CONVEX POSIX Concepts

Document No. 710-005030-000

First Edition
December 1989

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX POSIX Concepts
Order No. DSW-312
First Edition

Printed in the United States of America

Revision Information for
CONVEX POSIX Concepts

Edition	Document No.	Description
First	710-005030-000	Initially released with ConvexOS V8.0 in December 1989.

Table of Contents

Appendices

Preface

Purpose and Audience

This document provides details on the Portable Operating System Interface for Computing Environments IEEE Std 1003.1-1988 (POSIX.1), POSIX.1 functions, ConvexOS-specific extensions, ANSI C programming language as it relates to POSIX.1, and what POSIX.1 conformance means.

The *CONVEX POSIX Concepts* document is intended for customers wishing to know about POSIX.1 and how it relates to ConvexOS.

Organization

This document is organized into the following chapters:

- Chapter 1 describes POSIX.1, conformance, and how these relate to ConvexOS.
- Chapter 2 discusses POSIX.1 functions as they relate to ConvexOS specifics, CONVEX extensions, and backward compatibility.
- Chapter 3 contains information about behaviors of POSIX.1 and non-POSIX.1 processes.
- Chapter 4 describes the interaction among POSIX.1, ANSI C, CONVEX C V4.0, and other language bindings.
- Chapter 5 provides two examples of making an application compatible with POSIX.1.
- Appendix A lists the POSIX.1 functions.
- Appendix B contains a glossary of POSIX terms.
- Appendix C provides instructions for reporting problems to the CONVEX Technical Assistance Center (TAC).

Notational Conventions

The following conventions are used in this document:

- Words enclosed in rounded rectangles indicate keyboard keys that you press. For example, (RETURN) refers to the carriage return key. Words separated by a hyphen and enclosed in rounded rectangles indicate two keys that you must press simultaneously. For example, (CTRL-X) indicates that you must press the (CTRL) key while simultaneously pressing the keyboard ⊗ character key.
- **Boldface** type indicates user-entered information for a computer program. You should enter these command sequences exactly as they appear.
- Brackets ([ ]) designate optional entries. Brackets are also used to distinguish parts

of command strings that may be omitted.

- A horizontal ellipsis (. . .) shows repetition of the preceding item(s).
- A vertical ellipsis shows continuation of a sequence where not all of the statements in an example are shown.
- Commands, utilities, and files that are documented in the *ConvexOS Programmer's Reference* are italicized; occurrences that include a number enclosed in parentheses refer to the appropriate section of the manual. For example, *vmstat*(1) means that documentation for the *vmstat* command is located in Section 1.
- The | symbol is used in command sequences in which you must pick no more than one alternative from a list of command options. For example, in the following command sequence,

  (fp)> **s[et] s[pu-selftest]** = [*d[isable]* | *e[nable]*]

  you must choose either *d[isable]* or *e[nable]*, but not both.
- The percent symbol (%) signifies the standard C shell user prompt.

Associated Documents

Using ConvexOS software successfully may require information not within the scope of this manual. The following documents are provided by CONVEX to help you with POSIX.1:

- *CONVEX POSIX Impacts* identifies changes POSIX.1 and ANSI C have on ConvexOS V8.0 and discusses binary backward compatibility.
- *CONVEX POSIX Conformance* defines how ConvexOS V8.0 relates to POSIX.1, discusses implementation-defined features, and is used in correlation with the IEEE Std 1003.1-1988.
- *IEEE Std 1003.1-1988* describes the requirements for the portable operating system interface.
- *ConvexOS Programmer's Reference* documents ConvexOS commands, utilities, and files.
- *CONVEX ANSI C Concepts* contains information on CONVEX C V4.0, ANSI C, compilers, and converting programs to C V4.0..
- *CONVEX C User's Guide* describes how to use the CONVEX C compiler.

Ordering Documentation

To order CONVEX documentation, complete the CONVEX Documentation and Subscription Service Order Form enclosed in the Documentation Catalog included with this manual.

In some situations, the current edition may not be desired. To receive a specific edition of a manual, contact the local CONVEX sales office or call the Technical Assistance Center.

Technical Assistance

If you have questions that are not answered in this book, contact the CONVEX Technical Assistance Center. Use the following phone numbers:

Within the continental U.S.	1(800)952-0379
From locations in Alaska, Hawaii, & Canada	1(214)497-4379
From all other locations	Contact the nearest CONVEX office.

Reader's Forum

If you wish to mail your comments to us, please use the form at the end of this manual and list the document page number with your questions and comments. Thank you.

Chapter 1
POSIX and ConvexOS

1.1 What Is POSIX?

In the early 1980's, /usr/group began a standards movement. Along the way, it was joined by other groups wanting to produce a UNIX-related standard. These efforts resulted in POSIX. POSIX actually refers to a group of proposed standards sponsored by various working committees of the IEEE. The Portable Operating System Interface for Computer Environments IEEE Std 1003.1-1988 (POSIX.1) is the first of the POSIX standards to be adopted. It was ratified on August 22, 1988 and represents a standard system call interface and environment based on the UNIX operating system. It is intended to support application portability at source-code level.

POSIX.1 describes external characteristics and facilities that are important to application developers rather than internal construction techniques used to achieve these capabilities. Special emphasis is placed on those functions and facilities needed in a wide variety of commercial applications. POSIX.1's objective is for a strictly conforming application to compile on a new host without any code modifications.

ConvexOS V8.0 is an implementation of the Berkeley UNIX operating system containing POSIX.1 functionality with extensions for supercomputer environments. ConvexOS V8.0 retains backward compatibility for existing binaries and sources.

POSIX is based largely on UNIX Seventh Edition, UNIX System III, UNIX System V, BSD V4.2, and BSD V4.3 documentation.

CONVEX will keep its operating system current with all areas of POSIX as each working committee completes and ratifies its group of standards. Interested customers can monitor the progress of POSIX through participation in IEEE and by reading the comp.std.unix network news group.

Enhancements of POSIX.1 on ConvexOS V8.0 are outlined in the *CONVEX POSIX Impacts* document released to all customers. The scope of that document is limited to POSIX.1 and how it might affect your application.

How CONVEX interprets POSIX.1 is contained in the *CONVEX POSIX Conformance* document. This manual describes all the implementation-defined features required by the IEEE.

1.2 Why Use POSIX?

POSIX is more of an environment than anything else. As mentioned earlier, there are many UNIX-based implementations in use today. POSIX will be standardizing many of the common areas for operating systems and interfaces to them, so application portability should be easier. That is why the IEEE has dealt with external characteristics and facilities and not internal constructions. POSIX standards should be seen as a way of bringing many diverging systems together at last.

1.3 Working Committees

The following list indicates each POSIX working committee and its area of standardization:

Committee	Subject
1003.0	POSIX Guide
1003.1	Operating System Interface
1003.2	Shell and Tools Interface
1003.3	Verification and Testing Methods
1003.4	Real Time Extensions
1003.5	Ada Bindings
1003.6	Security Extensions
1003.7	System Administration
1003.8	Distribution Services
1003.9	FORTRAN Bindings
1003.10	Supercomputing Application Environment
1003.11	Transaction Processing
1201.x	Interfaces for User Portability

Chapter 2
POSIX and ConvexOS Specifics

2.1 Introduction

In POSIX.1, the IEEE requires that a document be created "for an implementation claiming conformance." The *CONVEX POSIX Conformance* document contains all implementation-defined features for POSIX.1 but no information on extended facilities (that is, CONVEX extensions and information on backward compatibility). This CONVEX-specific information is documented in this chapter. Subheadings are identical to chapter names in POSIX.1 (where CONVEX has additional information to discuss) with backward-compatible and CONVEX-extension sections. Each section contains a function-by-function breakdown.

2.2 Process Primitives

2.2.1 Backward Compatibility

execve(): In former versions of the operating system, when a program was *setuid()* to nonsuperuser but was executed when the real UID is "root," the program had the powers of a superuser as well.

kill(): Previous versions of the operating system based the permission test solely on a match of the effective UIDs of the sender and receiver.

sigaction(): The *sigaction* function replaces the *sigvec* function.

The *struct sigaction* replaces the *struct sigvec*.

Signal-handling functions formerly returned type *int*; now they return type *void*. This may be the source of many compile-time warnings.

By default, previous releases restarted system calls on signals; the default now is *not* to restart system calls. This is accomplished by initializing the *sa_flags* bit _SA_INTERRUPT to be set when a program is exec'ed; by clearing the bit, the old behavior may be obtained.

sigprocmask(): The *sigprocmask()* function replaces *sigblock()* and *sigsetmask()*.

wait(): Previous releases of the operating system documented use of a pointer to *union wait*, where the current release advocates use of a pointer to integer. To convert existing code, simply change calls like

```
union wait wait_union;

wait(&wait_union);
```

to one of the following:

```
wait(&wait_union.w_status);

wait((int *)&wait_union);
```

Existing code will then continue to work as before.

Previous versions of the operating system by default restarted the *wait* family functions when a signal was received while awaiting termination of a child. Refer to *sigaction*(2) for details of controlling this behavior.

2.2.2 CONVEX Extensions

exec():

```
exect(name, argv, envp);

char *name, *argv[], *envp[];
```

The *exect* version is used when the executed file is to be manipulated with *pattach*(2). The program is forced to single step a single instruction giving the parent an opportunity to manipulate its state.

sigaction(): Normally, signal handlers execute on the current stack of the process. This may be changed, on a per-handler basis, so that signals are taken on a special stack.

As an extension, if the _SA_ONSTACK bit is set in *sa_flags*, the system will deliver the signal to the process on a signal stack specified with *sigstack*(2).

An alternate mode of signals can be used by setting the _SA_PARALLEL bit in *sa_flags*. When this bit is set, signals that are masked by the delivery of a signal will not be reflected in the masks returned by *sigblock*(2) and *sigsetmask*(2) system calls. It also changes the action taken upon return of the signal handler to unblock signals blocked when the handler was invoked rather then resetting the mask to what it was before signal delivery.

