR. This slows throughput.



The D bit is set for an entire call.

Moving window - the packets considered in the window will "slide" up as packets are acknowledged. This is ideal. Otherwise, sender must wait at the end of a window.

Q-Bit: Control Data follows for Session Level.

Interrupt Packet - "Out of Band Signalling"
Processed LIFO. Can only send one at a time & must be responded to.
Holds up to 32 bytes of user data.
Interrupt Confirmation Packet is returned.
Holds no user data.

Restart Packet (uses log chan Ø) - Clears SVCs & resets PVCs. When all DTEs confirm clear/reset, the DCE responds with Restart Confirmation (will respond after 60 seconds anyway)
Restart is 5 bytes. Holds cause code and diagnostic code:

ØØ - local procedure error
Ø3 - net congestion
Ø7 - net operational

See green card for diagnostic codes.

Reset Request - logical channel stays up. Counters set to Ø. Unack pkts are resent. 5 Bytes with cause and diagnostic codes.

Clear Virtual Circuit - might be returned in reply to Call Request which fails, or a Call Request that holds 16 bytes of user data. If this is all the data, why bring the line up. Return clear with a code saying data received.

Optional User Facilities (on Call Req.)

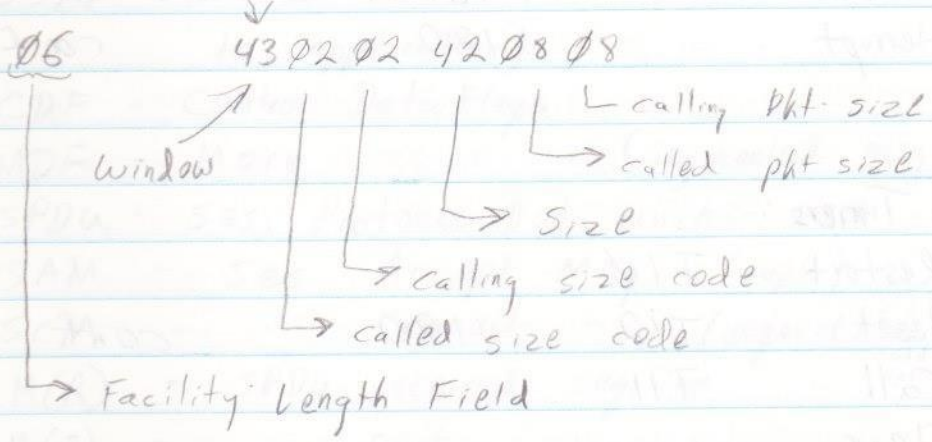
	Code	
Reverse Charging	Ø1	
Fast Select	Ø1	128 bytes on Call Req + Clear
Closed User Group	C3	- only certain DTE can talk to each other. Private net inside a net
Flow Control		- packetsize
One-Way Lcg Chan		
Packet Retransmission		permits REJ
D-bit Modification		- sets D-bit on all data packets
Call Barring		- network security bars select adr.

Throughput Class

Fast select may be used w/ credit card validation. 128 bytes of user data on Call Req + a response of clear w/ 128 bytes. Then Clear confirm tidies up. Fast + clean for extremely short transactions.

The Fast select <u>can</u> respond with a call Accepted if more data needs to be transferred (as determined by the application). No data on call accept.

Window + Packet size can be negotiated. Not normally done. First byte identifies window. Facility length ID:



Call Accepted Packet echos the specified facilities.

Throughput-Class - a logical channel may be "slowed" down. If a non-buffered printer at 1200 BPS is connected to a 56 KB line, then a throughput class can be specified to slow the bits.
Not normally done.

If Packet Retransmission is specified, it is only used DTE to DCE, never DCE to DCE.

Diagnostic Packet -

DCE Timers:

Reset	T12	180 secs	confirmation
Call	T11	200	call conn or clear
Clear	T13	180	conf
Data		150	
Rej		60	
Super Pkt		60	
interrupt		180	conf

DTE Timers:

Restart	T10		
Reset	T12	180	conf
Call	T11		
Clear			
Data			
Rej			
Supar			
Int			

Packet Layer X.25 / BX.25 Differences:

- BX.25 has additional diag codes
- " permits DTE to DTE
- " higher levels defined
- " Diag code required
- " Int req. session layer
- " Rej not used

Layer 4 - Session Layer 9-29-88

SSAP - Ses. Ser. Access Point - Adr of Session (link to application)
CDF - Control Data Flags
MDF - More " " (segmented msg)
SPDu - Ses. Protocol Data Unit - user data
SAM - Ses Accept Msg (reply)
SCM - " Connect " (request) - Origin type - Type - one way ... - code set - options
M(R) - SPDu received seq. #
M(S) - " sent "

Session layer binds applications to X.25. X.25 doesn't cover session layer. This is BX.25 convention. Uses three data bases:

Local - session addresses
Remote - " "
DTE access path -

Session layer optional w/ PVCs.

Session addresses are used to associate with an application. These addresses are unique within each DTE. Communicating sessions must know each others session address.

Session layers can work w/o a network (DTE to DTE).

Sessions may use Fast Select. The SCM is on the Call Request packet. The SAM comes back in the Call Accepted.

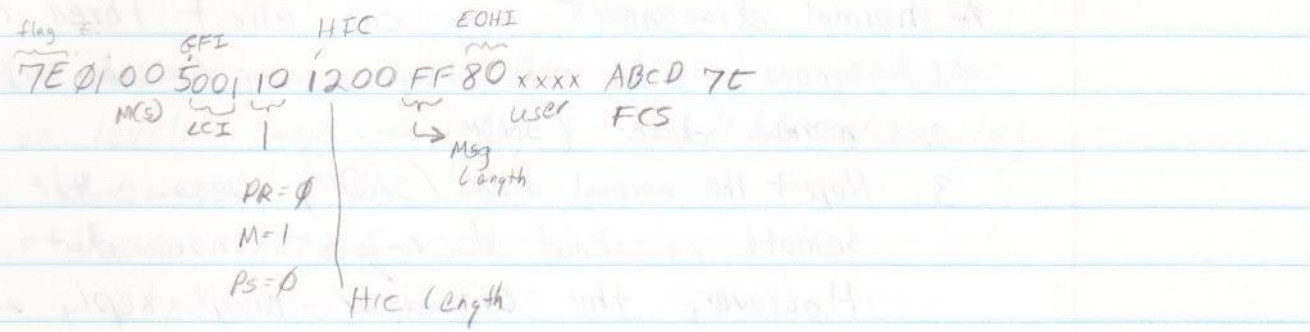
SNDM initiates disconnect & the remote replies with an SNDM. Then resources are cleared & logical channels are freed.

Session Layer Header - variable

Item Code - 4 bits (HIC - Heading Item code)
- 0010 - Control Msg (session conn, accept
- 0011 - Parameter - origin type . . .
- 0100 - Control Data for a higher level
- 0101 - User Data for higher level
- 0001 - Data
- 1100 - User Calling & Called Session Adr.

