Convex Exemplar Software Release Notice

Document No. 710-029530-015

August 1995

Convex Press
Richardson, Texas
United States of America

Convex
Exemplar Software
Release Notice

Document No. 710-029530-015

This notice is recyclable.

Printed in the United States of America

Convex SPP-UX V3.0.5

Overview

This release notice describes the V3.0.5 release of the SPP-UX operating system. It highlights new features and changes to existing features, and supplements the permanent documentation with information developed too late for inclusion. Read this document before reporting problems; your questions may be answered here.

Hardware requirements

The V3.0.5 release of SPP-UX supports the following hardware platforms:

- Convex Exemplar SPP-1000/XA systems consisting of one to eight hypernodes
- Convex Exemplar SPP-1000/CD compact design systems
- Convex Exemplar SPP-1200A/XA systems consisting of two to four hypernodes

Install the hardware according to the instructions in the site preparation and installation guides that accompany your system.

Configuration restrictions

The following configuration restrictions must be met on your Exemplar system in order to run SPP-UX V3.0.5:

- Each hypernode must have between 256 MB and 2 GB of physical memory.
- Each hypernode must have between one and three SCSI Sbus controllers.
- Node IDs must be sequential, starting at 0.
- Node 0 must have a vnode pager for the entire system; this means that the file /paging/space must reside on node 0, and

must be called by the `swapon` command in the `/etc/rc` file, or in one of the files executed by `/etc/rc`.

- A default pager partition must remain active on each hypernode at all times.

In addition, OpenBoot Prom (OBP) does not support booting a large filesystem as root.

Upgrade prerequisites

Exemplar SPP-1000/XA and SPP-1000/CD systems that are updating to the V3.0.5 release of SPP-UX require

- SPP-UX V2.1.1 or
- SPP-UX V3.0.2 or later

Exemplar SPP-1200A/XA systems have no upgrade prerequisite.

Corequisites

The V3.0.5 release of SPP-UX has the following corequisites:

- OpenBoot firmware must be at version V1.3.2.
- Test station diagnostic software and CST software must be at V3.3.
- All layered products, including compilers and tools, must be those that accompany the V3.0.5 distribution of SPP-UX

These corequisites must be installed on your Exemplar test station immediately after you install the Exemplar software upgrade.

Associated documentation

The following Convex documents are applicable to the SPP-UX V3.0.5 release:

- *Exemplar Architecture*, First Edition (DHW-014)
- *Exemplar Site Preparation*, First Edition (DHW-500)
- *Exemplar Software Distribution Notice* (710-031230-007)
- *Exemplar Installation Guide*, First Edition (DHW-500)
- *SPP-UX System Administration Guide*, First Edition (DSW-853)

New features

The following features are new in this SPP-UX V3.0.5 release:

- Product licensing
- Large File Systems support
- Phase II of Growable Stacks
- SPP-1200A/XA systems support
- New diskutil command line interface
- Remote OpenBoot prom (*robp*) utility
- `syspic` utility
- 3490 support
- SPP-1200 I/O Architecture
- Ethernet (requires SPP-1200 I/O Architecture)

The following sections summarize important aspects of each new feature.

Product licensing

SPP-UX is now a licensed product. If users do not acquire a license when running SPP-UX, they are instructed to contact the system administrator at your site for assistance. Convex products are licensed using the FlexLM licensing system. Refer to the SPP-UX Distribution notice which accompanies this release and the *FLEXlm End User Manual* for licensing information.

Large File Systems support

This release of SPP-UX contains Large File Systems support.

SPP-UX V3.0.5 allows you to create and manipulate files up to one terabyte minus 512 bytes in length. (One terabyte is 2^{40} bytes.) However, because not all utilities and applications handle files larger than two gigabytes, you must be aware of the conditions for working with such files. For more information about Large Files functionality see the "Large files" section on page 11.

Note

Once the large file aware utilities have been installed on your system you cannot revert to an older version of the operating system without removing the utilities. Common symptoms of a operating system/ utilities mismatch are failure to boot with "bad system call" messages. This occurs because the utilities try to execute the new system calls that are not supported in older versions of the operating system.

Caution

If you decide to reinstall an older version of SPP-UX, there is a possibility of data loss in any large file system created with SPP-UX V3.0.3.

Note

OBP does not support booting a large filesystem as root. The root device must be made with the magic number— FS_MAGIC_LFN— to boot successfully.

