NEWCASTLE CONNECTION

Release 1,0 on the
QU68000

D930061e

Document-No. D930061e

Trademarks:
MUNIX for PCS
DEC, PDP for DEC
UNIX for Bell Laboratories

The Newcastle Connection – Release 1.0 on the QU68000
Installation Guide and QU68000 - Specifics

PCS
Pfaelzer-Wald-Str. 36
D-8000 Muenchen 90

ABSTRACT

This document describes how to install Release 1.0 of the Newcastle Connection on the QU68000 and some specific points for that implementation.

1. Reading in the Tape

The software is in tar format on a magnetic tape or several floppies. When it is read in, it will create several files in /usr/doc, /usr/man, /etc, /usr/NCbin and /usr/NClib. Especially /usr/NCbin is very voluminous and you should take precautions against disk overflow! We have chosen not to overwrite existing programs in /bin, /usr/bin or /usr/ucb, but to have special directories for the Newcastle Connection programs. The reasons were

a) We deliver the Newcastle Connection as an add on to Munix, not as an integral part of the basic software.

b) There is a certain overhead involved with programs that are linked with the Newcastle Connection code, and we do not want users without network to share this overhead.

But if you think the programs use too much space in this way, you can simply copy programs from /usr/NCbin over their non-connected counterparts in /usr/bin et al. After that, you must link /usr/NCbin to /usr/bin, by executing the program link, or issuing the system call link("/usr/bin","/usr/NCbin").

The second choice you must make is, whether you want to be able to access remote systems always (with overhead if you are working local) or whether you want to switch the NC on or off, as is your need. In the first case, edit /etc/passwd to replace your login shell with /usr/NCbin/NCsetup. In the second case, leave your login shell as it is, and call NCon before you want to access a remote system. NCon is a small shell script in /usr/bin which sets some environment variables and then executes NCsetup. For a description of NCsetup see *setup* (8NC).

On the tape is a new standard library /usr/lib/libcnc.a and a new C compiler interface ccN. As

```
cc *.o
```

is equivalent to

```
ld /lib/crt0.o *.o -lc,
```

so

```
ccN *.o
```

is equivalent to

```
ld /lib/nc.crt0.o *.o -lcnc.
```

This means, if you have a C program which you want to execute remotely or which shall access remote files, use ccN instead of cc to compile and link it. If

you compile with ccN, the condition

```
#ifdef NEWCASTLE_CONNECTION
```

is true.

The directory /etc contains some tables, plus the spawner /etc/usam and the file server /etc/usrv of the NC. You should include a line

```
/etc/startnc
```

in the file /etc/rc, so that the spawner is always started when the system goes up multi-user. The tables /etc/utab, /etc/pwmap, /etc/groupmap and /etc/map_port_eadr should be empty or nonexistent in the beginning. They are created and filled by ***mksys*** (1NC) and ***unite*** (1NC).

The next choice is, whether you want the directory tree as described in the NC documentation or not. Newcastle suggests to create a directory /nodename and then to move everything under / into this directory. A new /etc/init will then execute at the beginning the system calls

```
uname(&utsname); /* get nodename of this system */
chroot(utsname.nodename); /* chroot to /nodename */
```

The old / is now accessible as /.. . It can be used to store names of other systems, which can then be accessed as /../system1 etc. This is a very clean method to keep other systems logically separated from the own. But there are some drawbacks, which, as our experience shows, can confuse novice users or system administrators. One is, that the system will not reboot, if at system generation time the contents of /usr/sys/name.c are not consistent with the name of the new root directory. The chroot() will then fail and leave the system unable to find /bin/sh, /dev/console etc. One other drawback is, that the minitor still sees the file tree as it is. Instead of loading /sa/restor, the standalone restor program, one would now have to load /nodename/sa/restor.

The Newcastle Connection code does not really demand the directory tree to be set up as mentioned. You could as well access other systems as e.g. /system1. If you choose the first method, you must first cd to /NCetc, execute the procedure buildtree, and then execute

```
/etc/mksys /../system ...
```

for each remote system. Otherwise you just execute

```
/etc/mksys /system ...
```

2. Inclusion of the ethernet driver

The ethernet driver is in the file /usr/sys/ether.o. You should first put this module into the library with the other device drivers:

```
cd /usr/sys
ar rv lib2 ether.o
```

Be careful to correctly fill in the entries in /usr/sys/name.c. Now generate a new unix (newconf). After the configuration, the file conf.h must contain

```
#define ETHER 1
```

and

```
#define ETHADDR ...
```

somewhere. Since /usr/include/sys/ether.h defines MAXPORTS to be 50, you can now create 50 special files in /dev for the basic block ports (see ***bbp*** (4). The following code should be appended to /dev/makefile:

```
bbp:    ; rm -f bbp*
        for i in 0 1 2 3 4 5 6 7 8 9; \
        do /etc/mknod bbp$$i  c 22  $$i; \
        done
```

```
for i in 0 1 2 3 4 5 6 7 8 9;\
do\
    for j in 1 2 3 4;\
    do /etc/mknod bbp$$j$$i  c 22  $$j$$i; \
    done \
done
chmod 0666 bbp*
```

The call "make bbp" will then create /dev/bbp0 to /dev/bbp49.

3. Final comments

We linked many programs together with the new standard library libcnc.a, but could not possibly test them all yet. Most likely cause of a crash will be insufficient stack size. All of the connected programs have at least 8kb of stack size, but it may be that some of them require more. Programs like write which open /dev/tty will not work. If you are missing some programs that you urgently need to be connected, we can easily link them too with the NC code. We are somewhat dependent here on the knowledge of your needs. You can often achieve the desired purpose by writing a small shell procedure, which copys a remote file to your system, executes the program locally and copys the result back.

The Newcastle Connection – Release 1.0
Overview and Installation Guide

Computing Laboratory
University of Newcastle Upon Tyne
Claremont Tower, Claremont Road,
Newcastle Upon Tyne,
U. K. NE1 7RU

ABSTRACT

This document describes how to install Release 1.0 of the Newcastle Connection. It has been tested for PDP-11s running UNIX† Version 7; installation on other systems will most probably not go as smoothly as this document suggests. The main part of this document, describing the installation procedure, is broken into five sections: reading the distribution software, customising the Newcastle Connection software for your system, basic installation, verification, and releasing the Newcastle Connection and "connected" programs for general use. The document also includes sections on administering the Newcastle Connection and on bugs and features peculiar to Release 1.0.

1. Introduction

This document should accompany the tape (or other physical medium) on which your copy of the Newcastle Connection Release 1.0 has been distributed. Whatever the distribution medium, the software has been written on it in UNIX Version 7 "tar" format. Check the label on the tape for density and blocking specifications.

It should be noted that these instructions are intended for installation starting "from scratch" with a standard Bell Version 7 UNIX, and that distributors of the Newcastle Connection would do much of this installation before distribution in order to distribute an integrated system.