Parm Length - 4 bits
Parameters - variable
End - X'80'

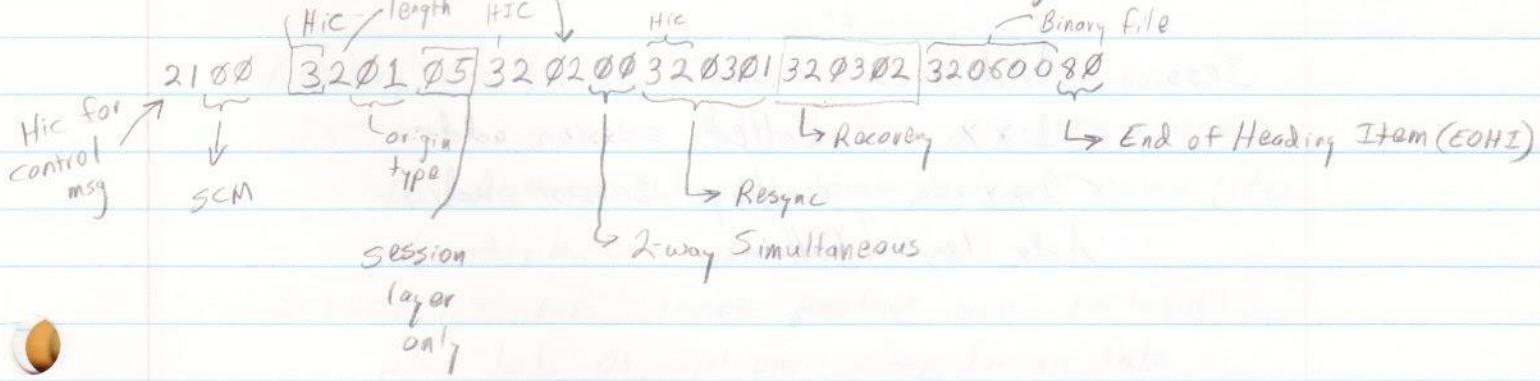
The Parm Length (or Heading Length) will be either 2 or 3, indicating 2 or 3 bytes of M(S) plus 1 or 2 bytes of message length. Two bytes of length will handle a 64K msg length value. A single byte can describe up to 255 bytes.



Session Parms: (on Ref Card)

Origin Type - who's the highest level?
- Session
- Higher
- AFT - AT&T File Transfer

Dialogue Type
- 2 way simult
- 2 way alt
- 1 way outgoing
- 1 way incoming
- Restricted - card image



Reject / Disconnect Control Msg. (SRM + used in response to a SCM with something wrong in it. Disc happens after SCM. Both have reason codes (which appear in the daemon log).

Session Estab & Termination:

A normal disconnect msg can elicit three responses:

1. ignore - other end keep sending (+ then SNDM)
2. normal disc (SNDM)
3. Reject the normal disc (SRDM) because the remote is about to request more data. However, the originator may reply with abort disc (SADM) which rejects the reject and the connection is gone.

Session Recovery - if allowed, will reconnect (SRECM) and reaccept (SREAM) after line failure.

Session Resync - requests resync (SRESM) by msg # + if ok ack in (SRESAM). This resyncs on a msg number. checkpointed

Session Addresses:

C1xx - called session addr
C2xx - calling session adr.
Adr length follows.

A session layer may not exist, as on a pad. In this case the originating session must know that no session level exists on the other side. What's left is essentially Level 3 to Level 3.

Unix and the Session Layer

A daemon interfaces the application and the session level. Keeps a log: /usr/lib/sess/dmn_log
No doc avail on log.
Daemon has 6 cmds
- bx25local
- " remote
- " dte

} database related

- " gss - prints a db
- " link - control Levels 2 & 3; & status
- " chan - allocates PVCs.

The daemon consists of 6 processes

The resync option causes the Level 3 D-bit to be set on.

Library Calls: Placed in C pgm
SATTACH - session attach to id user process to daemon. Wants local partner name (to get attributes from db).
SCONNECT - pass remote partner name (& local), plus lots of options: sleep for no data, resync/recovery, quality.
Session identifier comes back

SSEND - scid, flag, adr of application buf
SXSEND - 32 bytes of user data
SSELECT - will get back a list of all sessions which have data. Used to handle multiple sessions (maximum of 16).
SRECV - check rec buf
SDISC - session normal disc
SABORT - abort session at any time
SDETACH - free resources

Establishment

1. Start Daemon - bx25daemon 5 &
2. Create session Parties Databases (one time only)
3. Attach a Link (svc) - bx25link
4. Assign DTE Addresses
5. Install Channels for PVCs only
6. Init Link -i (SABM goes out) Can say that one end is acting like a DCE.

9/30/88

Commands (Network Administrators Guide)

BX25chan -
install, remove, status, + extended status on PVCs
Specifies window size, packet size, + link number.
-e + -s status options are good for debugging.

BX25link - att, detach, activate, deactivate, + get status for links.

bx25link -a -m/dev/tn75.0 -t -p0 -s12 -l0

- -a: activate
- -m/dev/tn75.0: device
- -t: DTE
- -p0: High PVC
- -s12: High SVC
- -l0: Link #

b x 25 link -l -10 ← activate. send SABM.
-h -10 ← halt (Level 2 DISC)
-d -10 ← detach

-n → change packet size & window size
-t → put in DTE addresses if connected to a network. (Telenet & APS)
May have multiple -t commands.

On a DTE to DTE connection, the b x 25 link activate command must specify one side as DTE (A) and one as DCE (B). The attach must be similarly modified.

DB Manipulation:
b x 25 gss - interrogates remote db.

Local Partner Database Entry -
local partner name (need for sattach from C pgm)
pathname to the application pgm (needed for incoming calls - daemon will start it running).
class of connection - Session level or Level 3 direct.
type of partner - } ① unique - only one session at a time.
connection control - } ② exclusive - target will duplicate itself and permit a second session.
③ shared - multiple sessions may be connected to the same process (max of 16).
Public/Private - controls access to your database (to b x 25 gss query).
SPDU - buffer size; & PSDU size for level 3 direct.
Reverse charging acceptance flag -
Closed User Group ids -

Outgoing/Incoming bps throughput class. Default is line speed.

My remote entry must match a local entry at the target machine.