The handler routine can be declared:

```
void handler(sig, code, scp);
int sig, code;
struct sigcontext *scp;
```

sig Signal number into which hardware faults and traps are mapped.

code Parameter that further defines the type of hardware exception that occurred.

scp Pointer to the *sigcontext* structure (defined in < signal.h>) and used to restore the context from before the signal.

The following defines the mapping of hardware traps to signals and codes. All of these symbols are defined in < signal.h >.

Hardware condition	Signal	Code
Arithmetic traps:		
Integer overflow	SIGFPE	FPE_INTOVF_TRAP
Integer division by zero	SIGFPE	FPE_INTDIV_TRAP
Floating overflow trap	SIGFPE	FPE_FLTOVF_TRAP
Floating/decimal division by zero	SIGFPE	FPE_FLTDIV_TRAP
Floating underflow trap	SIGFPE	FPE_FLTUND_TRAP
Reserved Operand trap	SIGFPE	FPE_RESOP_TRAP
Segmentation Violations:		
Read access violation	SIGSEGV	SEG_READ_TRAP

Write access violation	SIGSEGV	SEG_WRITE_TRAP
Execute access violation	SIGSEGV	SEG_EXEC_TRAP
Invalid segment	SIGSEGV	SEG_INVSDR_TRAP
Invalid page table page	SIGSEGV	SEG_INVPTP_TRAP
Invalid memory reference	SIGSEGV	SEG_INVDATA_TRAP
I/O access violation	SIGSEGV	SEG_IOACC_TRAP
Ring Violations:		
Inward address reference	SIGBUS	BUS_INWADDR_TRAP
Outward ring call	SIGBUS	BUS_OUTCALL_TRAP
Inward ring return	SIGBUS	BUS_INWRTN_TRAP
Invalid syscall gate	SIGBUS	BUS_INVGATE_TRAP
Invalid return frame length	SIGBUS	BUS_INVFRL_TRAP
Illegal instruction:		
Error exit instruction	SIGILL	ILL_ERRXIT_TRAP
Privileged instruction	SIGILL	ILL_PRIVIN_TRAP
Undefined op code	SIGILL	ILL_UNDFOP_TRAP
Trace pending	SIGTRAP	
Bpt instruction	SIGTRAP	

wait():

```
#include <sys/time.h>
#include <sys/resource.h>

int pid;
pid = wait3(status, options, rusage);
int *status;
int options;
struct rusage *rusage;

int pid;
pid = cvxwait(status, options, rusage);
int *status;
int options;
struct cvxrusage *rusage;
```

wait3() provides an alternate interface for programs that must not block when collecting the status of child processes. The *status* and *options* parameters are defined as above. If *rusage* is nonzero, a summary of resources used by the terminated process and all its children is returned (this information is currently not available for stopped processes).

cvxwait() is identical to *wait3()* but returns a struct cvxrusage instead of a struct rusage. The primary difference between these is that struct cvxrusage contains information about the level of parallelism of the terminated process.

2.3 Process Environment

2.3.1 Backward Compatibility

getgroups(): In previous versions of the operating system, the array filled in by *getgroups()* was of type *int*; it is now of type *gid_t* .

getpgrp(): Previous versions of the operating system specified argument *pid* to *getpgrp()*. A value

of zero meant the current process. This mechanism is supported only in backward-compatible mode.

setpgid(): This function was previously known as *setpgrp()*. The functionality of *setpgrp()* was essentially that of *setpgid()* without the restrictions on session membership.

setuid(): *seteuid()*, *setruid()*, *setegid()*, and *setrgid()* set the effective and real user and group IDs of the current process.

times(): Previous versions of *times()* filled the struct tms members with units measured in 60ths of a second. **CLK_TCK**s are not necessarily the same.

Previous versions of *times()* always returned zero or -1.

2.3.2 CONVEX Extensions

getgroups():

```
#include <sys/types.h>
#include <unistd.h>

int ngrps, gsize;
ngrps = int_getgroups(gsize, intgidset);
int *intgidset.
```

Usage is otherwise as for *getgroups()*.

2.4 Files and Directories

2.4.1 Backward Compatibility

access(): Previous versions of the operating system allowed an empty path name as a synonym for the current directory.

chdir(): Previous versions of the operating system allowed an empty path name as a synonym for the current directory.

chown(): The *owner* and *group* parameters were formerly declared to be type *int*, although the high word was unusable.

creat(): Previous versions of the operating system did not allow the EINTR errno to occur by default (refer to *sigvec()*).

directory(): Previous versions of the operating system wanted <sys/dir.h>, not <dirent.h>, to be included.

link(): Previous versions of the operating system allowed an empty path name as a synonym for the current directory.

open(): Previous versions of the operating system allowed an empty path name as a synonym for the current directory.

rename(): Previous versions of the operating system allowed an empty path name as a synonym for the current directory.

rmdir(): Previous versions of the operating system allowed an empty path name as a synonym for the current directory.

stat(): Previous versions of the operating system allowed an empty path name as a synonym for the current directory.

The file status information was previously expressed as follows:

```
#define   S_IFMT      0170000    /* type of file */
#define   S_IFIFO     0010000    /* fifo special */
#define   S_IFCHR     0020000    /* character special */
#define   S_IFDIR     0040000    /* directory */
#define   S_IFBLK     0060000    /* block special */
#define   S_IFREG     0100000    /* regular */
#define   S_IFLNK     0120000    /* symbolic link */
#define   S_IFSOCK    0140000    /* socket */
#define   S_ISUID     0004000    /* set user id on execution */
#define   S_ISGID     0002000    /* set group id on execution */
#define   S_ISVTX     0001000    /* refer to sticky(8) */
#define   S_IREAD     0000400    /* read permission, owner */
#define   S_IWRITE    0000200    /* write permission, owner */
#define   S_IEXEC     0000100    /* execute/search permission, owner */
```

The mode bits 0000070 and 0000007 encode group and others permissions (refer to *chmod()*).

unlink(): Previous versions of the operating system allowed an empty path name as a synonym for the current directory.

2.4.2 CONVEX Extensions

chmod(): The _S_ISVTX mode bit is added (refer to *sticky*(8)).

chown(): A value of _SAME_UID passed for owner or _SAME_GID for group results in the current owner or group remaining unchanged.

If the final component of *path* is a symbolic link, the ownership and group of the symbolic link is changed, not the ownership and group of the file or directory to which it points.

directory():

```
long telldir(dirp);
DIR *dirp;

seekdir(dirp, loc);
DIR *dirp;
long loc;
```

seekdir sets the position of the next *readdir* operation on the *directory stream*. The new position reverts to the one associated with the *directory stream* when the *telldir* operation was performed. Values returned by *telldir* are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the *telldir* value may be invalidated due to undetected directory compaction. It is safe to use a previous *telldir* value immediately after a call to *opendir* and before any calls to *readdir*.

stat():

```
lstat(path, buf);
char *path;
struct stat *buf;
```

lstat is like *stat* except in the case where the named file is a symbolic link, in which case *lstat* returns information about the link, while *stat* returns information about the file the link references.

CONVEX provides the following file test macros:

_S_IFLNK(st.st_mode) True if a symbolic link

_S_ISSOCK(st.st_mode) True if a socket

The file mode bits will contain _S_ISVTX if the sticky bit is set (refer to *sticky*(8)).

2.5 Input and Output Primitives

2.5.1 Backward Compatibility

close(): The EINTR return value was formerly difficult to observe (refer to *sigvec()*); it is now the default, and the use of non-POSIX extensions is required to avoid it (refer to *sigaction()*).

read(): The O_NONBLOCK mode of operation was formerly known as O_NDELAY.

The current implementation of O_NONBLOCK is slightly different than the previous one in that O_NONBLOCK will affect no processes other than the caller, whereas O_NDELAY would affect all processes with file descriptors open for that device.

The error return EWOULDBLOCK occurs in backward-compatible mode in those situations that now return EAGAIN.

write(): Where previous versions of the operating system returned EWOULDBLOCK for *errno*, EAGAIN is now returned.

The default signal behavior is now to interrupt and not restart a *write* operation.

2.5.2 CONVEX Extensions

fcntl(): CONVEX adds the following *cmd* values as extensions:

_F_GETOWN	Get the process ID or process group currently receiving SIGIO and SIGURG signals; process groups are returned as negative values.
_F_SETOWN	Set the process or process group to receive _SIGIO and _SIGURG signals; process groups are specified by supplying *arg* as negative; otherwise, *arg* is interpreted as a process ID.
_F_SETBLKSIZE	Set the physical record size for block tape operations to the value of *arg*. This value defaults to 65536 when the file is opened.

CONVEX adds the following file status flags as extensions:

_FASIO	Use daemon processes to accomplish asynchronous I/O. Subsequent operations on this *fd* cause the user process to proceed in parallel with the I/O transfers. Several I/O transfers may be proceeding in parallel to or from the same file (*fd*), each under the control of a separate daemon process, limited only by a system-wide

configuration maximum (`ps -axl` displays daemon processes as *asiodaemon*). Synchronization may be accomplished with the *asiostat*, *select*, or *fstat* system calls, or by the use of _FASYNC.

It is assumed that all requested I/O will be performed; the *read* or *write* system calls will return as if all data had been transferred, even though the transfer may be impossible. To determine results, the *asiostat* system call will return the sum of the number of bytes transferred on all asynchronous I/O requests on a given file descriptor since the last *asiostat* call. The _FASIO property may be inherited by forked children, and it is illegal to set _FASIO on socket descriptors.

Asynchronous I/O seems to help tape performance, but due to the existence of the buffer cache, read ahead, and write behind, sequential file transfer is already quite asynchronous, and no performance improvement may be noted. The *brk()*, *_exit()*, and *fork()* system calls will wait for all asynchronous I/O invoked by the calling process to be completed before being processed. The *asiostat()*, *close()*, *fcntl()*, *flock()*, *fstat()*, *fsync()*, *ftruncate()*, and *ioctl()* system calls will wait for all asynchronous I/O on the specified file descriptor (*fd*) to be completed before being processed.