The remainder of this section gives a brief overview of how the Newcastle Connection is implemented for Release 1.0, and how it should be installed. This is followed by the actual installation guide, a section on how the Newcastle Connection can be used and controlled by the system administrator, and some bugs (or features) of Release 1.0.

The Newcastle Connection is a transparent layer of software which makes several UNIX systems appear as one. Such a combination of several systems is termed a UNIX United system. It consists of a single global file store which appears to contain only files and directories. In fact, the Newcastle Connection "tags" certain directories as referring to remote UNIX systems, and intercepts all system calls referring to objects on remote systems. Intercepted system calls are then packed into protocol messages which are transmitted to and executed on the remote system, and the values returned are then transmitted back to the originating system. As user identifiers (userids) on

†UNIX is a Trademark of Bell Laboratories.

UNIX are fixed-length integers, they cannot so easily be extended. The Newcastle Connection thus arranges that all operations on a remote system are carried out with a userid valid for that system.

In order to intercept UNIX system calls, every program which uses the network is linked with a Newcastle Connection subroutine library. Routines in this library use the same names as the standard C language interface routines to the UNIX kernel. The true interface routines to the kernel are renamed for use only by the Newcastle Connection. (Programs not linked with the Newcastle Connection library continue to operate normally on their local system, and receive standard UNIX error when they attempt to perform remote accesses.) When a system call is to be carried out on a remote system, the Newcastle Connection forwards the encoded form of the call to a "UNIX server" on that system which has previously been created for the calling process. (The code for the UNIX server lives in the file "/etc/usrv".) In order to create UNIX servers when necessary, every UNIX system in a UNIX United system has a Newcastle Connection "spawner" process which always runs at a known network address. (The code for the spawner, or "UNIX server activation manager", lives in the file "/etc/usam".)

Release 1.0 of the Newcastle Connection supports a single name neighbour space in the UNIX United naming tree, and a single physical network for intercommunication between the various UNIX systems. (A name neighbour space is a set of "system" directories in the UNIX United tree where the paths between the directories are either direct, or consist only of paths through true directories.)The directory structure between name neighbours is replicated on each system. Each system thus stores only its own local files and directories, plus its copy of the shared structure, including specially tagged "system" directories for its name neighbours. Future releases of the Newcastle Connection will allow multiple networks, and more arbitrary placing of "systems" in the global UNIX United naming tree.

Networking with the Newcastle Connection is based on the idea of an address space, which is an abstract communication medium supplied to the Newcastle Connection by underlying software. Hosts on a address space are known by "station addresses", and processes on a host by "port numbers". The allocation of port numbers to processes is dynamic, except that spawner processes always use the same port number on a given address space. Communication on an address space is by means of Remote Procedure Call (RPC) messages, which can be implemented on top of any process-to-process communications facility.

More information on the overall design of UNIX United and the Newcastle Connection can be found in "The Newcastle Connection, or UNIXes of the World Unite!", by D. R. Brownbridge, L. F. Marshall, and B. Randell, in *Software--Practice and Experience*, vol. 12, pp1147-1162, 1982. More information on the remote procedure call protocol used between UNIX systems in a Newcastle Connection Release 1.0 UNIX United system can be found in the document "Newcastle Connection Remote Procedure Call Protocol", which is included in "nroff -ms" format in the files "doc/protocol/[1-4]protocol" on the distribution tape.

The first step in the installation is to read the tape (or other medium) onto a UNIX system. Once this has been done, parts of the code must be modified to reflect the peculiarities of the hardware, software and networks supporting the Newcastle Connection. This is described in Section 3 of this installation guide.

Once the code has been customised, an attempt should be made to execute the command file "install". This creates the Newcastle Connection library, including a new C setup routine, a new "/etc/init" program, a "cc" C compiler which recognises a "-N" option for linking with the Newcastle Connection, the spawner and UNIX servers, and various other programs used to set up and maintain the Newcastle Connection. "Install" is described in Section 4.

Once "install" appears to have succeeded, very careful preliminary testing should be done to ensure that the Newcastle Connection is functioning properly. This should include loop-back testing if permitted by the network driver, or limited testing between two systems. This verification step is described in Section 5.

Once the Newcastle Connection appears to be functioning properly, its "installation" can begin in earnest. This involves describing the parts of the UNIX United file tree stored on this system to the Newcastle Connection, creating the appropriate tables indicating the local user and group ids of UNIX servers initiated for users on other systems, and relinking (or recompiling) all programs which should be able to make use of the overall UNIX United tree. As this may well point up problems with the Newcastle Connection which did not appear during the preceding step, single programs should be relinked and tested before spending a lot of time doing them all. Good starting candidates are the shell ("sh") and list ("ls") commands. This final stage of installation is described in Section 6.

Finally, Section 7 describes the administrative controls available to a system administrator for regulating both incoming and outgoing access to the rest of the UNIX United system. Section 8 details known bugs, restrictions, and other exotica peculiar to Release 1.0; some of these will be attended to in future releases.

2. Reading the Tape

The tape (or other medium) you have received is in UNIX "tar" format, and the density is as specified on the label on the tape box. In order to install the Newcastle Connection you must first make a directory for the source, for example :-

```
mkdir /usr/src/nc
```

You must then make this your current directory and read the tape, thus -

```
cd /usr/src/nc
tar x
```

The file "READ_ME_FIRST" describes what the various directories extracted from the tape contain; all of the directories have a "READ_ME" file with more detailed information in it.

3. Preparing the Newcastle Connection Code

The source code which you have been given assumes that you are using standard Bell UNIX Version 7 on a PDP-11, and that the network interface available to the Newcastle Connection is identical to that of the Rutherford Appleton Laboratories Basic Block Protocol driver. The more your system differs from that assumed, the more modifications you will have to make to the code and installation procedure.

Whenever the Newcastle Connection wishes to make a true UNIX system call, it uses one of the macros defined in the file "h/syscall.h". These macros

define the new names of the system calls; for PDP-11 Version 7, these are usually the old names prefixed with an underscore ("_"). For the standard Bell distribution, renaming these kernel interface routines is accomplished by applying an editor script (supplied with the Newcastle Connection) to the assembler source code distributed by Bell, and assembling the result. These editor scripts are in the directory "sys" of the distribution. The names used by the editor scripts must be consistent with those in "h/syscall.h". For UNIX (look-alike) systems, this renaming process may have to be performed in some other manner. If you are not using the standard Bell Version 7 distribution, the editor scripts may not produce the desired result.

In order to ensure proper initialisation of the Newcastle Connection code in a process, the C language startup routine must be modified to call the function "_ncinit()" before control is passed to the user program. The standard Newcastle Connection distribution includes editor scripts which may be applied against the standard Bell Version 7 distribution source code to produce modified assembler versions of the various C startup routines. These editor scripts are in the directory "csu" on the distribution.