The DTE info in the remote entry must match an existing DTE entry. Holds:

- SVC link #
- calling/called addresses
- fast select flag
- reverse charge request flag
- network Y/N
- closed user group
- in/out rates (in bps)
- phone numbers for autocall

bx25local -p ← prompt -a -d -m

remote " ↓ ↓ ↓

DTE " add delete modify

daemon log

local

ALOC.TXT Thursday, August 25, 1988 Page 1

```
cat ALOC
filename: /dev/tty
ses_local name:local  ses_remote name: remote
local process started
local process ATTACHED to daemon
fts_conn: usrdata: /dev/tty
local process trying to establish a connection with remote process
sconnect returns 0, hndlr_rsp = 0
local process waiting for S_CONN_ACCEPT from remote process
local indication handler invoked
res code 0
S_CONN_ACCEPT received from remote
local indication handler scid: 0 uscid 0
sconnect returns 0, hndlr_rsp = 1
fts_connct: CONNECTED
 Local name: local Remote name: remote
 Entering fread
 Entering fts_write
local process trying to send 54 bytes to remote
local Ssend call was successful
local process is done sending data to remote process
local process is going to detach itself from the bx25 daemon
scid: 0 udata:  connect accepted
 length: 120
sesname: local
local process is sending S_DISC to remote
local process is waiting for S_DISC_ACCEPT from remote
local indication handler invoked
res code 112
S_DISC_ACCEPT received from remote
reason code 112
local indication handler scid: 0 uscid 0
 loop: handler_rsp :11
handler_rsp 11  rc: 0
fts_disc: DISCONNECTED
 session completed
$
```

remote

local name is "remote"

```
$ cat ARMT
remote process started
fts_accept: Local Name: remote ATTACHED
start the local process
remote waiting for S_CONN from local
remote indication handler invoked
res code 0
S_CONN received from local process
remote process sending S_CONN_ACCEPT to local process
remote indication handler scid: 0 uscid 0
fts_accept: Session Established
remote process trying to receive data from local process
remote received Total_bytes: 54
remote indication handler invoked
res code 112
S_DISC received from local process
remote process sending S_DISC_ACCEPT to local process
reason code 112
remote indication handler scid: 0 uscid 0
remote received Total_bytes: 0
remote process is done receiving data from local process
remote process is waiting for a S_DISC from local process
remote process is going to detach itself from the bx25 daemon
remote process terminated
$
```

TABLE OF CONTENTS

Troubleshooting at the 3B

OVERVIEW

The AT&T X.25 Network Interface Software provides commands for setting up and managing the X.25 databases, channels and links. This unit describes the commands, their usage, and possible error messages. In addition, an attempt to provide a troubleshooting scenario for problem determination has been included.

OBJECTIVES

On completion of this lesson, you should be able to:

- Identify various teleprocessing problems specific to a 3B in packet switching.
- Explain the steps used in testing and troubleshooting an X.25 communications link.
- Determine link and channel status using a datascope and the UNIX BX.25 status commands.

The next step is to use the *bx25link −s −1 2* command to displ status of link 2:

```
/etc/bx25link −s −1 2
```

The status output is as follows:

```
linkno: 2 empbufsz: 256 state: HALTED l3role: DTE
nspkt: 128 nswind: 2 hipvc: 30 hisvc: 125 lcalloc: 125
lev2q: NOT FULL RST_on_l2q: NO hiq: EMPTY loq: EMPTY
mdev: 1 type: 1 csidev: 1 lev2_sync: NO nlev2_fails: 0
stfail: NO errcode: 0 l2_xmtq_len: 0 l2_emptq_len: 0
```

Link and Channel Administration

The following examples show how to start level 2 and level 3 protocols, how to install PVC channels, and how to obtain the status of the links and channels associated with both PVCs and SVCs.

The first example illustrates the use of the *bx25link* and *bx25chan* commands to set up a DTE-to-DTE connection using one TN75 and one TN82. Since this is a DTE-to-DTE connection, a decision has to be made between end users as to which host is to be the DTE and which host is to be the DCE. It is also important that one side of the connection uses link level address A and the other side uses address B (the address B is selected by specifying *-b* in the *bx25link*

−i command), otherwise, both sides will not be synchronized at level 2 and, as a consequence, level 3 will be unable to synchronize.

Example of TN82 (/dev/tn82.1) associated with link 2.

The first step is to attach the X.25 link to the desired TN82 device port:

/etc/bx25link−a −m /dev/tn82.1 −t −p 30 −s 125 −1 2

In this example, TN82 port 1 (/dev/tn82.1) is attached to link 2 (Remember that the unit and port number are the same for the TN82). This command associates the device tn82.1 with link 2. The **−t** option specifies that this link interface is the DTE. The **−p** and **−s** options specify that logical channels 1 to 30 are to be assigned to PVCs only, and logical channels 31 to 125 are to be dynamically assigned to SVCs. The −1 2 option specifies that the link number associated with tn82.1 is link 2.

The next step is to use the *bx25link −s −1 2* command to display the status of link 2:

/etc/bx25link −s −1 2

The status output is as follows:

```
linkno: 2 empbufsz: 256 state: HALTED 13role: DTE
nspkt: 128 nswind: 2 hipvc: 30 hisvc: 125 lcalloc: 125
lev2q: NOT FULL RST_on_12q: NO hiq: EMPTY loq: EMPTY
mdev: 1 type: 1 csidev: 1 lev2_sync: NO nlev2_fails: 0
stfail: NO errcode: 0 l2_xmtq_len: 0 l2_emptq_len: 0
```

The fields and their meaning are as follows:

linkno − This is the link number specified by the *−1* option of the *bx25link −a* command, in this case, 2. The link number values can range from 0 to *bx25links −1*, where *bx25links* is defined in the system description file. This file is by default **/etc/system** for the 3B family (it could also be user−defined).

empbufsz – empty receive buffer size. This buffer should be as large as the largest packet size used for a logical channel on this link. Its value can be either 256 or 512 octets. In the example, it is 128, which is the default value.

state – state of the level 3 interface of this link. The interface can be in one of four states:

1. HALTED when the link has not been activated (by the *bx25link* *–i* command) or the link has already been halted (by the *bx25link* *–h* command).
2. PROC RST when a restart indication is being processed.
3. WAIT RSTCONF when the link is waiting for restart confirmation.
4. RST COMPLETE when level 3 is in the flow control ready condition. In order for the level 3 state to change from PROC RST to RST COMPLETE, level 2 must be synchronized, since a RESTART REQUEST is transferred making use of the services of level 2.

l3role – the level 3 role; either DTE or DCE. In this case, DCE.

nspkt – the non-standard default level 3 packet size for this link. In this case, 128 octets.

nswind – non-standard default level 3 window size. In this case, 2.

hipvc – highest logical channel number associated with PVCs. The logical channel numbers to be assigned to the PVCs range from 1 to *hipvc*. In this case, from 1 to 30.

hisvc – highest logical channel number assigned to SVC. The logical channel numbers for SVCs range from hipvc +1

to *hisvc*. In this case, from 31 to 125.

lcalloc – logical channel number from where the system starts searching for available logical channel numbers. The DTE side is assigned SVC numbers from the high end of the range (i.e., *hisvc*), and the DCE side is assigned SVC numbers starting from the low end of the range (i.e., hipvc +1).

For example, *bx25link –a –m /dev/tn82.1 –t –p 30 –s 125 –1 2* specifies that this link is a DTE; when *bx25link –s –1 2* command is executed, then the value printed for *lcalloc* will be 125, since DTE starts search from the high end (*hisvc*).

lev2q – indicates the status fo the level 2 queue associated with this link, whether or not it is full.