_FASYNC

If the _FASIO bit is set, a _SIGIO signal will be sent to a process when all its outstanding asynchronous I/O (using daemon processes) has been completed. If the _FASIO bit is not set, _FASYNC enables the _SIGIO signal to be sent to the process group when I/O is possible, for example, upon availability of data to be read. (This latter facility is available only on tty operations.)

_FNCACHE

This bit requests the kernel to bypass the incore buffer cache and perform I/O transfers directly to/from user address space. As a side effect, file system read ahead and write behind are disabled. For files that are accessed sequentially, the net result is usually lost performance. _FNCACHE might be helpful in situations where the file in question is accessed randomly, but you should test results with and without this option set before using it. The operating system cannot bypass the buffer cache if transfers are not aligned on the file's blocksize boundaries because disk controllers cannot start transferring in the middle of sectors. The *fstat()* system call returns the proper blocksize as *st_blksize*. Transfers should start (by seeking if necessary) at integral multiples of *st_blksize*.

read(): *read()* may operate synchronously (default) or asynchronously, depending on the _FASIO flag set with the *fcntl()* system call. Synchronous reads suspend the caller until the system has processed the read request and return the number of bytes read and placed in the buffer or 0 if end-of-file has been reached. The file position pointer is incremented by the number of bytes read. Asynchronous reads return before the request has been fully processed, and unless there are errors in the arguments passed, always return the number of bytes requested, whether or not it is possible to complete the request. The file position pointer is incremented by the number of bytes requested. To determine what happened during asynchronous transfers, refer to *asiostat()*.

write(): *write()* may operate synchronously (default) or asynchronously, depending on the _FASIO flag set with the *fcntl()* system call. Synchronous writes suspend the caller until the system has processed the write request, and return the number of bytes written. The file position pointer is incremented by the number of bytes written. Asynchronous writes return before the request has been fully processed, and unless there are errors in the arguments passed, always return the number of bytes requested, whether or not it is possible to

complete the request. The file position pointer is incremented by the number of bytes requested. To determine what happened during asynchronous transfers, refer to *asiostat()*.

2.6 Device- and Class-Specific Functions

2.6.1 Terminal Attributes

Version 7.1, and earlier, of the operating system used *ioctl()* to get and set terminal attributes. With POSIX.1, terminal attributes are stored in a *termios* structure and can be set and retrieved with *tcsetattr()* and *tcgetattr()*.

2.6.1.1 Window Size

These functions also set and get window-size parameters. If the SETWINSIZE bit is set in the *c_lflags* field of the *termios* structure and passed to *tcsetattr()*, the window-size field in the *termios* structure becomes the new window size (and a SIGWINCH signal is sent if appropriate).

2.6.1.2 Raw Mode

To go to raw mode, pre-POSIX applications used the following:

```
ioctl(fd, TIOCGETP, &sgttyb);
sgttyb.sg_flags |= RAW;
ioctl(fd, TIOCSETP, &sgttyb);
```

Applications under ConvexOS V8.0 now use:

```
tcgetattr(fd, &termios);
termios.c_iflag &= ~(IGNBRK|BRKINT|IGNPAR|PARMRK|INPCK|ISTRIP|
                                    INLCR|IGNCR|ICRNL|IXON|IXANY|IMAXBEL);
termios.c_oflag &= ~OPOST;
termios.c_cflag &= ~(PARENB|CSIZE);
termios.c_cflag |= CS8;
termios.c_lflag &= ~(ECHOCTL|ISIG|ICANON);
termios.c_cc[VMIN] = 1;
termios.c_cc[VTIME] = 0;
tcsetattr(fd, TCSANOW, &termios);
```

2.6.1.3 Special Characters

Individual special characters can be disabled by setting them to the value of the DISABLE special character. Previous versions disabled special characters by setting them to the value '\377'. Previous versions of the operating system used the following:

```
ioctl(fd, TIOCGETP, &sgttyb);
sgttyb.sg_intrc = '\377';
ioctl(fd, TIOCSETP, &sgttyb);
```

ConvexOS V8.0 uses:

```
tcgetattr(fd, &termios);
termios.c_cc[VINTR] = termios.c_cc[VDISABLE];
```

```
tcsetattr(fd, TCSANOW, &termios);
```

2.6.2 *termios* Flags

2.6.2.1 *c_iflag* Input Flags

IGNBRK — Ignore break. If a break condition is detected on the line, it is completely ignored and invisible to the application.

BRKINT — Map a break to an interrupt. If IGNBRK is not set and a break condition is detected on the line, a SIGINT is sent to the foreground process group of the tty.

IGNPAR — Ignore parity errors. If a parity or framing error is detected, it is completely ignored and invisible to the application.

PARMRK — Mark parity and framing errors. If a break condition is detected and both IGNBRK and BRKINT are not set, the sequence '\0377', '\0', '\0' is placed in the input stream. If a parity or framing error is detected and IGNPAR is not set, the sequence '\0377', '\0', c is placed in the input stream. No PARMRK would generate a '\0' when parity or framing errors occur.

Also when PARMRK is set and IGNPAR and ISTRIP are not set, a valid '\0377' is read as '\0377', '\0377'.

INPCK — Enable input parity checking. If set, characters with parity errors are ignored (if IGNPAR is set), read as '\0377', '\0377', c (if PARMRK is set), or read as '\0'. If INPCK is not set, parity errors are ignored and the character is placed in the input stream as received.

ISTRIP — Input bytes are stripped to seven bits. This occurs after parity checking but before any special character processing.

INLCR — Translate input \n to \r. If \n is input and INLCR is set, \r is read independent of the setting of IGNCR.

IGNCR — Ignore \r on input. This is independent of the setting of ICRNL (that is, \r is ignored before translated to \n).

ICRNL — Translate input \r to \n. This occurs only if IGNCR is not set.

In the previous tty driver, this occurred on read if CRMOD was set and RAW was not.

IXON — Start/stop output control is enabled. A stop character on input causes output to stop. A start character on input causes output to resume. If start and stop characters are the same, they toggle output.

IFLOW — Same as IXON.

IXOFF — When input queue approaches a certain limit, the stop character is sent to the device to stop it from sending input. A start character is sent when the input queue drains below a certain limit. Do not send the stop character if it is VDISABLEd.

This is equivalent to the TANDEM bit in the previous tty driver.

ITANDEM — Same as IXOFF.

IXANY — Allow any character to restart output after stop. If not set, all characters are dropped until a start character is seen. If the start character is equal to the stop character, any character will restart after stop.

This is equivalent to the DECCTQ bit in the previous tty driver.

IEXTEN — Turn on extensions. Interpret non-POSIX control characters: VDSUSP, VEOL2, VQUOTE, VERASE2, VLNEXT, VFLUSHO, VWERASE, VREPRINT.

IMAXBEL — When set, a CTRL-G is sent to the terminal when the input queue is full. If not set, input and output queues are silently flushed when the input queue overflows. In either case the input character is ignored.

The previous tty driver sent CTRL-G only if using the new tty line discipline; it always dropped the character. It never flushed the queues.

2.6.2.2 *c_oflag* Output Flags

OPOST — Perform output processing. The values of the other flag bits are checked only if OPOST is set.

In the previous tty driver, this behavior was controlled by either the RAW or LITOUT bits.

ONLCR — Translate \n to \r\n on output.

In the previous tty driver, this happened if CRMOD was set.

ONLCRNL — Same as ONLCR.

OXTABS — Expand tabs to spaces. If a \t is printed, send the proper number of spaces to the terminal to move the cursor to the next tab stop.

ONOEOT — Do not deliver a \04 to the terminal. It is silently discarded.

In the previous tty driver, this was true if RAW, LITOUT, and CBREAK were not set.

2.6.2.3 *c_cflag* Control Flags

CSIZE — Character size mask:

- CS5 Characters have 5 data bits
- CS6 Characters have 6 data bits
- CS7 Characters have 7 data bits
- CS8 Characters have 8 data bits.

The previous tty driver determined character size this way:

1. If B134: 6 bits, odd parity

2. else if RAW mode: 8 bits, no parity
3. else if EVENP set: 7 bits, even parity
4. else if ODDP set: 7 bits, odd parity
5. else no parity or both: 8 bits, no parity.

CSTOPB — If set, use two stop bits. Otherwise, use one.

The previous tty driver used two stop bits for 110 baud. Otherwise, it used one.

CREAD — Receiver is enabled. No characters are read if this is not set.

PARENB — Output parity is enabled.

PARODD — Use odd parity. Only used if PARENB is set.

HUPCL — Hang up on last close; drop DTR on device. Set with TIOCHUPCL *ioctl*.

CLOCAL — Ignore modem status lines. Do not wait for carrier present before I/O.

2.6.2.4 *c_lflag* Local Flags

ECHO — Echo input to the terminal.

ECHOKE — If set when a kill character is typed, visually erase the line with the string "\b \b" for each character. If either ECHOPRT is set or output has been interspersed with recent unread input, ECHOKE behaves exactly as ECHOK.

The previous tty driver used the CRTKIL bit.

ECHOE — If ECHOE is set, visually erase characters with the string "\b \b." If not set (and ECHOPRT is not set), simply echo the erase character when it is typed.

ECHOK — If ECHOKE is not set when a kill character is typed, the kill character is echoed and \n is printed to the screen.

ECHONL — Echo \n when input even if ECHO is off.

ECHOPRT — Visual erase mode for hardcopy terminals. If ECHOE is not set and ECHOPRT is set, consecutive erased characters are erased visually within \ and /.

This was PRTERA in the previous tty driver.

ECHOCTL — Echo control characters as "^c".

This was CTLECH in the previous tty driver.

ISIG — If set, each input character is checked against INTR, QUIT, SUSP, and DSUSP. If the input character matches one of these special control characters, the function associated with that character is performed.

ICANON — Enable canonical mode input processing. Characters are assembled into lines, and line editing is performed.

If not set, non-canonical mode input takes place with at least MIN bytes

received or TIME timeout value expires before a read is satisfied.