Depending on the UNIX United naming tree which you choose to implement, it may be necessary to ensure that processes have their root directory "/" positioned differently from that of the "init" process which is normally found in "/etc/init". The root directory "/" for the "init" process is set to the physical root of the boot disk device (block 0). If your UNIX United system stores a "system" directory in "/..", processes other than "init" need to have their root directory repositioned. To do this, you should ensure that "init" performs the appropriate true UNIX "chroot(...)" system call. As mentioned, this will be necessary for any UNIX systems where the UNIX United pathnames for other systems are logically above the root directory "/".

In order to make it easier to link programs with the Newcastle Connection, an editor script is provided to add a "-N" option to the standard Bell "cc" command. This option causes the program to be loaded with the Newcastle Connection system calls and routines rather than the standard C library interface to the UNIX kernel. The script is in the file "utility/cc.ed" of the distribution. It will almost certainly fail to produce the desired result if you are using a different C compiler.

Customising the network interface is described in a separate document, "The Newcastle Connection Release 1.0 Network Interface Installation Guide". This document is available in "nroff -ms" format in the file "doc/netman/guide". Briefly, the network interface can be customised to handle various address formats for your network, and to give control to exit routines at key points in "fork" and "exec" processing. This allows some slight deviation from standard UNIX semantics for file descriptors associated with network devices. The modifications essentially involve appropriate network address structure declarations, and writing some very short sections of code for address manipulation.

Finally, you will have to edit the file "h/NClocal.h", which contains all the system-dependent parameters required by the Newcastle Connection. The modifications are documented in detail in the file itself. They include byte order within a short, short order within a long, long/short alignment, and operating system version. Default values for user and group ids of remote users of this system may be defined, and "anonymous" user and group ids MUST be defined. If both default values are defined, the spawner will accept calls from any remote user. Users without entries in "/etc/pwmap" ("/etc/groupmap") will have the userid (groupid) of their UNIX server set to

the default value (see "unite(1NC)"). The anonymous ids will be returned by the Newcastle Connection in "stat" and "fstat" calls for the owner user and group ids of remote files which have no corresponding user/group id on this system. They may be any otherwise unused values. You may wish to include names for them in the files "/etc/passwd" and "/etc/group" so that "ls" prints more meaningful information. You must also define the "magic" device numbers used by the Newcastle Connection for remote systems and alias devices. (Alias devices are devices whose pathnames are used on remote systems rather than on this system. They are useful to avoid having to change programs like "tar" and "lpr" when the devices which they should use are actually on another system.) Finally, "h/NClocal.h" contains some macro definitions used to control conditional compilation of parts of the Newcastle Connection. See the comments in the file.

In order for the "install" command of the next section to work properly, you should check the file "NCprofile", which sets up some shell variables. See the documentation in the file.

4. Creating the Newcastle Connection Code

Once the Newcastle Connection code has been prepared, you should attempt the command "install" in the distribution directory. It accepts the following parameters:

preview - invoke "make" with a -n option to inhibit execution
clean - remove files in preparation for a new "install"
release - copy Newcastle Connection files to public directories
after a successful installation

Invoking "install" without parameters causes it to invoke "make" on the file "NCmake" in the distribution directory, which successively invokes "make" on each of the subsidiary directories. The parameter "preview" prints the commands which would be executed without actually executing them. The parameter "clean" causes all old files produced by previous (partial) calls to "install" to be removed in preparation for a new attempt. The parameter "release" copies files which have been created by "install" to their normal places in public directories. The command "release" simply copies the files without doing an "install" first. You should probably try "install preview" before actually attempting to make the Newcastle Connection.

Should "install" succeed, all is well. Should it fail, diagnosis and repair of the problems must be done by you or your distributor. As the problems encountered can be quite diverse and almost certainly unexpected, no effort is made to detail them here.

As noted below, it would be wise to do some preliminary verification between successfully running "install" and running "release".

5. Verifying the Operation of the Newcastle Connection

Once the runnable versions of Newcastle Connection software and the new C library have been made, the software should be carefully tested before releasing it for general use. In particular, note that the consequences of a Newcastle Connection bug in critical programs such as the shell "sh" can be quite disastrous. With this in mind, it is a good idea to set up one super-user login name which does not result in running a "connected" shell.

The first step in verification is to ensure that when the Newcastle Connection code is included in a user program, it is truly transparent for local operation. This should be true whether or not a Newcastle Connection data

string (called "___N") is present in the process' environment. The command "pretest" in directory "check" of the distribution can be used to perform this testing. Its single parameter specifies a directory which can be used during the test. To do this, change the current working directory to check, and simply invoke it:

```
cd check
pretest /tmp
```

This will either print a message saying that the test has succeeded, or a list of differences between the output of the unconnected and connected versions of the test program "check".

In the normal case, the above test would not have had a "___N" string in the process' environment, as no parent process would have executed the program "NCsetup" to create it. This should be done by issuing the shell command:

```
exec ../utility/NCsetup
```

from the directory "check". Its effect is to create a "___N" string, and to pass it in the environment to the shell program in "/bin/sh" which it invokes with a real UNIX "exec" call.

Having thus created the "___N" string in the environment, run the pretest command file again. Again, this should not result in any differences between the connected and unconnected versions of "check".

At this point, you may consider the Newcastle Connection to be stable enough to proceed with the overall installation. This is done by the command -

```
release
```

in the distribution directory.

After the execution of "release" all of the Newcastle Connection software will have been copied to its normal locations in the public directories. Among the files involved are:

```
/etc/startnc  - program to start execution of the spawner
/etc/stopnc   - program to stop execution of the spawner
/etc/usam     - the spawner program itself
/etc/usrv     - the UNIX server program
/etc/setugi   - used by the UNIX server to change its
                user/group id
/bin/NCsetup  - should be entered in /etc/passwd as
                the shell program for all users authorised
                to use the Newcastle Connection
/lib/libcnc.a - the Newcastle Connection library of UNIX
                system calls
```

Other files are Newcastle Connection commands from the directory "utility" (see the manual pages for them in directory "man"), the modified "cc" command, certain data files used by the Newcastle Connection itself, and the modified C startup routines.

The final phase of verification involves using the so-called wheelkicker in the command file "perform" to verify the operation of the Newcastle Connection in more general cases. In order to do this, you must create an entry in the file system for a remote system. The easiest way to do this is to create a system whose network address is the same as the local system (loop-back).

Otherwise, some Newcastle Connection code (namely "/etc/usam", "/etc/usrv", "/etc/setugi") will first have to be installed on the other system.

In either case, the appropriate file system entries are made with the program "mksys", now installed in "/etc". (See the manual page for "mksys(1NC)" in the directory "man" of the distribution.) You will also have to make an entry in the files "/etc/pwmap" and "/etc/group" so that at least one remote user to or from the pseudo-remote system can access this system via the Newcastle Connection. This is done with the program "unite(1NC)". See the manual page for it.

The final stage of verification is to execute the command file "perform" from the directory "check". It takes two arguments: a local directory and a remote directory. If all is operating correctly, it will report success. When this occurs, you can prepare the Newcastle Connection for normal use as described in the next section.