RST_on_12q – indicates whether or not restart packet is on the level 2 queue. In this case, the answer is NO.

hiq – indicates whether or not the high priority queue is empty. The high priority queue is used by the control packets such as call request/call accept, clear/clear confirmation, reset/reset confirmation, interrupt/interrupt confirmation, reject, and receive not ready packets.

loq – indicates whether the low priority queue is empty.

mdev – indicates the minor device of the tn75 or tn82 device used.

type – indicates whether the device is a TN75 (type: 0) or a TN82 (type: 1). In this case, the value is 1 for a TN82 device type.

csidev – the CSI channel number to be used with the *vpmsave* command to trace level 2 and level 3 events.

lev2_sync – indicates whether or not level 2 is synchronized.

nlev2_fails – number of times the N2 counter was exceeded. The N2 counter is user selectable on the 3B20. Its value range from 0 to 7. The default value is 7.

stfail – indicates whether or not the start and stop command for level 2 failed. If NO, the start/stop did not fail; if YES, the start/stop failed.

errcode – termination code returned the last time the level 2 script was terminated. See *vpm*(7) of the *UNIX System V Administrator's Reference Manual* for interpretation of termination codes. Termination codes are summarized

Normal termination code is zero, when the link is halted by using the *bx25link -h* command.

12_xmtq_len – level 2 transmit queue length.

12_emptq_len – level 2 empty queue length.

Summary of Termination Codes

Code	Meaning
0	Normal termination. CD received halt command from driver.
1	Not used.
2	Not used.
3	Not used.
4	Jump address not even. Internal error, call customer support.
5	Not used.
6	Not used.
7	Not used.
8	Not used.
9	Not used.
10	The CD detected loss of the modem ready signal at the modem interface (DSR). Check that the modem is not disconnected, power is on, and modem tests pass.
11	Transmit buffer sequence number error. Internal error.
12	Command error. An invalid command or an improper sequence of commands was received from the driver. This is an internal error, call customer support.

Code	Meaning
13	Not used.
14	Invalid transmit state. Internal error, call customer support.
15	Invalid receive state. Internal error, call customer support.
16	Not used.
17	Not used.
18	Not used.
19	Same as code 6.
20	Same as code 7.
21	Not used.
22	Used for debugging CD. If it happens, call customer support.
23	The driver's OK-check has timed out.
24	Not used.
25	CD driver unable to accept command. Internal system error.
26	The CD OK-check has timed out. This is a system internal error.
27	No such line number on CD. The line number specified is not defined in the system.

Print the status of the X.25 logical channel 2 on li. associated with a PVC. (This command also works for SVCs):

/etc/bx25chan −s −c 2 −1 2

Must know svc # for svcs in progress.

The output from */etc/bx25chan −s −c 2 −1 2* command is follows:

```
chno: 2 linkno: 2 pktsz: 128 windowsz: 2 connid: 0
state: HALTED sending: NO rcving: NO xmtqhi: NO
xmtq_len: 0 recvq_len: 0 dgnsend: 0 dgnrecv: 0 causerecv: 0
```

diag code

cause code

3B20s once ran two versions of X.25

1. DETERMINE VERSION OF X.25

The first step that needs to be determined is what communications software package a given application is actually implementing (i.e. X.25, BX.25 Session Layer, etc.). This information is essential to investigate the problem. In all of these cases it is assumed that correct shells exist to activate the link, deactivate the link, and check the status of the link in trouble. The names of these shells should be specified in the application documentation. These shells must be syntactically correct and also must match the current configuration being analyzed.

a) Old X.25 can be classified by the use of the following UNIX commands in the link shells:

   1. /etc/x25pvc

   2. /etc/x25lnk

   3. From a programming perspective the following system calls would be used for communications:

      - open, read, write, ioctl

b) BX.25 can be classified by the use of the following UNIX commands in the link shells:

   1. /etc/bx25chan - used only when PVC's (permanent virtual circuits) are being used

   2. /etc/bx25link

   3. Within UNIX there exists bx25 daemon processes and a set of library routines that an application would call for communications activity.

2. DETERMINE LINK STATUS

In all cases we must determine if the link is active. Execute the shell to check the status of the link. Do we get the desired result?

yes - this part of the analysis is complete
no - run the shell to bring the link down
run the shell to bring the link up
wait for up to 3 minutes
run the shell to check the status of the link
do we get the desired result?
yes - this part of the analysis is complete
no - is the hardware involved in service? (tn75, tn82)
execute disp
execute doff and don to the appropriate hardware.

Next hook up the data scope to isolate and analyze the problem and to determine why level 2 is still not working.
It is assumed from a hardware standpoint that the scope is attached correctly and that it is functioning properly. Set the data scope to monitor the FRAME level (link layer).

SAMPLE DISPLAY OF SCOPE WHEN MONITORING THE FRAME LEVEL

Below is sample display of what should be seen on the data scope after executing the shell to activate a specified link.

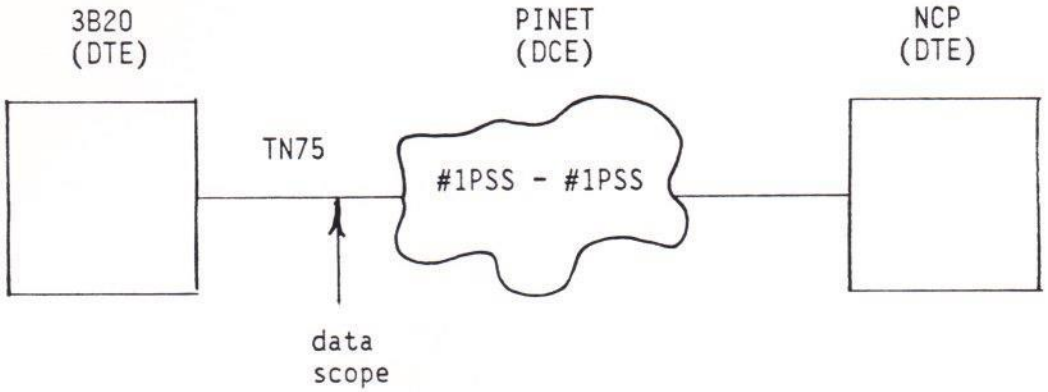
REPORT DISPLAY WITH REP_SHORT

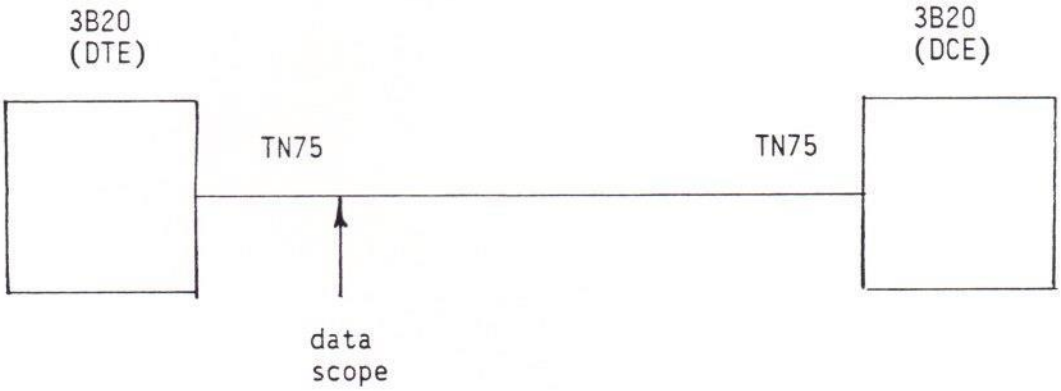
```
DCE  01  SABM

DTE  01  UA
DTE  03  I   0   RESTART INDICATION
DCE  03  RR
DCE  01  I   0   RESTART CONFIRM
DTE  01  RR
DTE  03  I   1   INCOMING CALL
DCE  03  RR
DCE  01  I   1   CALL ACCEPTED
DTE  01  RR
DTE  03  I   1   DATA 256   IDACOM ELE
DCE  03  RR
DCE  01  I   1   RR
DTE  01  RR
DTE  03  I   1   RESET INDICATION
DCE  03  RR
DCE  01  I   1   RESET CONFIRM
DTE  01  RR
DTE  03  I   1   CLEAR INDICATION
DCE  03  RR
DCE  01  I   1   CLEAR CONFIRM
DTE  01  RR
DTE  03  DISC
DCE  03  UA
```