ALTWERASE — Use the alternate word erase algorithm for CTRL-W. If set, words consist of adjacent alphanumeric and underscore characters or adjacent non-alphanumeric and underscore characters.

If not set, the words are erased as white space delimited entities.

IEXTEN — Enable extended functions. Functions controlled by IEXTEN: EOL2, LNEXT, FLUSHO, WERASE, REPRINT, ERASE2, and DSUSP and the ability of \ to quote ERASE, ERASE2, or KILL.

ECHO — If set, characters are echoed to the terminal as they are typed.

TOSTOP — If set, processes receive a SIGTTOU when attempting to generate output while not in the foreground process group.

NOFLSH — If set, do not flush pending input and output on receipt of a signal. If not set, INTR and QUIT cause input and output to be flushed and a SUSP causes only pending input to be flushed.

2.6.3 Special Characters

EOF — End of file. Causes a break in input.

EOL — Extra break-in-input character, like old t_brkc.

EOL2 — Extra EOL character. Only valid if IEXTEN is set in *lflag*.

ERASE — Erase character char.

ERASE2 — Extra ERASE character. Only valid if IEXTEN set in *lflag*.

WERASE — Word erase character.

KILL — Line kill character.

REPRINT — Line reprint character.

INTR — Interrupt character. Generate SIGINT on input.

QUIT — Quit character. Generate SIGQUIT on input.

SUSP — Suspend character. Generate SIGTSTP on input.

DSUSP — Delayed-suspend character. Generate SIGTSTP when read.

START — Start character. Start IXON flow-controlled output.

STOP — Stop character. Stop IXON flow-controlled output.

LNEXT — Literal-next character. Next character is literal input; no special meaning.

FLUSHO — Flush-output character. Causes output to be flushed; nothing delivered.

MIN	Minimum characters for non-canonical read.
TIME	Read timeout (in tenths of seconds) for non-canonical read.
QUOTE	Quote character. Used to quote erase/kill characters on input.
DISABLE	Disable character. Set others to this to disable special processing.

Chapter 3
POSIX and Non-POSIX Behaviors

3.1 Introduction

It is possible for a POSIX application to interact with a non-POSIX application. For example, you could have information from V7.1 of the operating system link with data in ConvexOS V8.0. Areas of interaction would most likely be in a parent/child relation or a task-to-task communication such as signal handling. Although this interaction is outside the scope of POSIX.1, this chapter discusses some implementations chosen by CONVEX.

3.2 Behaviors

Certain system calls behave differently depending on whether POSIX or non-POSIX processes are involved. The following sections explain the behavior of these functions:

- *fork()*
- *kill()*
- *exit()*.

For more information on signal and process handling, refer to Chapter 3, "Process Primitives," in the *CONVEX POSIX Conformance* document and IEEE Std 1003.1-1988.

3.2.1 *fork()*

When a signal is delivered to a process and the process has the signal blocked with *sigblock*(2) or *sigsetmask*(2), the signal delivery is pended until the process unblocks the signal. In V7.1, and earlier, of the operating system, when a new process was created using *fork()*, pending signals were not propagated to the new process. That behavior is still true for non-POSIX processes that *fork()* while signals are pending. When a POSIX process *fork()*s, signals pended for delivery to the parent process are also propagated as pended signals for the child process.

3.2.2 *kill()*

kill() has some special cases that allow the sender to send a signal to an entire process group. In these cases, the signal is sent to all processes in the process group (both POSIX and non-POSIX).

A SIGCONT may be sent from a non-POSIX process to any descendent (POSIX or non-POSIX) even if the receiving process has changed sessions and *setuid()*. A POSIX process may not send a SIGCONT to a descendent in the case where the descendent has changed both sessions and *setuid()*.

3.2.3 *exit()*

If the existing process is a POSIX session leader, a SIGHUP is sent to each process in the foreground process group of the exiting process' controlling terminal. If the exiting process is a POSIX group leader (but not a session leader) and any other member of the process group is STOPPED, SIGHUP and SIGCONT are sent to each process in the process group. If the exiting process is a non-POSIX group leader, SIGHUP is sent only to STOPPED children. In all of these cases, when a signal is sent to all processes in a process group, the signal is sent to both POSIX and non-POSIX processes.

Chapter 4
POSIX and Languages

POSIX.1 is currently defined in terms of a C programming language interface to the operating system, though interfaces for Ada and FORTRAN will soon be defined.

4.1 POSIX and C

The C language was created for use on the UNIX operating system as a general-purpose programming language. Up to a point, applications written in C are portable between computer systems. Programs that utilize only high-level features of C are easier to port than programs that use assembly-language features. But the original definition of C left areas of the language undefined.

To increase the portability of applications, clarify some ambiguities, and set down formal requirements for all aspects of the language, the American National Standards Institute (ANSI) began formulating a standard C language that dealt with the preprocessor, libraries, and the language itself.

Benefits of ANSI C are:

- Ease of maintenance
- Increase in efficiency
- Provide a new way to declare functions that helps reduce errors by allowing type checking between the call and declaration
- Increase in reliability of C applications
- Require that parentheses enforce the order of evaluation of expressions
- Allow greater optimization because of new rules for floating-point expression evaluation and aliasing.

For complete information on ANSI C and enhancements, refer to the *CONVEX ANSI C Concepts* document.

4.1.1 Compilers

There are three C compilers:

- Common C (previously called portable C)
- CONVEX C V3.0 (released just prior to CONVEX C V4.0)
- CONVEX C V4.0.

The common C compiler was installed as */bin/cc* on all CONVEX computers. It is now installed as */bin/pcc*. It performed very few optimizations on a program. Most applications that invoked

this compiler may still be compiled with the *-pcc* option of CONVEX C V4.0.

The CONVEX C V3.0 compiler was an optional product requiring a license. It was capable of vector and parallel optimizations. It is available only on systems that had it before CONVEX C V4.0.

There are two versions of the CONVEX C V4.0 compiler: one capable of only scalar optimizations and another that performs vector and parallel optimizations. While the former is available on all CONVEX machines, the latter is an optional product available to those who purchase or have a CONVEX C license. This compiler is invoked with the *cc* command name.

CONVEX will offer the enhanced version of ANSI C with its release of CONVEX C V4.0. At that time, and not before, ConvexOS V8.0 will be POSIX.1 compliant.

4.1.2 Compatibility Modes

The CONVEX C V4.0 compiler, which conforms to the ANSI C standard, offers four modes of compatibility that provide different language features, libraries, and include file contents:

- Default
- Conforming
- Strict
- Backward Compatible.

The default mode accepts programs written in ANSI C with POSIX.1 functions and CONVEX extensions. POSIX.1 defines the functions that a C program can use to interface with ConvexOS. These functions are listed in Appendix A.

The conforming mode provides specifications of ANSI C and access to POSIX.1 functions. However, no CONVEX extensions are permitted.

The strict mode of the compiler accepts applications that use only ANSI C language features and functions. Neither POSIX.1 functions nor CONVEX extensions are supported.

The backward-compatible mode accepts no ANSI C features or POSIX.1 functions. It is compatible with the common C compiler.

Backward-compatible mode, common C compiler, and implementation-defined features of CONVEX C V4.0 are detailed in the *CONVEX ANSI C Concepts* document.

4.2 POSIX and Libraries

The four compatibility modes also select libraries that define system functions for:

- CONVEX extensions
- POSIX.1 functions
- ANSI C language specifications
- Backward-compatible features.

The following options allow you to select a compiler mode and/or link with the specified libraries:

default	CONVEX extensions, POSIX.1, and ANSI C libraries
-std	POSIX.1 and ANSI C libraries
-str	ANSI C library
-pcc	Backward-compatible library

These libraries are searched automatically in the correct order for object code that will be linked into the executable program.

Shipped with CONVEX C V4.0 are include files that can be interpreted in several ways when a program is compiled. By default, include files are interpreted to define POSIX functions, ANSI C features, and CONVEX extensions. By specifically defining certain conditional compilation symbols, various features can be defined by include files. These symbols can be defined by any of these equivalent methods:

- #define directive in the program text
- -D option on the command line
- -D option in the CCOPTIONS environment variable.

Defining the symbol _POSIX_SOURCE alone causes the include files to be interpreted to define POSIX.1 and ANSI C features. The symbol _CONVEX_SOURCE may be additionally defined for CONVEX extensions *only if* the _POSIX_SOURCE symbol is defined. You cannot define _CONVEX_SOURCE if _POSIX_SOURCE has not been defined. The compiler, in the default mode, automatically defines both _POSIX_SOURCE and _CONVEX_SOURCE for you.

When both _CONVEX_SOURCE and _POSIX_SOURCE are defined, include files are interpreted to define all ANSI C features, POSIX.1 functions, and CONVEX extensions.

For compiler usage and examples, refer to the *CONVEX C User's Guide*.

4.3 POSIX and Ada

After IEEE Std 1003.5 has been completed and ratified, CONVEX intends the next major release of the Ada compiler to be POSIX compliant.

4.4 POSIX and FORTRAN

After IEEE Std 1003.9 has been completed and ratified, CONVEX intends the next major release of the FORTRAN compiler to be POSIX compliant.

Chapter 5
Creating a POSIX Application

5.1 *nu*

This is an example of changes needed for an application that involves the group and password databases. Minimum requirements are specified in Chapter 9, "System Databases," of POSIX.1. The appropriate modifications appear in boldface type. The example is an interactive-only version of *nu* and does not include issues involving:

- CONVEX Share Scheduler
- Password aging
- Disk quotas.