6. Preparing the Newcastle Connection for Use

For Release 1.0 of the Newcastle Connection, it is suggested that you organise your UNIX United naming tree so that all remote systems are accessed from any one system by pathnames beginning "/..". In order to do this, you must reorganise the local version of the UNIX file store so that the directory or directories in "/.." exist, which involves moving all directories in the existing directory "/" down one level, and then arranging that "/etc/init" execute the appropriate "chroot" call so that all other processes execute with their root set properly.

The command file "buildtree" in the directory "init" of the distribution will perform this reorganisation of the file system for you. Before executing it, you should be VERY sure that you understand what it is trying to do, and that all of its parameters are set properly. As it will tell you when run, you must execute "buildtree" as root, with no file systems mounted, and no other users logged in. It will also check to see that you have placed an object module called "init" in its directory. This should be the same as your normal "init" program, modified to execute a true "chroot" UNIX system call as its first statement. The pathname in the "chroot" must be the same as the $WHERE argument which you provide to "buildtree".

Once "buildtree" has been run, you can create entries for remote systems in the directory "/..". This is done by use of the command "mksys(1NC)", whose syntax is as follows:

mksys <pathname> <identifier>

The <pathname> argument specifies the file system entry to be created for a remote UNIX system; it will normally begin with "/..". The <identifier> will be passed to the network interface routine "_netitoa()", and must provide it with sufficient information to create a proper network address for the remote system. For Release 1.0, the identifier must be in the range [0..255] inclusive. Mksys will create a special file at <pathname>, and create an entry in its own file "/etc/utab". The contents of the file, which is pairs of system names and identifiers, may be printed with the "-p" option of "mksys":

mksys -p

For the Newcastle Connection to present a consistent interface to users on any of the systems making up the UNIX United system, all the directory structure related to remote systems (the directory "/.." and any others you choose to create in it) must be replicated on all of the systems. This is

achieved by executing equivalent sequences of "mksys" commands on each of the systems. (In fact, the Newcastle Connection can be made to operate satisfactorily when the replicated directory (or directories) are inconsistent, but such operation is at your own risk.)

Having created the local part of the UNIX United tree structure on your system, you should use the program "unite(1NC)" to inform the Newcastle Connection of how remote users are to be mapped to users on this system. See the manual page for more information. For any of your local users whom you wish to allow to use the Newcastle Connection, you should modify "/etc/passwd" so that their login shell is "/bin/NCsetup". As mentioned above, this simply creates the Newcastle Connection data string "___N" in the process' environment before executing the standard shell in "/bin/sh".

The final stage in the installation procedure is to relink all programs which you wish to be able to use the Newcastle Connection with the modified C library in "/lib/libcnc.a". This can most easily be done by use of the "-N" option on the modified "cc" command.

7. Use and Administration of the Newcastle Connection

The Newcastle Connection is careful to ensure that the administrator or "super-uses" of each system in a UNIX United system retains control of his individual machine. Through the standard UNIX permission and access control mechanisms, each super-user can control who is allowed access to network devices, which users can run "connected" programs to access remote machines, and which programs are relinked to enable them to access remote machines.

At the lowest level, access to the UNIX device associated with the network is through a normal "open" system call, and so the userid and groupid of the calling process are checked against the permission bits of the device inode.

At a higher level, control of the permissions associated with "/lib/libcnc.a" (the Newcastle Connection system call library) determines whether or not a given program can be (re-) linked to access remote machines, or to be executed by remote users. For example, if he wished, the super-user could "connect" only a very select subset of normal command programs.

In order for any particular user to access a remote system, two conditions must be satisfied:

1. The program which is executed must be connected (i.e., linked with "/lib/libcnc.a"), and
2. the Newcastle Connection data string "___N" must be correctly initialised in the environment before the program is executed.

This initialisation of "___N" is performed by the program "/bin/NCsetup". Standard UNIX permission checking determines which users may execute that program. (For naive users, leaving their login shell set to the default of "/bin/sh" rather than setting it to "/bin/NCsetup" may well be a close enough approximation to prohibiting remote access for them.)

For remote users accessing this machine, the Newcastle Connection provides the system administrator with quite fine control over the privileges with which such users execute on his machine. When a remote user attempts to access a file (or execute a program) on this machine, the spawner process is aware of the remote system identity as well as the remote userid and groupid of the caller. The files "/etc/pwmap" and "/etc/groupmap", which are owned by the super-user, are read by the spawner to determine the local userid and

groupid to which the remote user id is "mapped". In order to service all requests for a remote user process, the spawner creates a UNIX server process with the mapped userid and groupid. This ensures that remote users can only access the local system with privileges under the control of this system's super-user. (In future releases of the Newcastle Connection, it may be possible to exert even finer control by having the spawner choose one of a number of UNIX servers having more or less functionality, depending on the identity of the remote user.)

Note that the mapping from remote to local users may be many-to-one; there is no requirement that each remote user have a distinct local identity. The two tables used by the spawner ("/etc/pwmap", "/etc/groupmap") are maintained by the program "unite(1NC)".

You should be aware that "root" (or super-user) is treated identically to other userids as far as this mapping is concerned: "root" on another machine may be mapped onto some other non-privileged userid on this machine. If several systems are in fact controlled by a single administrator, it may well be advantageous to map "root" onto "root". In particular, this allows all super-user functions for the set of systems to be performed by one single login rather than several. Software updating, disk dumping, process killing (especially useful when the only terminal line into a small computer hangs up), etc., can be done very conveniently. However, you should be very wary of mapping "root" onto "root" if you are even slightly suspicious of other system administrators.

Administration on a larger scale can be performed by cooperating system administrators. As the Newcastle Connection makes an entire UNIX United system appear to be a single UNIX system, and because the Newcastle Connection always executes a file on the machine where it is stored, it is possible to do some rather static load balancing by moving executable programs from one system to another. By naming a program file on a remote machine, the program will be run on that machine, in the environment of the caller. (In this context, "environment" includes the root directory "/", and the userids, groupids, and process ids visible on the caller's system.) Normal shell conventions for I/O redirection are applicable, and all pathnames are interpreted relative to the caller's root directory or working directory.

In order to allow controlled escape from the caller's environment, the Newcastle Connection provides a single new "system" call and corresponding program, both called "excr" (execute with changed root). In fact, there is a family of "excr(2NC)" system calls corresponding to the normal "exec(2)" family. In terms of the "excr" program, a terminal user causes remote "excr" execution with a command of the form:

excr <system> <command> <arg> <arg> . . .

The effect of this is to execute the named command on a remote <system>, passing the indicated arguments. <system> is a pathname referring to a remote system. The "excr" program crudely mimics the normal shell search path: if <command> begins with "/", it is sought relative to the root of <system>; otherwise, "/bin" and "/usr/bin" on the remote system are searched. If the file is marked executable but execution fails, the shell in "/bin/sh" on the remote system is invoked on the file.