HARDWARE CONFIGURATIONS

Below are some sample environments





Remote entities in the above environments may also be able to monitor communications at their locations.

3. ANALYZE LINE PROBLEM

Execute the shell to bring the link down and then execute the shell to bring the link up. On the scope we are concerned with the FRAME TYPE field and its associated entries. The scope we should be monitoring activity on both the DTE and the DCE sides of the link.

1. If nothing on the DTE side of the scope is seen, it means there is a problem between where the scope is hooked up and the 3B20 inclusive.

   Possible problems include:

   - bad TN board
   - something not properly cabled/bad or wrong cable, e.g. not RS232C.
   - TN board not in service - disp TN?? to see if the SADL board is in service

   Possible solutions include:

   - doff and don the TN board involved
   - run diagnostics on the TN board
   - replace the TN board

   or analog loopback to clear board

2. If nothing on the DCE side of the scope is seen, it means there is a problem between where the scope is hooked up and the remote side (PINET or the other 3B20).

   Possible problems include:

   - modem equipment that we are hooked up to (locally)
   - problems at remote end
   - something not cabled properly or bad/incorrect cable (either locally or at remote end)

3. If activity on both sides of the scope is seen, but the links are still inactive:

   common →

   - are the link addressess correct? P=1 DTE/DCE 01/03
   - make sure modem testing or circuit testing is not currently in progress
   - verify modem settings

4. If nothing on either side of the scope is seen.

   Possible problems include:

   - scope not hooked up properly
   - scope not functioning
   - multiple problems as previously indicated

5. In the field next to the P/F field on the scope there should be "G"'s, this indicates GOOD frames. "B"'s (BAD frames) will cause continuing problems.

   If "B"'s are on the DTE side:

   - verify cabling
   - verify proper grounding
   - check TN board - run diagnostics

   If "B"'s are on the DCE side:

   - verify cabling
   - verify proper grounding
   - check modem equipment
   - check for problem at remote end (PINET or the other 3B20)
   - run modem to modem testing with remote location

   If the link is connected to PINET, then modem testing will be done with personnel at the local PINET node. Call 1-800-PACKET 1

   If the link is directly connected with the remote location (NCP or another 3B20) then modem to modem testing will be done with personnel at the remote site. (See contact list)

In each of the above cases isolate and correct all problems until the desired results are achieved:

1. links are active

2. level 2 sync - yes

3. RST_COMPLETE

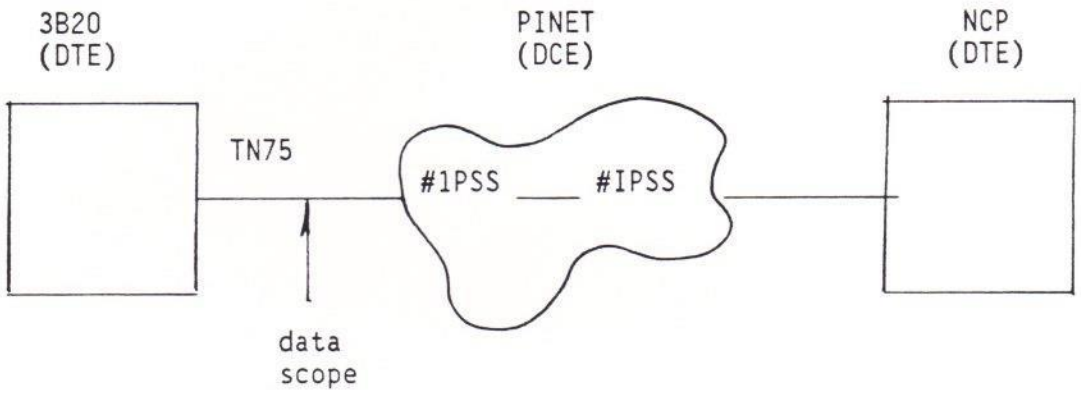
4. The scope should display RR's (after up to about 3 minutes) on both sides after executing the link up shell.

5. The scope should display "G"'s (GOOD frames) on both sides of the link

4. MONITORING "PACKET" LEVEL COMMUNICATIONS

This example is using the official version of X.25 with SVC's (switched virtual circuits). This monitoring can only be done after the link is active with the remote side. This is necessary for further communications to take place i.e., in order for application data to be sent and/or received from the remote end (NCP or another 3B20).

An attempt must be made to send and/or receive application data in order to analyze a problem at this level of communications. Assuming the following environment:



Using SVC's (switched virtual circuits) to send data from the 3B20 to the NCP via the #1PSS and terminate the session.