5.1.1 Example

```
#ifndef lint
static char copyr[] = ''Copyright 1989 Convex Computer Corp.'';
#endif

/*
*
* nu --- add a new user to the system.
*
*/

#ifdef _POSIX_SOURCE
#include <sys/types.h>
#endif

#include <ctype.h>
#include <grp.h>
#include <pwd.h>
#include <signal.h>
#include <stdio.h>
#include <strings.h>
#include <sysexits.h>
#include <sys/file.h>
#include <errno.h>
#include "nu.h"

#ifdef _POSIX_SOURCE
  pid_t         uid;                    /* user id */
  pid_t         ruid;                   /* id from uidcount */
  gid_t         gid;                    /* group id */
#else
  int           uid;                    /* user id */
  int           ruid;                   /* id from uidcount */
  int           gid;                    /* group id */
```

```
#endif

int             i;                          /* temporary */
char            buf[BUFSIZ];                      /* temporary */
char            cmd[BUFSIZ];                      /* UNIX command string */
char           *PgmName;                    /* name of this program */
char           *nurcfile    = DEFNURC;     /* name of defaults file */
FILE           *lfd         = NULL;               /* lock file descriptor */
struct group *grp;
struct passwd      *pwd;

void            disable_xrpt();             /* internal procedure */
void            enable_xrpt();                    /* internal procedure */
char           *skipblks();                       /* internal procedure */

#ifndef _POSIX_SOURCE
 char           *sprintf();                 /* sprintf(3S) */
 char           *crypt();                   /* crypt(3) */
#endif

extern int   errno;

main(argc, argv)
      int   argc;
      char  **argv;
{
      int   fd_passwd;                      /* open /etc/passwd */

      if ( PgmName = rindex(argv[0], '/') )
            ++PgmName;
      else
            PgmName = argv[0];

      /* must be the super-user to do this */
      if (geteuid()) {
            error("not super-user\n");
            exit(EX_NOPERM);
      }

      /* establish signal handlers */
      enable_xrpt();

      /* first, process the arguments */
      while (--argc) {
        argv++;
        if (argv[0][0] == '-') {
            switch (argv[0][1]) {
            case 'n':
              if (--argc)
                  nurcfile = *++argv;
              else {
                  error ("missing argument for -n\n");
                  exit (EX_USAGE);
              }
              break;
            default:
              error ("invalid option %s\n", *argv);
              exit (EX_USAGE);
            }
```

```
    }
    else {
        error ("invalid argument %s\n", *argv);
        exit (EX_USAGE);
    }
}

/* let's see if we are the only one running */
lfd = fopen(NULCKFILE, "r");          /* see if the lockfile exists */
if (lfd == NULL) {                    /* nope, create it */
        lfd = fopen(NULCKFILE, "w");
        if (lfd == NULL) {            /* couldn't do it */
                error("cannot create lockfile %s\n", NULCKFILE);
                exit(EX_TEMPFAIL);
        }
}
i = flock(fileno(lfd), LOCK_EX);      /* try to lock it */
if (i != 0) {                         /* didn't work */
    error ("cannot establish lockfile %s\n", NULCKFILE);
    (void)fclose (lfd);
    exit(EX_UNAVAILABLE);
}

/* lock the password file for nu's use */
fd_passwd = open(PWTLCKFILE, O_WRONLY|O_CREAT|O_EXCL, 0644);
if (fd_passwd < 0) {
    if (errno == EEXIST) {
        error ("password file busy\n");
        (void)fclose (lfd);
        exit(EX_UNAVAILABLE);
    }
    fprintf(stderr, "%s: ", PgmName); perror(PWTLCKFILE);
    (void)fclose (lfd);
    exit(EX_UNAVAILABLE);
}

/*
 * read the default data from nurc file
 * if it does not exist, then use the hardwired defaults of nu.
 */
getdefaults();                        /* initialize default values */

bcopy (defaults, newuser, MAXKEY*sizeof(struct keyword));
/* ask for the login name */
while (TRUE) {
        if (! getresponse("Login name: ", newuser[LOGIN].value))
        continue;
        if (newuser[LOGIN].value[0]) {
        if (getpwnam(newuser[LOGIN].value))
                fprintf (stderr,
                "Login %s already exists in /etc/passwd\n\n",
                newuser[LOGIN].value);
        else if (strlen(newuser[LOGIN].value) > 8) {
          fprintf (stderr, "Login name %s ",newuser[LOGIN].value);
          fprintf (stderr, "longer than eight characters\n");
        }
        else
                break;
        }
```

```
        else
                fprintf (stderr, "You must enter a login name\n");
        }


        uid = getnewuid();

        /* set up default group ID, if they specified one */
        gid = -1;
        if (newuser[GID].value[0]) {    /* see if set GID value */
#ifdef _POSIX_SOURCE
                if (grp = getgrgid((gid_t)atoi(newuser[GID].value)))
#else
                if (grp = getgrgid(atoi(newuser[GID].value)))
#endif
                gid = grp->gr_gid;
        }
        else if (grp = getgrnam(newuser[GROUP].value))
                gid = grp->gr_gid;

        /* set up default home directory */
        if (!newuser[HOMEDIR].value[0])
                (void) sprintf (newuser[HOMEDIR].value, "%s/%s",
                        newuser[DIR].value, newuser[LOGIN].value);

        /* ask for the user id */
        do {
                if (uid < 0)
                (void)sprintf(buf, "User id: ");
                else
                (void)sprintf(buf, "User id [%d]: ", uid);
                if (! getresponse(buf, buf))
                continue;
                if (buf[0])
#ifdef _POSIX_SOURCE
                uid = (pid_t)atoi(buf);
#else
                uid = atoi(buf);
#endif
                else {
                        if (uid < 0) {
                          fprintf(stderr, "You must enter a user id\n\n");
                          continue;
                        }
                }
                pwd = getpwuid(uid);
                if (pwd) {
                        fprintf(stderr, "User id %d already exists\n\n", uid);
                        uid = -1;
                        continue;
                }
        } while (uid < 0);


        /* now the group id */
        while (TRUE) {
                if (gid < 0)
                (void)sprintf(buf, "Group id: ");
                else
```

```
            (void)sprintf(buf, "Group id [%s]: ", grp->gr_name);
            if (! getresponse(buf, buf))
            continue;
            if (buf[0]) {
              if (isdigit(buf[0])) {
#ifdef _POSIX_SOURCE
                    gid = (gid_t)atoi(buf);
#else
                    gid = atoi(buf);
#endif
                    /* make sure valid --gid's must be "short" */
                    /* in some places (e.g., proc.h)       */
                    if (gid < 0 || gid > MAXGID) {
                      fprintf (stderr,
                          "Group ID should be between 0 and %d\n\n",
                          MAXGID);
                      if (! grp) gid = -1;
                      continue;
                    }
                    grp = getgrgid(gid);
              }
              else
                    grp = getgrnam(buf);
              if (! grp) {
                    fprintf (stderr, "Group %s does not exist\n\n", buf);
                    gid = -1;
                    continue;
              }
              gid = grp->gr_gid;
              break;
            }
            else {
                    if (gid < 0) {
                      fprintf (stderr, "You must enter a group id\n\n");
                      continue;
                    } else break;
            }
      }
      (void)strcpy(newuser[GROUP].value, grp->gr_name);


      /* and the home directory */
      while (TRUE) {
            if (!newuser[HOMEDIR].value[0])
              (void)sprintf(buf, "Home directory: ");
            else
              (void)sprintf(buf, "Home directory [%s]: ",
                          newuser[HOMEDIR].value);
            if (! getresponse(buf, buf))
              continue;
            if (buf[0])
              (void)strcpy(newuser[HOMEDIR].value, buf);
            else {
              if (!newuser[HOMEDIR].value[0]) {
                    fprintf (stderr, "You must enter a home directory\n\n");
                    continue;
              }
            }
            if (chdir(newuser[HOMEDIR].value) == 0) {
```

```
        fprintf(stderr, "Directory %s already exists\n\n",
                newuser[HOMEDIR].value);
              newuser[HOMEDIR].value[0] = '\0';
              continue;
            }
            else
            break;
      }

      /* and the home directory protection */
      while (TRUE) {
            if (!newuser[PROT].value[0])
              (void)sprintf(buf, "Home directory protection: ");
            else
              (void)sprintf(buf, "Home directory protection [%s]: ",
                          newuser[PROT].value);
            if (! getresponse(buf, buf))
              continue;
            if (buf[0])
              (void)strcpy(newuser[PROT].value, buf);
            else {
              if (!newuser[PROT].value[0]) {
                   fprintf (stderr, "You must enter the protection for the
                   home directory\n\n");
                   continue;
              }
            }
            if (!goodmode(newuser[PROT].value)) {
              fprintf (stderr,
                   "%s is not a valid mode\n\n", newuser[PROT].value);
              newuser[PROT].value[0] = '\0';
              continue;
            }
            else
            break;
      }

      /* prompt for the shell */
      while (TRUE) {
            if (!newuser[SHELL].value[0])
              (void)sprintf(buf, "Login shell: ");
            else
              (void)sprintf(buf, "Login shell [%s]: ", newuser[SHELL].value);
            if (! getresponse(buf, buf))
              continue;
            if (buf[0])
              (void)strcpy(newuser[SHELL].value, buf);
            else {
              if (!newuser[SHELL].value[0]) {
                   fprintf (stderr, "You must enter a login shell\n\n");
                   continue;
              }
            }
            if (access(newuser[SHELL].value, X_OK)) {
              fprintf (stderr,