For example, both the command "who" and the command "ps" produce their output by reading files specified by root-relative pathnames. Assuming a remote system named "/../rem.unix", then the semantics of normal remote execution imply that both

```
who; ps
```

and

```
/../rem.unix/bin/who; /../rem.unix/bin/ps
```

produce identical results: a list of the users logged in to the local machine, followed by a list of processes belonging to this user on the local machine. However,

```
excr /../rem.unix who
```

will produce the (presumably) intended result of printing a list of the users logged on to machine "/../rem.unix".

Finally, as a UNIX United system tends to be significantly larger than a single UNIX system, it is worthwhile recalling the various pathname abbreviations supplied in the UNIX shell (and hence supported through the Newcastle Connection). The working directory is of course one of the most often-used means of abbreviating pathnames; the Newcastle Connection allows a user process (or processes) to have working directories on a remote machine. The shell provides a variable "$PATH" which lists those directories in which it will search for commands; the Newcastle Connection permits the specification of remote directories in "$PATH". Similarly, other shell variables may be set to allow the user to specify remote pathnames more conveniently. For even more difficult cases, shell command files can be written to further reduce the apparent length of pathnames or arguments supplied by the terminal user.

8. Restrictions and Known Bugs

The UNIX United architecture implies that UNIX systems may be "attached" to arbitrary directories in the global UNIX United naming tree. In future releases of the Newcastle Connection, this generality will be implemented. However, Release 1.0 of the Newcastle Connection implements only a restricted form of UNIX United.

More specifically, Release 1.0 operates with a single name neighbour space and a single physical address space or network. As mentioned above, this implies that all the UNIX systems linked by Release 1.0 must share a replicated inter-system directory structure. Having one or more directories which appear "between" systems allows an arbitrary number of systems to belong to the same (single) name neighbour space. When there is no shared directory structure, only two systems belong to the name neighbour space: one at each end of the path element connecting them. This restriction to a single name neighbour space will be removed in future.

The restriction in Release 1.0 to a single address space effectively implies that the Newcastle Connection will not do any internetworking. Note, however, that the software which presents the address space abstraction at the network interface to the Newcastle Connection may well be performing arbitrary internetworking itself. In future, the Newcastle Connection will itself perform a simple form of internetworking.

For historical reasons, the UNIX servers as implemented in Release 1.0 are not themselves "connected". That is, each UNIX server attempts to interpret each pathname presented to it as being on its local system. In future, UNIX servers will be "transitive", and will forward system calls to more remote systems. They will also be "reflexive", in that a server will occasionally report that a pathname presented to it refers to the caller's system. (This occurs when the system on which the caller runs is not the same as the system where his root directory "/" is located, for example.)

The lack of reflexive and transitive UNIX servers restricts the systems accessible to a running program. If either the root directory "/" or the current working directory of a process are not on the same system as the process is running, the remote server for the directory involved cannot correctly interpret pathnames for any other system. Thus, if "/" is remote, the only root-relative pathnames which are accessible to the process are on the same system as "/". Similarly, if the working directory is remote, the only pathnames relative to that directory which are accessible to the process are on that remote machine.

8.1. Other Restrictions

A "setuid" program allows a program to be run with an effective userid different from that of the caller. It is used to implement "trusted" programs such as "mail(1)" (which needs access to all mailboxes), and to implement proprietary programs needing access to private data. For philosophical reasons, the Newcastle Connection prohibits normal remote execution of "setuid" programs. The problem is that the setuid program would perceive only userids and groupids of the remote machine. For proprietary programs, these may not be meaningful for its own accounting and/or control purposes. Note, however, that "excr" execution of "setuid" programs operates as expected.

The two systems calls "ptrace" and "times" are not intercepted by the Newcastle Connection. For "ptrace" this reflects a disinclination to allow remote debugging, rather than an intrinsic impossibility. (As the Newcastle Connection is transparent to the process being debugged, the debugging should be done on the local system rather than remotely.) For "times", the statistics which it returns have no unambiguous interpretation in the Newcastle Connection context. Comments and suggestions on both of these subjects are solicited.

8.2. Known Bugs

The program "pwd" cannot be made to work with the Newcastle Connection, because of the way it identifies the root directory "/" by searching for an inode which is its own parent. Future releases may provide a new "system" call to replace "pwd".

On the standard Bell distribution, the assembler "as" is written in assembler, and hence cannot be "connected". This implies that the C compiler "cc", which calls it, cannot be completely connected. All the components of "cc" can be successfully connected except the assembler, which implies that you can do some remote accesses from the compiler. Problems occur if your working directory is remote, if you specify a remote object module to be created, or if you attempt remote execution of the compiler.

Executing a "chroot(2)" system call to a remote system does not work as one might expect. Using "excr" instead may be of some use.

The system calls "stat" and "fstat" expose "nasty" system-dependent constants such as inode numbers. Thus, files on different systems may appear identical (in which case "cp" refuses to copy), and programs which depend on inode number tests may not behave correctly. There appears to be no easy way around these difficulties, although future releases may allow programs to explicitly test for remoteness of pathnames and file descriptors. Similar problems arise with other fields in the "stat" buffer; the only ones which are handled reasonably are the userid and groupid.

There is a problem in the Newcastle Connection version of "chdir(2)" which will be fixed in future releases. If the working directory is changed to a remote system, and then moved part of the way back towards the original system, so that it should end up in one of the directories between systems, the current directory the program actually sees is the copy on the remote system, rather than the one on the original system.

Programs which use the file "/dev/tty" explicitly in "excr" mode are likely to fail, as the device actually accessed is the terminal associated with the spawner on the remote system, rather than that associated with the standard input or invoker on the original system. Thus, for example,

```
excr /../rem.unix login
```

will fail as login tries to turn off echo on the terminal device which is unfortunately accessed relative to the changed root. Suggestions are welcome. This is a hazy area where semantics of certain files are not enforced by the kernel, implying that the Newcastle Connection cannot appropriately mimic those semantics in an elegant fashion. However, if the remote spawner was initiated by "/etc/init" at system boot, no terminal is associated with it, and "excr" remote login will succeed. This problem may be fixed in future releases.

NAME

excr – run a program on another system

SYNOPSIS

excr system program arg ...

DESCRIPTION

If the program name begins with a '/' then the program with that name on the named system will be run. However, if there is no '/' then the directories "/bin" and then "/usr/bin" on the named system are searched for the program. When it is found, the program is executed as though its root were on the named file system. If the normal "exec" returns errno = ENOEXEC, an attempt is made to execute the shell on the named file.

This provides the user with a different facility to the one provided by remote execution. Thus the command "/../U5/bin/who" issued on system U1 will print out a list of the users on U1; to obtain a list of the users on U5 the ***excr*** command must be used - "excr /../U5 who".

DIAGNOSTICS

An error message is printed if the program does not exist or cannot be executed.