SAMPLE DISPLAY OF SCOPE WHEN MONITORING THE PACKET LEVEL

Below is sample display of what should be seen on the data scope when sending data from the 3B20 to the NCP via APS. (NEXT PAGE)

```
REPORT DISPLAY WITH REP_COMP AND TIME_ON

12 57 714.6   DCE ADDRESS=01   FRAME=SABM     P=0
12 57 716.5
12 57 724.1   DTE ADDRESS=01   FRAME=UA       F=0
12 57 726.1
13 0  785.5   DTE ADDRESS=03   FRAME=INFO     P=0     NR=0   NS=0
                  GF=1 D=0 Q=0 LCN=0    RESTART INDICATION PACKET
                  CAUSE      = 00 UNDEFINED
                  DIAGNOSTIC = 00 NO ADDITIONAL INFORMATION
13 0  791.8
13 0  800.9   DCE ADDRESS=03   FRAME=RR       F=0     NR=1
13 0  802.9
13 0  826.7   DCE ADDRESS=01   FRAME=INFO     P=0     NR=1   NS=0
                  GF=1 D=0 Q=1 LCN=0    RESTART CONFIRM PACKET
13 0  831.3
13 0  851.5   DTE ADDRESs=01   FRAME=RR       F=0     NR=1
13 0  853.5
13 2  715.5   DTE ADDRESS=03   FRAME=INFO     P=0     NR=1   NS=1
                  GF=1 D=0 Q=0 LCN=1    INCOMING CALL PACKET
                  ADDRESS  - CALLED = 36600011        CALLING = 45500011
                  FACILITY - THROUGHPUT CLASS - CALLED = 9600  CALLING = 9600
                  FACILITY - SIZE     - CALLED 256    CALLING = 256
                  FACILITY - WINDOW   - CALLED 3      CALLING = 3
                  CALL USER DATA FIELD
                  C0 00 00 00 03 01 00 25 80 00 64
13 2  744.1
13 2  752.5   DCE ADDRESS=03   FRAME=RR       F=0     NR=2
13 2  754.6
13 2  790.9   DCE ADDRESS=01   FRAME=INFO     P=0     NR=2   NS=1
                  GF=1 D=0 Q=0 LCN=1    CALL ACCEPTED PACKET
                  ADDRESS  - CALLED = 36600011        CALLING = 45500011
                  FACILITY - THROUGHPUT CLASS - CALLED = 9600  CALLING = 9600
                  FACILITY - SIZE     - CALLED 256    CALLING = 256
                  FACILITY - WINDOW   - CALLED 3      CALLING = 3
13 2  810.3
13 2  830.5   DTE ADDRESS=01   FRAME=RR       F=0     NR=2
13 2  832.4
13 5  21 .1   DTE ADDRESS=03   FRAME=INFO     P=0     NR=2   NS=2
                  GF=1 D=0 Q=0 LCN=1    DATA PACKET PR=0   PS=0   M=0
                 <IDACOM ELECTRONICS LTD   BRINGS TO YOU THE     P T  !!!      >
                 <     THE PROTOCOL TESTER THAT LEADS THE WAY INTO THE FUTURE >
                 < OF DATA COMMUNICATIONS.   NOW A TRUE   'FORTH'   GENERATION>
                 < COMPUTER IS AVAILABLE TO ASSIST WITH SOLVING PROTOCOL      >
                 < PROBLEMS.      >
13 5  238.9
13 5  247.3   DCE ADDRESS=03   FRAME=RR       F=0     NR=3
13 5  249.3
13 5  274.0   DCE ADDRESS=01   FRAME=INFO     P=0     NR=3   NS=2
                  GF=1 D=0 Q=0 LCN=1    RR   PACKET PR=1
13 5  278.6
```

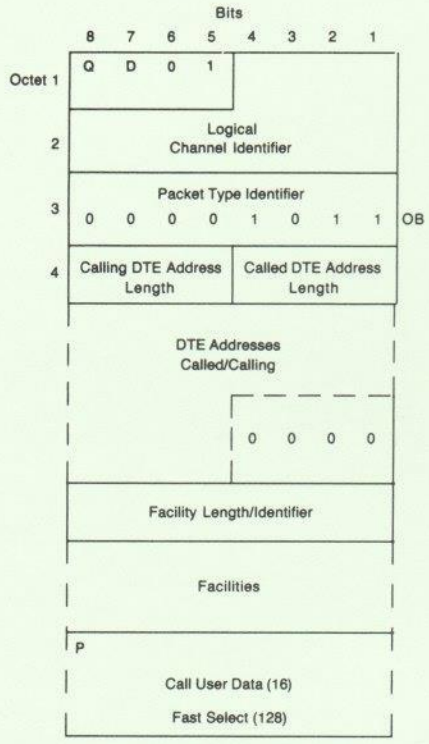
CORPORATE EDUCATION
& TRAINING
PISCATAWAY, NJ

FRAME, PACKET AND SESSION QUICK REFERENCE CARD

COURSE 1E7170
X.25/BX.25 NETWORK
ARCHITECTURE
MARCH 1988

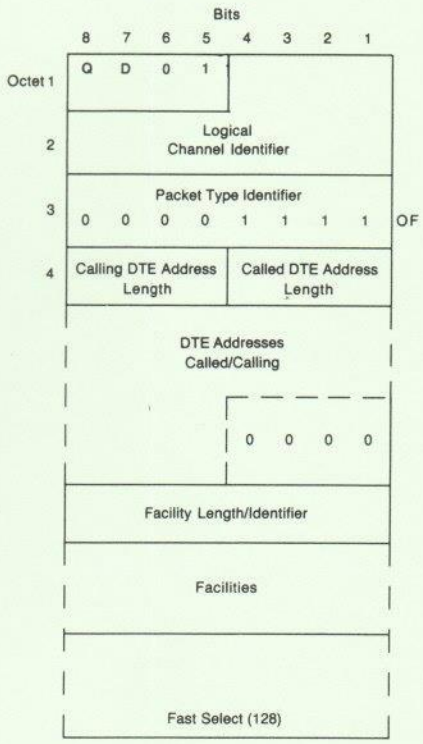
PACKET FORMAT (Modulo 8)

Call Request and Incoming Call

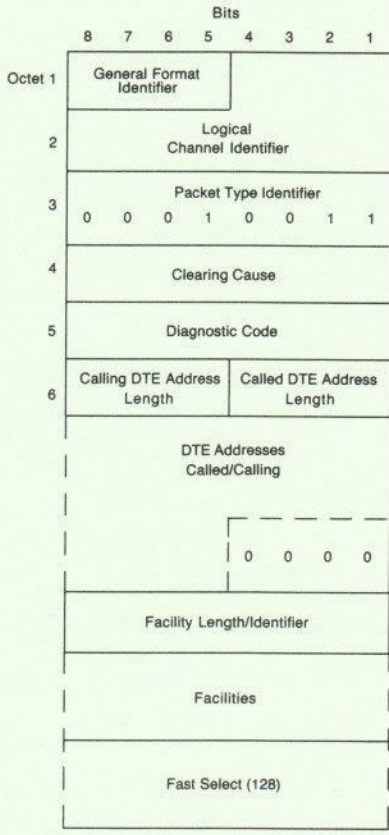


P = Protocol Identifier (co = user defined)