                   "File %s does not exist/is not executable\n\n",
                   newuser[SHELL].value);
              newuser[SHELL].value[0] = '\0';
              continue;
```

```
        }
        else
        break;
}


addpwent("*", uid, gid);

/* unlock the temp files */
(void) unlink(PWTLCKFILE);      /* unlink ptmp so chfn will work */
(void) unlink(PWRTLCKFILE);     /* unlink rtmp */

disable_xrpt();

/* change the user data to something intelligible */
fprintf(stdout, "\n");
fprintf(stdout, "Changing the user information...\n");
(void)sprintf(cmd, "%s %s", CHFN, newuser[LOGIN].value);
if (system(cmd)) {
        error ("cannot invoke %s -- change user info later\n", CHFN);
}

enable_xrpt();

/* see if there are type restrictions */

/* typed? */
fprintf(stdout, "\n");
while (TRUE) {
        if (!newuser[TYPED].value[0])
          (void)sprintf(buf, "Password to be typed: ");
        else
          (void)sprintf(buf, "Password to be typed [%s]: ",
                        newuser[TYPED].value);
        if (! getresponse(buf, buf))
          continue;
        if (buf[0])
          (void)strcpy(newuser[TYPED].value, buf);
        if (strcmp(newuser[TYPED].value, ON) &&
                strcmp(newuser[TYPED].value, OFF)) {
          fprintf (stderr, "You must answer 'Y' or 'N'\n");
          newuser[TYPED].value[0] = '\0';
          continue;
        }
        else
        break;
}

/* give the user an initial password */
fprintf(stdout, "\n");
fprintf(stdout, "Entering the user password...\n");
(void) sprintf(cmd, "%s %s", PASSWD, newuser[LOGIN].value);
for (;;) {
        int stat;

        stat = system(cmd) >> 8;
        if (stat == 0)                  /* successful */
        break;
        else if (stat == 2)         /* password mismatch */
```

```
        fprintf (stdout, "Try again...\n");
              else {
                fprintf (stdout, "%s failed (returned status %d)\n",
                     PASSWD, stat);
                fprintf (stdout, "WARNING: No password set for %s\n",
                     newuser[LOGIN].value);
                break;
              }
        }

        /* create his/her directory */
        fprintf(stdout, "\n");
        fprintf(stdout, "Creating the home directory...\n");
        (void) setupdir();

        /*
         * clean up time - get rid of the lock file, write back a new uid
         * if the uidcount file was used, etc, and exit
         */
        if (ruid == uid)
              (void) write_newuid(ruid + 1);

        fputs( "Rebuilding passwd database...\n", stderr );
        if ( makedb( PASSWDFILE) < 0 ) {
              fprintf( "nu: mkpasswd failed\n", stderr );
              cleanup( 1 );
        }

        cleanup (EX_OK);
}

static
makedb( pwfile)
        char *pwfile;
{
        int status, pid, w;

        if (!(pid = vfork())) {
              execl( "/etc/mkpasswd", "mkpasswd", "-p", pwfile, 0 );
              _exit(127);
        }
        while ((w = wait(&status)) != pid && w != -1);
        if (w == -1 || status)
              return(-1);
        return(0);
}

/* cleanup --- clean things up after an interrupt */
cleanup(exit_stat)
int exit_stat;
{
        if (lfd) {
          (void)fclose(lfd);
          (void)unlink(NULCKFILE);
          (void)unlink(PWTLCKFILE);
          (void)unlink(PWRTLCKFILE);
        }
        (void)endpwent();
        exit(exit_stat);
```

```
}

/* disable_xrpt --- disable common interrupts during critical section */
void
disable_xrpt()
{
        (void)signal(SIGHUP, SIG_IGN);
        (void)signal(SIGINT, SIG_IGN);
        (void)signal(SIGQUIT, SIG_IGN);
        return;
}

/* enable_xrpt --- enable trapping of common interrupts */
void
enable_xrpt()
{
        (void)signal(SIGHUP, cleanup);
        (void)signal(SIGINT, cleanup);
        (void)signal(SIGQUIT, cleanup);
        return;
}

/* fixup --- perform smart copy of str2 into str1 */
fixup(str1, str2)
char *str1, *str2;
{
        if (index(str2, ':')) {
                fprintf(stderr, "Illegal character \":\" in string\n");
                return(FALSE);
        }
        while ((*str2 == ' ') || (*str2 == '\t'))
                ++str2;
        while (*str2 && *str2 != '\n')
                *str1++ = *str2++;
        *str1 = '\0';
        return(TRUE);
}

getnewuid()
{
        FILE  *ufid;
        FILE  *fopen();

        if (!strcmp(newuser[NOUID].value,ON))
          return (getnextuid()); /* calculate from passwd file */
        if ((ufid = fopen(UIDFILE, "r")) == NULL) {
          fprintf(stderr, "%s: cannot open %s for input\n", PgmName, UIDFILE);
          exit(EX_OSFILE);
        }
        (void) fscanf(ufid, "%d", &ruid);
        (void) fclose(ufid);
        return(ruid);
}

write_newuid(uid)
#ifdef _POSIX_SOURCE
        pid_t uid;
#else
        int uid;
```

```
#endif
{
        FILE *ufid;
        FILE *fopen();

        if ((ufid = fopen(UIDFILE, "w")) == NULL) {
          fprintf(stderr,"%s: cannot open %s for output\n", PgmName, UIDFILE);
          exit(EX_OSFILE);
        }
        fprintf(ufid, "%d\n", uid);
        (void) fclose(ufid);
}

/*
 * getnextuid - search thru the /etc/passwd file, and find the
 *           highest used user ID, returning that number + 1.
 */
getnextuid()
{
        int maxuid = 0;
        struct passwd *pwent;

        (void) setpwent();
        while (pwent = getpwent()) {
          if ((pwent->pw_name[0] == '+') || (pwent->pw_name[0] == '-'))
              continue;          /* skip YellowPages-type entries */
          if (pwent->pw_uid > maxuid)
              maxuid = pwent->pw_uid;
        }
        return (maxuid + 1);
}

/* getresponse --- get/fix a response from the user */
getresponse(prompt, reply)
char *prompt, *reply;
{
        char buf[ARB];

        fprintf(stdout, prompt);
        (void)fflush(stdout);
        (void)fgets(buf, sizeof(buf), stdin);
        return(fixup(reply, buf));
}

goodmode(protection)
char *protection;
{
        register char *p = protection;

        for (; *p; ++p) {
          if (!isxdigit(*p))
              return(0);
        }
        return(1);
}

getdefaults ()
{
        FILE  *fp_defs;
```

```
        char   *end;        /* pointer to newline, if any, in line */
        char    line[ARB];

        if ((fp_defs = fopen(nurcfile, "r")) != NULL) {
          while (fgets(line, ARB, fp_defs) != NULL) {
              if (end = rindex (line, '\n'))
                *end = '\0';
              findkey(line, defaults, ':');
          }
        }
        else if (strcmp(nurcfile, DEFNURC)) {
          /* only make it an error if they gave a specific nurc file */
          error ("could not open nurc file to read defaults\n");
          cleanup (EX_NOINPUT);
        }

        /*
         * If they give a default GID, convert it to a default GROUP name,
         * so that detecting a changed value (of GID or GROUP) from within
         * a batch file will be possible.
         */
        if (defaults[GID].value[0]) {
#ifdef _POSIX_SOURCE
          if (grp = getgrgid((gid_t)atoi(defaults[GID].value)))
#else
          if (grp = getgrgid(atoi(defaults[GID].value)))
#endif
              (void) strcpy (defaults[GROUP].value, grp->gr_name);
          else
              error("default group id (%s) invalid; using group %s instead\n",
                  defaults[GID].value, defaults[GROUP].value);
          defaults[GID].value[0] = '\0';
        }
}

findkey (line, keywords, sepch)
char   *line;
struct      keyword keywords[];
char   sepch;              /* separator char between keyword & value */
{
        char   *valptr;
        int    i;          /* index thru keyword array */

        for (i = 0; i < MAXKEY; i++)
          if (!strncmp(line, keywords[i].name, strlen(keywords[i].name)))
              break;              /* found it */
        if (i < MAXKEY)     {      /* found a match */
          if ((valptr = index (line, sepch)) == NULL)
              (void)strcpy(keywords[i].value, ON); /* just boolean value */
          else              /* remove leading blanks */
              (void)strcpy(keywords[i].value, skipblks(valptr+1));
        }
        else                /* couldn't find that keyword in array */
          error ("unknown field specifier '%s' ignored\n", line);
}

char *
skipblks (cp)
char   *cp;
```

```
{
        while (cp && *cp == ' ')
          cp++;
        return (cp);
}

addpwent(password, uid, gid)
char   *password;
#ifdef _POSIX_SOURCE
        pid_t   uid;
        gid_t   gid;
#else
        int     uid, gid;
#endif
{
        char    pentry[BUFSIZ];         /* password file entry */
        FILE   *pfd   = NULL;                   /* password file descriptor */

        /* make the passwd entry */
        (void)sprintf(pentry, "%s:%s:%d:%d:%s,%s,%s,%s:%s:%s\n",
          newuser[LOGIN].value, password, (int)uid, (int)gid,
          newuser[NAME].value, newuser[OFFICE].value, newuser[EXT].value,
          newuser[HOMEPH].value, newuser[HOMEDIR].value,newuser[SHELL].value);

        disable_xrpt();

        /* store the password entry */
        pfd = fopen(PASSWDFILE, "a");
        if (pfd == NULL) {
          fprintf(stderr, "Cannot open %s for appending\n", PASSWDFILE);
          cleanup(EX_OSFILE);
        }
        if (flock(fileno(pfd), LOCK_EX) == 0)
          (void)fseek(pfd, 0L, 2);
        fprintf(pfd, pentry);
        (void) fclose(pfd);

        enable_xrpt();
}

setupdir ()
{
        disable_xrpt();

        if (mkdir(newuser[HOMEDIR].value, 0775)) {
          error("cannot create login directory %s\n", newuser[HOMEDIR].value);
          return (FALSE);
        }

        (void)sprintf(cmd, "%s %s %s", CHMOD,
          newuser[PROT].value, newuser[HOMEDIR].value);
        (void)system(cmd);

        (void)sprintf(cmd, "%s %s/* %s/.??* %s 2> /dev/null", CP,
          newuser[SKEL].value, newuser[SKEL].value, newuser[HOMEDIR].value);
        (void)system(cmd);

        (void)sprintf(cmd, "chdir /tmp ; %s -o %s -g %s %s", CHALL,
          newuser[LOGIN].value, newuser[GROUP].value, newuser[HOMEDIR].value);
```

```
        (void)system(cmd);

        enable_xrpt();
        return (TRUE);
}

/*VARARGS1*/
error (a, b, c, d, e, f)
char  *a;
{
        fprintf (stderr, "%s: ", PgmName);
        fprintf (stderr, a, b, c, d, e, f);
}
```