FILES

/bin/excr

NAME

mail – mail(1) using the Newcastle Connection instead of uucp.

DESCRIPTION

Changes: in "system!person", "system" is now recognised as the name of the root of a system in the Newcastle Connection distributed naming tree, not a uucp(1) system name (especially, there are no embedded "!"s in "system").

Mail spawns a single mail process on each remote system to distribute the mail to all recipients on that system (local mail behaves as usual). Thus

"mail /../unix1!dave /../unix2!dave /../unix1!fred" calls
"mail dave fred" on /../unix1 and
"mail dave" on /../unix2.

SEE ALSO

mail(1) for a full description of this command.

BUGS

Cannot forward messages to remote systems from within mail.
If execution of mail on a remote system does not succeed, it is difficult to know what messages have been sent.
Mail expects all "real" mail commands to live in files of the same path-name on all systems.

NAME

/etc/mkalias – create an alias to a remote file

SYNOPSIS

/etc/mkalias [-f] name identifier

DESCRIPTION

This program facilitates the creation of "aliases" to remote files. An alias is a name that appears to be local but in fact refers to a file of the **same name** on another system. Thus a system with no line printer could have a file "/dev/lp" which was an alias for the line printer on another system, (where it would have to be called "/dev/lp" as well). The operation of an alias is therfore to take the name provided by the user and attempt to perform the requested operation on the file of that name on the indicated system. The "identifier" parameter is of the same type as that provided to "mksys(1NC)": for Release 1.0, it must be an integer in the range [0..255] inclusive. It will be passed to your network interface to identify the name neighbour system on which the "real" version of this file appears.

BUGS

Aliased files must always be accessed relative to "/", otherwise they will not be found.

DIAGNOSTICS

Complains if file "name" exists, unless the "-f" option is present.

NAME

/etc/mksys – make a remote system node

SYNOPSIS

/etc/mksys [-p] | [[-f] name identifier ethernet-address]

DESCRIPTION

This program is used to create the special directory entries needed to communicate with remote systems via the Newcastle Connection. The first parameter is the name that the new entry is to have, and the second is the identifier of the system it is to refer to. The identifier must be in the range [0..255] inclusive for Release 1.0 of the Newcastle Connection. It will be passed to your network interface via the procedure "_netitoa()" to be converted into a physical address when required. The inverse function "_netatoi()" will be called by the Connection to translate a network address into an identifier. Depending on your network interface, it may be possible to encode "identifier" so that it is particularly easy to transform it to a physical address. For example, "identifier" can be used directly as a station address for a Cambridge Ring. The ethernet address is a 6 digit hexadecimal number. The station identifier and the ethernet address of a remote machine must be consistent with this machines' declarations in /usr/sys/name.c

The "-p" option causes "mksys" to print the list of name-identifier pairs known to the local system from the file "/etc/utab", plus the corresponding ethernet addresses from the file "/etc/map_port_eadr".

"Mksys" normally complains if "name" already exists. This can be overridden by the "-f" option.

FILES

/etc/utab - table of name-identifier pairs /etc/map_port_eadr - table of ethernet addresses, indexed by identifier

SEE ALSO

"The Newcastle Connection – Release 1.0: Network Interface Installation Guide", rmsys(1NC), utab(5NC)

NAME

/etc/rmsys – remove a remote system name

SYNOPSIS

/etc/rmsys name

DESCRIPTION

This program removes the special Newcastle Connection entry given by name.

FILES

/etc/utab - file of remote system names and identifiers

SEE ALSO

mksys(1NC), utab(5NC)

NAME

unite – enable a remote user to access the local system

SYNOPSIS

unite [-dgprv] [system [remote_id [local_id]]]

DESCRIPTION

Establishes ***local_id*** as the local surrogate for user ***remote_id*** on remote Unix *system*. e.g.

unite /../unix4 dave david

lets user 'dave' on system '/../unix4' execute processes on the local machine, as if he had logged in as 'david'. A missing local_id is assumed to have the same name as the remote user. A missing remote_id is assumed to mean each user of the remote system is to be mapped to the local user of the same name. Unite normally refuses to map "root" to "root"; any user names with userid 0 are ignored. *Unite* without parameters prints out the current list of remote and local user pairs.

Option "-d" deletes the remote system or user named.

Option "-g" applies ***unite*** to groups instead of users.

Option "-p" can be used to print a single pair, or the entries for a single system.

Option "-r" overrides the default control which will not normally map "root" (or any user name with userid 0) to local "root". It is only effective when an entire system is being united, and has no meaning if combined with '-g' option.

Option "-v" announces each item as it is created.

Both remote_id and local_id may be in numeric form. In this case, the "/etc/passwd" ("/etc/group") file on the relevant system is not accessed. This is useful when creating a United system.

FILES

/etc/pwmap /etc/groupmap, user and group mappings;
system/etc/passwd
system/etc/group
/etc/passwd /etc/group - local and remote user and
group tables.
/etc/utab - table of systems.

DIAGNOSTICS

Complains about incorrect parameters such as non-existent ids or systems.

SEE ALSO

pwmap(5NC), utab(5NC)

NAME

bbp - Basic Block Port Interface

SYNOPSIS

#include <sys/port.h>

DESCRIPTION

The Basic Block Port Interface is a simple network interface, originally developed for the Cambridge Ring, and now adapted to Ethernet. The bbp provides a set of so-called ports, through which processes on different machines (or on the same) can talk to each other. The port is characterized by the structure portinfo in *<sys/port.h>* :

```
struct portinfo {
        short           pi_type;        /* port type,unused for Ethernet */
        unsigned int    pi_inport;      /* bb_port number by which this port
                                         * is addressed by other stations
                                         */
        short           pi_station;     /* destination ring station */
        unsigned int    pi_outport;     /* destination bb_port number */
        unsigned int    pi_accept;      /* acceptable source station number */
        etheradr        pi_ethadr;      /* destination ethernet address */
};
```

The field *pi_type* is unused for Ethernet. For the Cambridge ring this field specifies if the data transfers are protected by parity checks or not. The field *pi_inport* specifies the input port number, by which this port is addressed by other stations. Their output port number, *pi_outport*, must be equal to pi_inport, if they want to talk to this port. The field *pi_station* is a two-byte station number.

A station number is a unique identification of each machine attached to the net. This station number is more manageable than the six byte Ethernet address contained in the field *pi_ethadr*. The ethernet address is mainly used for the hardware address recognition, whereas the station number is used by upper level software. Both the station number and the ethernet address are fixed at system generation time. They are, like the ascii system name, a unique name of the system. Their values can be found in the file */usr/sys/name.c* and can be gotten by the system calls *uname*(2) and *ethsys*(2).

The fields *pi_station*, and *pi_ethadr* together specify the destination machine; the field *pi_outport* is the port number on the destination machine, to which this port wants to talk. The field *pi_accept* specifies the machines from which this port is willing to receive. The values NOONE and ANYONE mean: accept packets from noone or anyone. Any other number means: accept packets only from the station with this number.