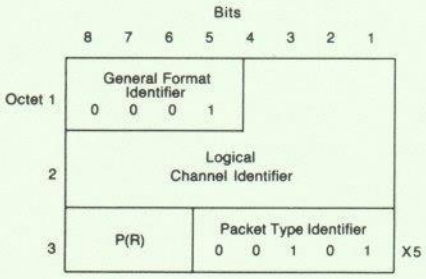
Call Accepted and Call Connected



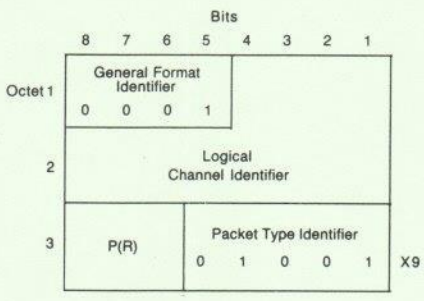
Clear Request and Clear Indication



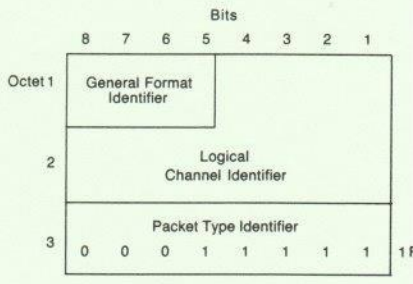
Receive Not Ready



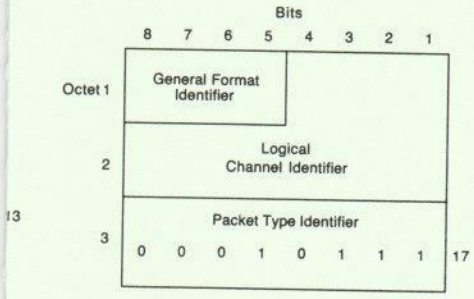
Reject



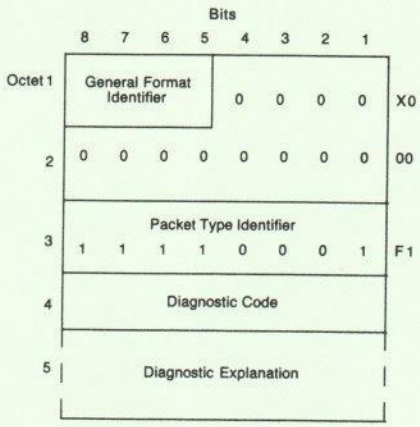
Reset Confirmation



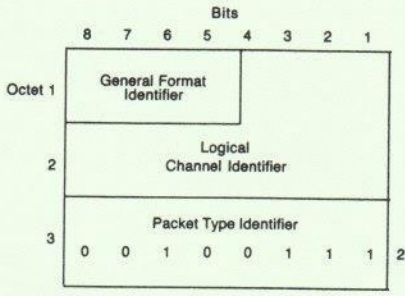
Clear Confirmation



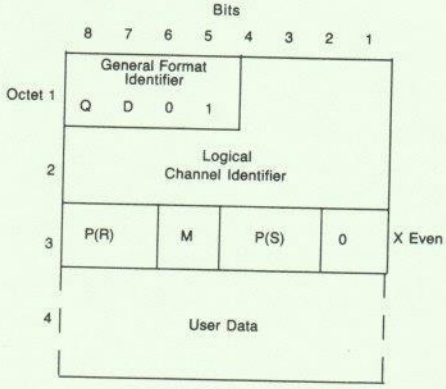
Diagnostic



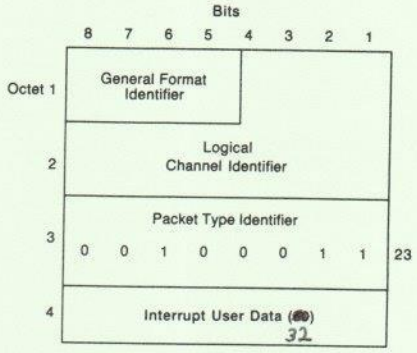
Interrupt Confirmation



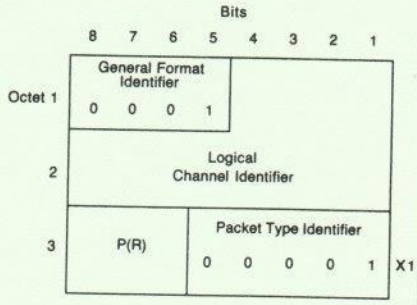
Data



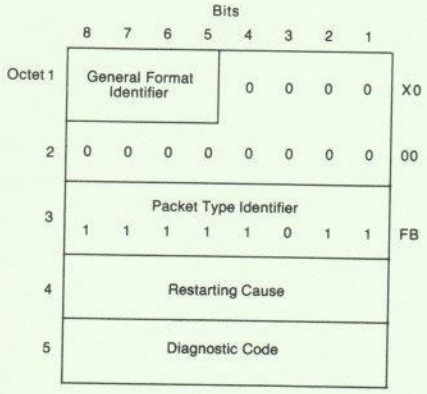
Interrupt



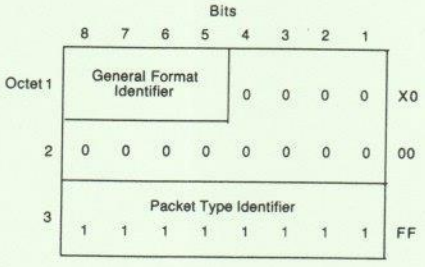
Receive Ready



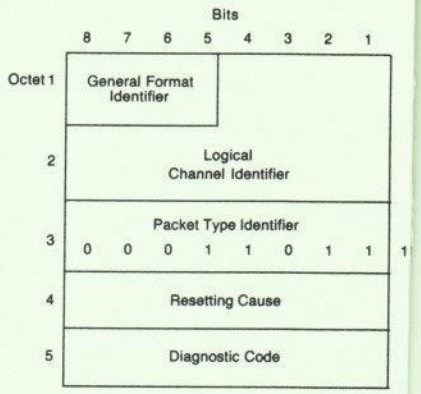
Restart Request and Restart Indication



Restart Confirmation



Reset Request and Reset Indication



Session Level

Session Message Type Identifier

OCTET 1 0010 LLLL Message Type	2X Hex	Octet2 Bits 87654321
Session Establishment and Termination		
Connect - SCM	00	00000000
Accept - SAM	01	00000001
Reject - SRM	02	00000010
Disconnect - SNDM	03	00000011
Disconnect Reject - SRDM	0C	00001100
Abort Disconnect (UNIX) - SADM		
Session Recovery		
Reconnect - SRECM	04	00000100
Reaccept - SREAM	05	00000101
Session Resynchronization		
Session Resynchronization	06	00000110
Resynchronization Ack	0D	00001101

Session Layer Header - HIC (Octet 1)

Hex Value	Bits 8765	
2	0010	Control Message Types e.g.,Connect, Accept
3	0011	Session Parameters e.g., Origin and Dialogue Type, Options
4	0100	Control Data For Higher Level e.g., CDF Control Data Flag
5	0101	User Data For Higher Level e.g., MDF-More Data Flag
1	0001	User Data For Higher Level e.g., Last or Only Data Field
C	1100	User Calling and Called Session Address

3rd OCTET
Session Parameters

OCTET 1 OO11 LLLL Session Parameter Type	3X Hex	Octet 2 Bits 87654321
Origin Type	01	00000001
Dialogue Type	02	00000010
Session Options	03	00000011
Reject Reason	04	00000100
Disconnect Reason	05	00000101
Presentation Type	06	00000110
SPDU Receive Sequence	07	00000111

SESSION CHARACTERISTICS (OCTETS 1,2 &3)

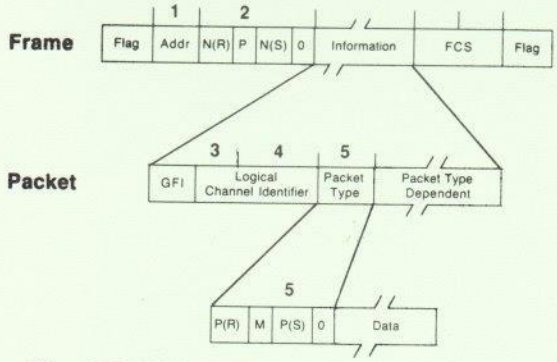
32	01 Origin	05 06 09 0A	Session only Higher-level AFT
	02 Dialogue	00 01 02 03 08	2WS 2WA 1W0 1W1 Restricted
	03 Options	01 02 03	Resync Recovery SPDU size
	06 Presentation	00 01 04	Binary ASCII (default) EBCDIC