5.1.2 Compiler Errors

If the code is compiled in the conforming mode, the following errors are generated by the C compiler (line numbers are not actual):

```
cc: Error on line 11 of /usr/include/grp.h: Syntax Error at "gid_t"
cc: Error on line 12 of /usr/include/grp.h: Syntax Error at "gid_t"
cc: Error on line 16 of /usr/include/grp.h: Syntax Error at "char"
cc: Warning on line 17 of /usr/include/grp.h: non-standard syntax at or before '}'
cc: Error on line 22 of /usr/include/grp.h: argument gid_t does not correspond to
  a function body
cc: Error on line 14 of /usr/include/pwd.h: Syntax Error at "uid_t"
cc: Error on line 15 of /usr/include/pwd.h: Syntax Error at "gid_t"
cc: Error on line 16 of /usr/include/pwd.h: Syntax Error at "gid_t"
cc: Error on line 17 of /usr/include/pwd.h: Syntax Error at "int"
cc: Error on line 18 of /usr/include/pwd.h: Syntax Error at "char"
cc: Error on line 19 of /usr/include/pwd.h: Syntax Error at "char"
cc: Error on line 20 of /usr/include/pwd.h: Syntax Error at "char"
cc: Error on line 21 of /usr/include/pwd.h: Syntax Error at "char"
cc: Warning on line 22 of /usr/include/pwd.h: non-standard syntax at or before '}'
cc: Error on line 24 of /usr/include/pwd.h: argument uid_t does not correspond to
  a function body
cc: Error on line 43 of nu_pcc.c: 'sprintf' redeclared: incompatible types.
cc: Warning on line 506 of nu_pcc.c: illegal pointer/integer combination.
cc: Error on line 507 of nu_pcc.c: struct/union member required.
cc: Error on line 507 of nu_pcc.c: illegal indirection.
cc: Error on line 507 of nu_pcc.c: struct/union member required.
cc: Error on line 507 of nu_pcc.c: illegal indirection.
cc: Error on line 509 of nu_pcc.c: struct/union member required.
cc: Error on line 510 of nu_pcc.c: struct/union member required.
cc: Error on line 565 of nu_pcc.c: struct/union member required.
```

The source of most problems is caused by a change to the structures *group* and *password*. Under previous versions of the operating system, the group element *gid* was declared as type int; POSIX.1 has defined *gid* to be of type *gid_t*. Similiarly, *uid* has been defined to be of type *uid_t*. The definition of *gid_t* and *uid_t* are in <sys/types.h>. This header must be added to the code and declarations changed from *int* to *gid_t* or *uid_t*.

Another problem is the redeclaration of *sprintf()*. The ANSI C specification in <stdio.h> is:

```
extern int   sprintf(char *, const char *, ...);
```

The local declaration can be removed:

```
char      *sprintf();              /* sprintf(3S) */
```

When the code is compiled in the conforming mode, the following error is produced:

```
cc: Warning on line 506 of nu.c: illegal pointer/integer combination.
```

Line 506 is

```
while (pwent = getpwent()) {
```

The function *getpwent()* is a CONVEX extension that is not defined by POSIX.1. In order to successfully compile the program, the conforming mode should be replaced by the default mode (no command line option) where CONVEX extensions are allowed.

5.2 *getpass()*

This example shows the changes required to convert an application using *stty()* or *ioctl()* for POSIX control of terminal I/O. Terminal I/O operations have been standardized in Chapter 7, "Device- and Class-Specific Functions," of POSIX.1.

5.2.1 Original Source

```
/* @(#)getpass.c   4.2 (Berkeley) 7/1/81 */
#include <stdio.h>
#include <signal.h>
#include <sgtty.h>

char getpass_pbuf[9];

char *
getpass(prompt)
char *prompt;
{
        struct sgttyb ttyb;
        int flags;
        register char *p;
        register c;
        FILE *fi;
        int (*signal())();
        int (*sig)();

        if ((fi = fdopen(open("/dev/tty", 2), "r")) == NULL)
                fi = stdin;
        else
                setbuf(fi, (char *)NULL);
        sig = signal(SIGINT, SIG_IGN);
        gtty(fileno(fi), &ttyb);
        flags = ttyb.sg_flags;
        ttyb.sg_flags &= ~ECHO;
        stty(fileno(fi), &ttyb);
        fprintf(stderr, "%s", prompt); fflush(stderr);
        for (p=getpass_pbuf; (c = getc(fi))!='\n' && c!=EOF;) {
                if (p < &getpass_pbuf[8])
                        *p++ = c;
        }
        *p = '\0';
```

```
        fprintf(stderr, "\n"); fflush(stderr);
        ttyb.sg_flags = flags;
        stty(fileno(fi), &ttyb);
        signal(SIGINT, sig);
        if (fi != stdin)
                fclose(fi);
        return(getpass_pbuf);
}
```

If the code is compiled in the conforming mode, the following error is generated by the C compiler (the line number is not exact):

```
cc: Error on line 12 of getpass_pcc.c: unknown size for variable 'ttyb'.
```

Line 12 uses the following structure:

```
struct sgttyb ttyb;
```

The sgttyb structure has been superseded by the *termios* structure defined in <termios.h>. *stty()* and *gtty()* have been superseded by *tcsetattr()* and *tcgetattr()*. Many of the *ioctl()*s described in *tty*(4) have POSIX.1 functions.

5.2.2 Modified Source

The final code follows (changes are in bold):

```
/* @(#)getpass.c   4.2 (Berkeley) 7/1/81 */
#include <stdio.h>
#include <signal.h>
#include <sys/termios.h>

char getpass_pbuf[9];

char *
getpass(prompt)
char *prompt;
{
        struct termios ttyb;
        tcflag_t flags;
        register char *p;
        register c;
        FILE *fi;
        void (*signal())();
        void (*sig)();

        if ((fi = fdopen(open("/dev/tty", 2), "r")) == NULL)
                fi = stdin;
        else
                setbuf(fi, (char *)NULL);
        sig = signal(SIGINT, SIG_IGN);
        tcgetattr(fileno(fi), &ttyb);
        flags = ttyb.c_lflag;   /* the ECHO feature is in the local flags */
        ttyb.c_lflag &= ~ECHO;      /* turn off echoing */
        tcsetattr(fileno(fi), &ttyb);
        fprintf(stderr, "%s", prompt); fflush(stderr);
        for (p=getpass_pbuf; (c = getc(fi))!='\n' && c!=EOF;) {
                if (p < &getpass_pbuf[8])
```

```
                *p++ = c;
        }
        *p = '\0';
        fprintf(stderr, "\n"); fflush(stderr);
        ttyb.c_lflag = flags;   /* restore old tty flags (re-enable echoing) */
        tcsetattr(fileno(fi), &ttyb);
        signal(SIGINT, sig);
        if (fi != stdin)
                fclose(fi),
        return(getpass_pbuf);
}
```

APPENDIX A
POSIX Functions

Function	Purpose
access	File accessibility
alarm	Schedule alarms
asctime	Extensions to time functions
cfgetispeed cfgetospeed cfsetispeed cfsetospeed	Baud rate functions
chdir	Change current directory
chmod	Change file modes
chown	Change owner and group of a file
close	Close a file
closedir directory opendir readdir rewinddir	Directory operations
creat	Create a new file
ctermid	Generate terminal path name
cuserid	Get user name
dup dup2	Duplicate an open file descriptor
exec execl execle execlp execv execve execvp main	Execute a file
_exit	Terminate a process

fcntl	File control
fdopen	Open a stream on a file descriptor
fileno	Map a stream pointer to a file descriptor
fork	Process creation
fpathconf pathconf	Get configurable path name variables
fstat stat	Get file status
getcwd	Get working directory path name
getegid geteuid getgid getuid	Get real and effective user and group IDs
getenv	Environment access
getgrgid getgrnam	Group database access
getgroups	Get supplementary group IDs
getlogin	Get user name
getpgrp	Get process group ID
getpid getppid	Get process and parent process IDs
getpwnam getpwuid	User database access
group	Group database access
isatty	Determine file descriptor association with a terminal
kill	Send a signal to a process
link	Link to a file
longjmp setjmp siglongjmp sigsetjmp	Nonlocal jumps
lseek	Reposition read/write file offset

mkdir	Make a directory
mkfifo	Make a FIFO special file
open	Open a file
passwd	User database access
pause	Suspend process execution
pipe	Create an interprocess channel
read	Read from a file
rename	Rename a file
rmdir	Remove a directory
setgid	Set user and group IDs
setlocale	Extensions to setlocale function
setpgid	Set process group ID for job control
setsid	Create session and set process group
setuid	Set user and group IDs
sigaction	Examine and change signal action
sigaddset sigdelset sigemptyset sigfillset sigismember	Manipulate signal sets
sigpending	Examine pending signals
sigprocmask	Examine and change blocked signals
sigsuspend	Wait for a signal
sleep	Delay process execution
sysconf	Get configurable system variables
tcdrain tcflow tcflush tcsendbreak	Line control functions

tcgetattr tcsetattr	Get and set state
tcgetpgrp	Get foreground process group ID
tcsetpgrp	Set foreground process group ID
termios	General terminal interface
time	Get system time
times	Process times
ttyname	Determine terminal device name
tzset	Set time zone
umask	Set file creation mask
uname	System name
unlink	Remove directory entries
utime	Set file access and modification times
wait waitpid	Wait for process termination
write	Write to a file

APPENDIX B
Glossary

ANSI	American National Standards Institute
ANSI C	Abbreviated name for ANSI X3J11 committee programming C language standard
CONVEX C V4.0	CONVEX C compiler that supports the ANSI C language
ConvexOS V8.0	CONVEX implementation of the Berkeley UNIX operating system containing POSIX.1 functionality with extensions for supercomputer environments
FIFO	First-in-first-out or named pipe
FIPS	Federal Information Processing Standards
NBS	National Bureau of Standards
PCTS	POSIX Conformance Test Suite; the FIPS test suite that verifies the compliance of an implementation to POSIX.1
POSIX	Portable Operating System Interface for Computer Environments; the general group of proposed standards sponsored by various working committees of the IEEE
POSIX.1	IEEE Std 1003.1-1988

APPENDIX C
Reporting Problems

This appendix introduces the CONVEX Technical Assistance Center (TAC) and *contact* utility. The *contact* utility is an online system for reporting problems to the TAC. To learn *contact* by using it, enter **contact** at the system prompt and then answer the questions as they appear on the screen. To find out more about using *contact*, read through this appendix. It describes prerequisites and tips for using *contact* and the step-by-step process *contact* takes you through.