It is not necessary to specify the own station number and ethernet address, as the system knows them already and they cannot change.

The portinfo structure is set by an ioctl system call with command BBPSET, and read with command BBPGET.

EXAMPLE

Machine alpha has the station number 3 and the ethernet address

333333333333. Machine beta has the station number 5 and the ethernet address 555555555555. A process on alpha wishes to receive on port number 372. Another process on beta receives on port number 373; The process on alpha specifies

```
struct portinfo alphaport = { 0, 372, 5, 373, 5, {0x5555,0x5555,0x5555}};
        ioctl(fd,BBPSET,&alphaport);
```

whereas the process on beta specifies

```
struct portinfo betaport = { 0, 373, 3, 372, 3, {0x3333,0x3333,0x3333}};
        ioctl(fd,BBPSET,&betaport);
```

Both processes can now talk to each other by normal read and write system calls.

A port can be obtained by successively opening the files /dev/bbp0, /dev/bbp1 etc. If the open returns with errno ENXIO, the port does not exist, if errno is EACCES, the port is already opened. After the open the port must be configured with the command BBPSET. If the ioctl returns with errno EACCES, a port with the same *pi_inport* is already open. Just to get an unused port number, the value DYNAMIC can be given for *pi_inport*. The actual port number can then be gotten with the ioctl-command BBPGET.

EXAMPLE

```
struct portinfo aport = { 0, DYNAMIC, 5, 123, ANYONE, {0x5555,0x5555,0x5555}};
        ioctl(fd,BBPSET,&aport);
        ioctl(fd,BBPGET,&aport);  /* pi_inport contains a free port number */
```

Data is transferred with the normal read and write system calls. However, there is a limitation on the number of bytes that can be transferred with one write. On the Ethernet, the number 1024 is safe. The data of each write system call is sent as a packet over the net. The count of the read system call must be larger or equal than the size of the packet, otherwise the read returns with error EIO. read returns the size of the received packet. If the count for read or write is illegal, error EINVAL is returned.

At any time after BBPSET, the ioctl command BBPENQ will return the following structure, defined in *<sys/port.h>*:

```
struct portenq {
        short           pn_sender;      /* station number
                                           of sender of received block */
        short           pn_sendport;    /* port number
                                           of sender of received block */
        char            pn_xrslt;       /* last block transmission result */
        char            pn_blkavail;    /* a block is available to be read */
        etheradr        pn_sendadr;     /* ethernet address of sender */
};
```

The fields *pn_sender*, *pn_sendport*, and *pn_sendadr* specify the station number, port number, and ethernet address of the sender of a received packet. The field *pn_xrslt* contains the result of the last write. This is normally equal to BB_ACCEPTED on the ethernet, and equal to BB_ERROR only if excessive jams occurred on the net. The field *pn_blkavail* is unequal 0, if a packet has been received, but not yet read.

WARNING

The bbp contains no flow control. Incoming packets are simply discarded if they are not read fast enough. Protocols are entirely the responsibility of upper levels.

SEE ALSO

sbp(4)

FILES

/dev/bbp*

NAME

/etc/map_port_eadr – table of ethernet addresses at this system.

DESCRIPTION

/etc/map_port_eadr is just a linear table of 6 byte ethernet addresses, indexed by the station number (sometimes also called identifier). The address is arbitrary for the 3COM ethernet hardware, but may be hardwired in other controllers at a later time. At any time. there must be a one-to-one correspondance between the station numbers and ethernet addresses of machines connected to the same network.

NAME

/etc/pwmap, /etc/groupmap – table of user and group id mappings for the Newcastle Connection at this system.

DESCRIPTION

/etc/pwmap and */etc/groupmap* contain the tables used by the spawner at this system to determine the user and group ids of servers run on this system on behalf of users of a remote system. The formats of the two files are identical, and consist of a list of system entries, one for each remote system for which one or more users has been authorised. Each system entry consists of a header, and a sequence of fixed-length records for each mapping of a remote id. Each record consists of three 16-bit integers: the first contains flag bits unused in Release 1.0, and the next two contain the remote numeric id and the local numeric id to which it is mapped, respectively.

The header for each system consists of a 16-bit integer giving the number of remote user entries following, a 16-bit length referring to the string name which follows, and a variable-length string which is the path-name of the remote system relative to this system's root. The length field includes the null byte terminating the string.

SEE ALSO

unite (1NC), mksys(1NC), rmsys(1NC).

FILES

/etc/pwmap, /etc/groupmap

NAME

/etc/utab – table of name neighbour UNIX United systems known to the Newcastle Connection at this system.

DESCRIPTION

/etc/utab contains one entry for each name neighbour of the system on which it is stored. Each entry consists of a 16-bit identifier (which must be in the range [0-255] for Release 1.0), a 16-bit length field whose value is the length of the following string plus one for the null byte, and a string which specifies the pathname of the name neighbour relative to the root is this system. The string is stored including the terminating null byte.

The "identifier" will be passed to your network interface routine "_netitoa()" when required to convert it to a physical address for your network. The inverse operation is performed by "_netatoi()", which returns an identifier given a physical address.

This file is maintained by the programs "mksys(1NC)" and "rmsys(1NC)", which can be used to inspect, add, modify, or delete an entry.

The file is used by the Newcastle Connection during "exec" processing to translate physical addresses (the 16-bit identifiers) into system names.

SEE ALSO

unite(1NC), *mksys(1NC)*, *rmsys(1NC)*, "The Newcatle Connection – Release 1.0: Network Interface Installation Guide"

FILES

/etc/utab

NAME

/bin/NCsetup - initialise the Newcastle Connection tables in a process

SYNOPSIS

/bin/NCsetup

DESCRIPTION

This program initialises the "___N" environment string used by the Newcastle Connection before executing the shell. It should be named as a user's shell in the file "/etc/passwd", for all those the users who are to have access to the Connection.

DIAGNOSTICS

NAME

/etc/startnc - starts up the file server spawner
/etc/stopnc - closes down the file server spawner

SYNOPSIS

/etc/startnc [-d]
/etc/stopnc

DESCRIPTION

/etc/startnc starts up the UNIX server spawner, in the file "/etc/usam", and stores its process id in the file "/etc/usampid". If the spawner was already running, the program will shut it down before starting the new process. The program also handles new releases of the server software, which should be placed in the files "/etc/usrv.new", "/etc/setugi.new" and "/etc/usam.new" (the Newcastle Connection make files will do this automatically). The versions being replaced will be moved to the files "/etc/usrv.old", "/etc/setugi.old" and "/etc/usam.old". The option **-d** is used to select the version of the spawner held in the file "/etc/usam.dbg". This is conventionally a debugging version of the spawner, and the option should only be used whilst testing the system.

/etc/stopnc closes down the UNIX server spawner by sending the signal SIGTERM to the process whose id is contained in the file "/etc/usampid".