REJECT/DISCONNECT REASON

HEX CODE	REJECT
00	Called address unavailable
01	No answer
10	Invalid SCM
11	– Session characterisic invalid
12	– Session address missing
17	– Unknown or unacceptable parameter

	REJECT/DISCONNECT
00	No additional information
50-55	Timer expired or attempt count reached
60	Procedure Error
61	– Unrecognized message
62	– Message not allowed
63	– Data not allowed
64	– Message not available
65	– Message out of sequence
70	Higher Level Request
71	– Higher level failure
72	– Higher level error

Information Frames Contain |



GFI = Q, D, MODULO
2nd Octet of Frame = Frame Type
3rd Octet of Packet = Packet Type
(5th Octet of Frame)

Link Layer Frame Types

Control Frame Type

Type	Encoding 2nd Octet	Hex Value
I	N(R) P N(S) 0	X Even*
RR	N(R) P/F 0001	X 1
RNR	N(R) P/F 0101	X 5
REJ	N(R) P/F 1001	X 9
SABM	001 P 1111	2F/3F
DISC	010 P 0011	43/53
DM	000 F 1111	0F/1F
UA	011 F 0011	63/73
FRMR	100 F 0111	87/97

P/F - Poll or Final Bit
N(R) - Next Expected Frame Number
N(S) - Transmitter Send Frame Number
X - Any Hex Value

begin by Ø + increment

Packet Layer Types

Type	Encoding 3rd Octet	Hex Value
Call Req/Inc	0 0 0 0 1 0 1 1	0B
Call Acc/Conn	0 0 0 0 1 1 1 1	0F
Clear Req/Ind	0 0 0 1 0 0 1 1	13
Clear Conf	0 0 0 1 0 1 1 1	17
Data	P(R) M P(S) 0	X Even
Interrupt	0 0 1 0 0 0 1 1	23
Interrupt Conf	0 0 1 0 0 1 1 1	27
RR	P(R) 0 0 0 0 1	X 1
RNR	P(R) 0 0 1 0 1	X 5
Rej*	P(R) 0 1 0 0 1	X 9
Reset Req/Ind	0 0 0 1 1 0 1 1	1B
Reset Conf	0 0 0 1 1 1 1 1	1F
Restart Req/Ind	1 1 1 1 1 0 1 1	FB
Restart Conf	1 1 1 1 1 1 1 1	FF
Diagnostic	1 1 1 1 0 0 0 1	F1

* No Longer Supported for BX.25 DTEs
P(R) - Next Expected Packet Number
P(S) - Transmitter Send Packet Number
M - More Data Bit
X - Any Hex Value

Cause–OCTET 4	Hex VALUE	Decimal VALUE
For **CLEAR INDICATION** Packets DTE Originated	00	0
Number busy	01	1
Out of order	09	9
Remote procedure error	11	17
Reverse charging acceptance not subscribed	19	25
Incompatible destination	21	33
Fast select acceptance not subscribed	29	41
Invalid facility request	03	3
Access barred	0B	11
Local procedure error	13	19
Network congestion	05	5
Not obtainable	0D	13
RPOA out of order	15	21
For **RESET INDICATION** Packets DTE Originated	00	0
Out of order	01	1
Remote procedure error	03	3
Local procedure error	05	5
Network congestion	07	7
Remote DTE operational	09	9
Network operational	0F	15
Incompatible destination	11	17
Network out of order	1D	29
For **RESTART INDICATION** Packets Local procedure error	00	0
Network congestion	03	3
Network operational	07	7

FACILITY TYPE	LEADING OCTETS 1 CODE	2 VALUE	3 VALUE
Closed User Group	03	XX	—
Reversed Charging	01	01	—
Fast Select	01	X0	—
Reverse Charge and Fast Select	01	X0	—
Throughput Class 7-1200 9-4800 8-2400 A-9600	02	XX	—
Window Size	43	0X	0X
Packet Size 7-128 8-256	42	0X	0X
One Way Logical Channels And Call Barring	41	XX	0X
Refer to CCITT X.25 For Values of the X Variable and Other Facilities			

02, 03

Ø8 = 25
Ø7 = 128

Diagnostics–OCTET 5	Hex VALUE	Decimal VALUE
No additional information	00	0
Invalid P(S)	01	1
Invalid P(R)	02	2
Packet Type Invalid	10	16
For state r1	11	17
For state r2	12	18
For state r3	13	19
For state p1	14	20
For state p2	15	21
For state p3	16	22
For state p4	17	23
For state p5	18	24
For state p6	19	25
For state p7	1A	26
For state d1	1B	27
For state d2	1C	28
For state d3	1D	29
Packet not allowed	20	32
Unidentifiable packet	21	33
Call on one-way logical channel	22	34
Invalid packet type on a PVC	23	35
Packet on unassigned logical channel	24	36
Packet too short	26	38
Packet too long	27	39
Invalid general format identifier	28	40
Restart packet with nonzero in bits 1 to 4 of octet 1, or bits 1 to 8 of octet 2	29	41
Packet type not compatible with facility	2A	42
Unauthorized interrupt confirmation	2B	43
Unauthorized interrupt	2C	44
Unauthorized reject	2D	45
Time expired	30	48
For incoming call	31	49
For clear indication	32	50
For reset indication	33	51
For restart indication	34	52

bytes

Diagnostics–OCTET 5	Hex VALUE	Decimal VALUE
Call setup, call clearing	40	64
Facility code not allowed	41	65
Facility parameter not allowed	42	66
Invalid called address	43	67
Invalid calling address	44	68
Invalid facility length/maintenance action	45	69
Incoming call barred	46	70
No logical channel available	47	71
Call collision	48	72
Duplicate facility requested	49	73
Non zero address length	4A	74
Non zero facility length	4B	75
Facility not provided when expected	4C	76
Invalid CCITT-specified DTE facility	4D	77
Misc.	50	80
Improper cause code from DTE	51	81
Octet non-aligned	52	82
Inconsistent Q bit	53	83
Maintenance action (APS) →	54	84
International problem	70	112
Remote network problem	71	113
International protocol problem	72	114
International link out of order	73	115
International link busy	74	116
Transit network facility problem	75	117
Remote network facility problem	76	118
International routing problem	77	119
Temporary routing problem	78	120
Unknown called DNIC	79	121
Maintenance action (x.75)	7A	122

User Defined

NOTES

NUMBERING SYSTEMS

DECIMAL	OCTAL	BINARY	HEX
0	0	0000	0
1	1	0001	1
2	2	0010	2
3	3	0011	3
4	4	0100	4
5	5	0101	5
6	6	0110	6
7	7	0111	7
8	10	1000	8
9	11	1001	9
10	12	1010	A
11	13	1011	B
12	14	1100	C
13	15	1101	D
14	16	1110	E
15	17	1111	F
16	20	10000	10