C.1 Technical Assistance Center

The CONVEX Technical Assistance Center (TAC) is staffed by technical specialists who can address diverse questions and problems that arise in a supercomputing environment. If you have a hardware, software, or documentation question, contact the TAC. This group stands ready to solve such problems.

C.2 The *contact* Utility

The TAC recommends using the *contact* utility to report a hardware, software, or documentation problem. The *contact* utility is an interactive program that helps the TAC track reports and route them to the CONVEX personnel most qualified to fix a problem.

After you invoke *contact*, it prompts you for information about the problem. When you finish your report, *contact* electronically mails it to the TAC. You are notified within 48 hours that the TAC has received your report.

C.3 Prerequisites

To use *contact* requires

- a UNIX-to-UNIX Communication Protocol (UUCP) connection to the TAC
- full path name of the program or utility in question
- version number of the program or utility in question

C.3.1 UUCP Connection

Before using *contact*, check with your system administrator to be sure there is a UUCP connection to the TAC. A UUCP connection allows files to be copied from one UNIX-based system to another. The *uucp* (UNIX-to-UNIX copy) command relies on either a dial-up or hard-wired UUCP communication line.

C.3.2 Finding the Program Path Name

To determine the full path name of the program or utility in question, use the *which* command. The next screen illustrates using the *which* command to find the full path name of the loader (*ld*) utility.

```
>which ld
/bin/ld
>
```

In this example, the full path name of the loader is */bin/ld*.

For more information on the *which* command, refer to the *which*(1) man page. You can also use the *info* online information system. Enter **info which** at the system prompt.

If you use the C shell (*csh*), you can also use the *whence* command to find the program path name. The *whence* command works like *which*, but faster.

C.3.3 Finding the Program Version Number

To determine the version number of the program or utility in question, use the *vers* command. The next screen illustrates using the *vers* command to find the version number of the loader (*ld*) utility. Enter **vers**, then the path name of the program or utility.

```
>vers /bin/ld
/bin/ld: 7.0
>
```

In this example, the loader version number is 7.0.

For more information on the *vers* command, refer to the *vers*(1) man page. You can also use the *info* online information system. To do so, enter **info vers** at the system prompt.

C.4 Tips on Using the *contact* Utility

The *contact* utility is interactive and easy to use. This section lists tips to help use it efficiently. In particular, this section tells how to.

- use a *.contact* file
- abort a contact session
- resubmit an aborted report
- suspend a contact session
- move from one prompt to another
- use tilde-escape sequences in the *contact* utility

C.4.1 Using a *.contact* File

When you invoke *contact*, it prompts for information regarding the problem. The first prompt is for your name, title, phone number, and company name. You can, however, create a *.contact* file to skip this first prompt. Follow these steps:

1. Create a *.contact* file in your home directory.

2. Enter your name, job title, phone number, and company name, each on a new line.

When you invoke *contact*, it automatically includes the *.contact* file as input for the first prompt and proceeds to the next prompt.

C.4.2 Aborting the Report

To abort a contact report, either press the interrupt key (usually CTRL-C) or choose the abort option when prompted by the *contact* utility. Using CTRL-C to abort does not save the contents of the report. Using the abort option saves the contents of the report in a file named *dead.report* in your home directory.

C.4.3 Submitting the *dead.report* File

After you abort a contact session, the *contact* utility saves the report in a file named *dead.report* in your home directory. Using the *contact* command with the *-r* option automatically merges the contents of the *dead.report* file into the new contact session. Enter

contact -r

and *contact* finds the *dead.report* file in your home directory and merges it into the contact report. You can then edit the report. When you end the editing session, *contact* returns to the final prompt, which asks you to review, edit, submit, or abort the report.

C.4.4 Suspending a Report

Sometimes it is necessary to stop in the middle of a contact report and return to the shell (for instance, to suspend the contact session to find the program path name or version number). To suspend the contact session, press CTRL-Z. To return to the contact session, enter **fg**. Using CTRL-Z and the *fg* (foreground) command lets you toggle back and forth between the *contact* utility and the shell. You cannot, however, use CTRL-Z and *fg* to toggle back and forth if you are using the Bourne shell (*sh*).

C.4.5 Ending a Response

The *contact* utility prompts for information pertinent to your hardware, software, or documentation question. Some prompts require one-line responses; to move to the next prompt, press RETURN. Other prompts require more than a one-line response; to move to the next prompt, press CTRL-D.

C.4.6 Tilde-Escape Sequences

The *contact* utility treats input beginning with a tilde (~) as a special sequence. The character following the tilde is considered a request for a special function. The following tilde sequences are recognized by *contact*:

~e	start the text editor (defined in the EDITOR environment variable)
~h	display a list of available tilde-escape sequences
~p	print the contact report to the terminal screen
~r *filename*	read the contents of *filename* as a response to the current prompt. Some prompts require only a one-line response. This tilde-escape sequence works only for prompts that allow more than a one-line response.
~~	insert a single tilde as the first character in the line

C.5 Using the *contact* Utility

The *contact* utility prompts for the following information:

- your name, title, phone number, and corporate name
- name and version of the product
- one-line summary of the problem
- detailed description of the problem
- priority of the problem
- instructions on how to reproduce the problem
- comments about the problem
- comments about the documentation related to the problem
- files to include in the contact report

The following is a step-by-step discussion of these prompts.

Step 1a To invoke the *contact* utility, enter **contact** at the system prompt. The system responds with a welcome message and a series of questions regarding your hardware, software or documentation question. The next screen illustrates the *contact* command and the system response.

```
>contact
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
>
```

Step 1b If there is a *.contact* file in your home directory, *contact* skips the first prompt. The next screen illustrates the *contact* command and the system response when a *.contact* file is in your home directory.

```
>contact
Welcome to contact version 0.11 ()

Enter the name of the product involved
>
```

Step 2 The *contact* utility prompts for the version number of the product. If you do not know the version number, use CTRL-Z to suspend the session. Use the *which* (or *whence* if you use *csh*) and *vers* commands to find the version number of the product. Use the *fg* command to return to the session and enter the version number in the form X.X or X.X.X.X.

Step 3 The *contact* utility prompts for a one-line summary of the problem. This summary is the subject header in any further correspondence regarding the problem. Please make this summary as descriptive as possible in one line.

Step 4 The *contact* utility prompts for a detailed description of the problem. Please make this description as complete as possible. Include source code and a stack backtrace when possible. (Refer to the *adb*(1) or *csd*(1) man page for information on obtaining a stack backtrace.) The more information you provide, the quicker the TAC can isolate and solve the problem.

Step 5 The *contact* utility prompts for the priority of the problem. The next screen illustrates this prompt and priority levels from which to choose; you must enter a priority number.

```
Enter a problem priority, based on the following:
1) Critical    - work cannot proceed until the problem is resolved.
2) Serious     - work can proceed around the problem, with difficulty.
3) Necessary   - problem has to be fixed.
4) Annoying    - problem is bothersome.
5) Enhancement - requested enhancement.
6) Informative - for informational purposes only.
>
```

Step 6 The *contact* utility prompts for an explanation of how to reproduce the problem. Please include the command syntax and options you used and anything else you did to make the program run.

Step 7 The *contact* utility prompts for any other pertinent comments. Please include all relevant information.

Step 8 The *contact* utility prompts for suggestions regarding documentation supporting the product. Indicate whether documentation could be revised to address the problem.

Step 9 The *contact* utility asks for names of files necessary to reproduce the problem. The next screen illustrates the *contact* prompt and sample user response.

```
Are there any files that should be included in this report (yes | no)?
>yes
Please enter the names of the files, one to a line (^D to terminate)
>test.f
>~/subroutines/sub.f
>
```

Note

Tilde-escape sequences are not recognized in responses to this prompt. In *contact*, a tilde in this section means your home directory. This convention is based on use of the tilde for expanding file names in *csh*.

If files specified are small text files, they are automatically included in the contact report. If the files are too large to be included in this report, *contact* gives further instructions on how to submit these files.

To specify a directory, combine directory files into a single file using the *tar* command (refer to the *tar*(1) man page for further information) or enter each file name in the directory on a single line in the contact report.

Step 10 The *contact* utility prompts you to review, edit, submit, or abort the contact report. The next screen illustrates this prompt.

```
Please select one of the following options:
1) Review the problem report.
2) Edit the problem report.
3) Submit the problem report.
4) Abort the problem report.
>
```

Choose the number of the option you want to select. These options let you do the following:

Review — review the text of the contact report. You are then prompted again to select an option.

Edit — edit the text of the contact report. If you choose to edit the report, *contact* puts you in your default text editor.

Submit — sends the report to the CONVEX TAC. You are notified within 48 hours that the TAC has received the report. This option exits the *contact* utility and returns you to the shell.

Abort — saves the text of the report in a file named *dead.report* in your home directory. This option exits the *contact* utility and returns you to the shell.

Index

CONVEX POSIX Concepts
Document No. 710-005030-000, First Edition

Reader's Forum

You are invited to submit your comments concerning the clarity and service of this manual. Constructive critical comments are most welcome and will help us continue in our efforts to generate quality customer documentation. (Please list the page number for questions and comments):

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

From:

Name_____ Title_____

Company_____ Date_____

Address and Phone No._____

FOR ADDITIONAL INFORMATION/DOCUMENTATION:

Location	Phone Number
Within continental U.S.	1(800)952-0379
Outside continental U.S.	Contact local CONVEX office

Direct Mail Orders to:

CONVEX Computer Corporation
Customer Service
P.O. Box 833851
Richardson, Texas 75083-3851 USA

(Fold Here First)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson TX 75083-3851

(Fold Here Second)

(Tape or Staple)