FILES

/etc/usampid
/etc/usam
/etc/usam.dbg
/etc/usam.new
/etc/setugi
/etc/setugi.new
/etc/setugi.old
/etc/usrv.new
/etc/usrv.old
/etc/usrv.old

SEE ALSO

usrv(8NC), setugi(8NC), usam(8NC)

DIAGNOSTICS

A message will be printed when there is a new software release installed. If the caller is not superuser or if there is a problem with the execution of the spawner program, an error message will be printed.

NAME

/etc/usam - initiate a UNIX server for a remote client

SYNOPSIS

/etc/usam [root directory [working directory]]

DESCRIPTION

This program listens on a fixed port number for an incoming request for remote service. In response, it initiates a UNIX server on another port and returns this port number to the client, who now deals directly with its own UNIX server. The program also performs user/group validation and mapping for the incoming request, allowing the local system manager to maintain control of the user population. The parameters passed to the spawner allow the initiator to control exactly where the spawner lives in the file store hierarchy, and therefore to control where incoming users' UNIX servers live and the image of the file system that those users see. The default value for both fields is "/".

The fixed port number used by all spawners on your network is controlled by the macros "SET_USAM_PORT" and "USAM_INIT" in the file "h/netlocal.h" of the distribution directory of the Newcastle Connection.

FILES

/etc/pwmap, /etc/groupmap

SEE ALSO

usrv(8NC), unite(1NC), startnc(8NC), stopnc(8NC), pwmap(5NC), utab(5NC), "The Newcastle Connection – Release 1.0: Network Interface Installation Guide"

DIAGNOSTICS

Reports will be given on the console in the event of errors;

BUGS

No means of quiescing the spawner, without stopping it all together. Spawners started after boot time create servers which have an associated terminal ("/dev/tty"). This means that "excr" remote login cannot succeed.

NAME

usrv - UNIX server for a remote client

SYNOPSIS

/etc/usrv

DESCRIPTION

This program is spawned in response to incoming requests for service and provides a remote user with the facilities of the normal UNIX system file interface.

SEE ALSO

usam(8NC)

DIAGNOSTICS

Standard UNIX error return codes are handed back to clients in the external "errno" of the caller's program.

Newcastle Connection –Rel. 1.0 und CADMUS
Installationsanweisung

PCS
Pfälzer-Wald-Str. 36
D-8000 München 90

ABSTRACT

Nach dieser Anleitung können Sie eine nachträgliche Installation der Newcastle Connection auf dem CADMUS 9200 vornehmen.

1. Sie benötigen:

1.1. Dokumentation:

EXCR(1NC)	MAIL(1NC)	MKALIAS(1NC)	MKSYS(1NC)
RMSYS(1NC)	UNITE(1NC)	BBP(4)	ETHMAP(5NC)
PWMAP(5NC)	UTAB(5NC)	NCSETUP	STARTNC(8NC)
USAM(8NC)	USRV(8NC)		

1.2. Software:

- neue Standard-Library "usr/lib/libcnc.a", mit der Sie Ihre Programme binden müssen, die Sie "remote" ausführen wollen
- neues Compiler-Interface ccN
- bereits neu gebundene Standard-Utilities in /usr/NCbin Sie benötigen z.Z. etwa 6500 Blöcke auf der Platte. Falls Sie diesen Speicherplatz minimieren wollen, konsultieren Sie bitte PCS-Dokument D930061e "Installation Guide and QU68000-Specifics"

Falls Sie Zusatzpakete wie "Berkeley" oder "MED" gewählt haben, achten Sie bitte darauf, dass auch diese in einer "NC"-Version vorliegen.

1.3. Hardware:

Sie sollten ihre ETHERNET-Controller eingebaut haben. Anderenfalls können Sie nicht testen, ob ihre Newcastle-Connection einwandfrei arbeitet.

2.

Führen Sie nun folgende Schritte **auf jedem Rechner** aus:

2.1. Software einspielen

2.2.

Fügen Sie mit dem Editor in */etc/rc* die Zeile */etc/startnc* ein Die Newcastle-Connection wird so beim Uebergang in den Multi-User-Mode gestartet.

2.3.
Andern Sie /usr/sys/name.c:

```
#include "sys/utsname.h"

struct utsname utsname = {
    "cadmus",
    "pcs",      = Rechnername
    "Jan84",
    "1.5"
};

struct ethname ethname = {
    { 0x1234 0x5678 0x9abc},    = ETHERNET-Adresse
    0       = Knoten-Nummer
};
```

Sowohl Rechnername als auch ETHERNET-Adresse und Knoten-Nummer müssen im Netz eindeutig sein. Ansonsten sind Sie frei in Ihrer Wahl. Beachten Sie aber folgende Grenzen:

Name: 1..6 Buchstaben / Ziffern
Adresse: 12 hex-Ziffern
Nummer: 0..255

Wir empfehlen Ihnen aber, eine leicht zu merkende Kombination dieser drei Parameter zu wählen, z.B.:

```
#include "sys/utsname.h"

struct utsname utsname = {
    "cadmus",
    "pcs5",
    "Jan84",
    "1.5"
};

struct ethname ethname = {
    { 0x5555, 0x5555, 0x5555},
    5
};
```

Bei älteren Betriebssystem-Versionen nehmen Sie die Anderungen mit dem Editor vor. In neueren Versionen erfolgen sie während der Generierung.

2.4.
Generieren und booten Sie nun alle Systeme, die Sie in Ihrem Netz benötigen.

2.5.
Auf jedem Rechner wird für jeden Partner-Rechner die Prozedur */etc/mksys* mit den Parametern des Partner-Rechners aus dessen *name.c* ausgeführt.

Beispiel:

Drei Rechner in einem Netzwerk:

Erstes System: Namen in "pcs1" , Adresse in 1111 1111 1111,
Nummer in 1 ändern

Zweites System: Namen in "pcs2" , Adresse in 2222 2222 2222,
Nummer in 2 ändern

Drittes System: Namen in "pcs3" , Adresse in 3333 3333 3333,
Nummer in 3 ändern

ALLE SYSTEME GENERIEREN .

auf erstem System:

"mksys /pcs2 2 222222222222 " "mksys /pcs3 3 333333333333 "

auf zweitem System:

"mksys /pcs1 1 111111111111 " "mksys /pcs3 3 333333333333 "

auf drittem System:

"mksys /pcs1 1 111111111111 " "mksys /pcs2 2 222222222222 "

2.6.

Mit */dev/make*bbp erzeugen Sie nun in */dev* die Geräteeinträge für ETHERNET.

Sie können nun die Newcastle-Connection benutzen. Dazu führen Sie */usr/bin/NCon* aus.

Alle Kommandos aus */usr/NCbin* können Sie "remote" ausführen, wobei der Name des Rechners, auf dem das Kommando ausgeführt wird, dem Filenamen voranzustellen ist:

z.B.: *ls -l /pcs2/usr/lib*