If you choose to de-install SPP-UX V3.0.5 you must re-install the desired version and its associated utilities from tape.

Phase II of growable stacks

Previous versions of SPP-UX are limited because, as process stack regions are allocated, their maximum possible size is deducted from a running total of pagefile space and physical memory. Processes fail to fork/exec if there is not enough space, even if most of the space is unused.

In this version of SPP-UX, stack regions— including the process thread 0 stack and any thread stacks created by cpslib—have virtual memory in the process address space reserved. However, they do not use the paging file until pages are actually touched by the program.

Changes and enhancements to tunables and system calls include the following:

- The `maxssiz` tunable now sets the default maximum size to which the process thread 0 stack can grow.

  If you are upgrading your system to SPP-UX V3.0.5, you may want to update this value to a reasonable default maximum stack size in your existing tunables files.

  - For HP-UX SOM executables, the maximum size of the process stack is about 80 megabytes. If the tunable `maxssiz` is set larger than this, the stack is still limited to the 80 megabyte maximum.
  - For SPP-UX ESOM executables, the size of a thread stack is limited by the top of the gateway page and the base of the emulator region, about 768 megabytes. If larger values of `maxssiz` are supplied, default stacks are set to this maximum.
- The `dflssiz` tunable is no longer used and may be deleted from your existing tunables files.
- The maximum size of the process thread 0 stack can be increased or decreased from the value set by the tunables file

by using the `mpa` utility or by calling `setrlimit`. See the mpa(1) and the getrlimit(2) man pages for more information.

Note

Changes to stack size via setrlimit only apply to child processes of the setrlimit caller.

- The process thread 0 stack is placed in process virtual memory up against the base of the emulator region. In prior versions of SPP-UX, there was usually a large hole in a process' address space above the stack.
- Thread stack regions created by cpslib are also created with this attribute and while virtual memory is reserved, pagefile space is used only for pages that are touched.
- Programs that use memory regions in a manner similar to stacks—that grow from lower to higher addresses and use a variable amount of memory—may take advantage of this feature by using `mmap MAP_ANONYMOUS` and a new flag `CNX_MAP_STACK`. See the mmap(2) man page for more information.

SPP-1200A/XA and 1200/XA support

SPP-UX now runs on Exemplar SPP-1200A/XA and Exemplar SPP-1200/XA systems. Previous versions of SPP-UX do not run on these systems.

New diskutil command line interface

The `diskutil` utility has a new command line interface in diskutil. The `diskutil` interface, which creates and accesses stripe related information, now provides the following new functionality:

- Selecting a disk stripe as you would select a disk.
- Getting a list of all disk stripes on all hypernodes of a multi-node system.
- Getting information about a particular stripe across hypernodes.
- Viewing stripe-related information about a particular disk partition.

Remote OpenBoot Utility (*robp*)

The `robp` utility provides a user interface to OpenBoot system information while SPP-UX is running. `robp` allows access to OpenBoot commands such as `show-devs` and `printenv`, but

does not allow system information to be changed. `robp` does not provide a Forth language interface. For a list of functions currently available under `robp`, enter the `help` subcommand after you have started `robp` or see the robp(1m) man page.

`robp` is part of SPP-UX, and requires version 1.3.2 of OBP. `robp` requires post-installation configuration; see Chapter 2 of the *Exemplar Software Distribution Notice* for this release for more information.

`syspic` available

The system performance monitor utility `syspic` is available with this release of SPP-UX. The executable is located in /usr/convex/bin.

3490 cartridge tape drive

SPP-UX V3.0.5 supports the 3490 cartridge tape devices, in addition to 3480 and DAT drives.

SPP-1200 I/O Architecture

SPP-UX V3.0.5 supports the SPP-1200 I/O Architecture. This provides support for up to 8 SBUS controllers per hypernode.

Ethernet

SPP-UX V3.0.5 supports Ethernet, which requires the SPP-1200 I/O architecture.

Known problems

This section describes problems in this release of SPP-UX that are known at this time. Workarounds—where available—for these problems are described.

For a complete list of open bugs against SPP-UX V3.0.5, contact the Convex Technical Assistance Center.

Large filesystem as root

Problem

SPP-UX V3.0.5 and OpenBoot Prom (OBP) do not support booting a large filesystem as root.

Workaround

Do not attempt to use a large filesystem as root. In addition, the following restrictions apply to this release of SPP-UX:

- A bootable partition must be an HFS long-file-name type filesystem.
- You must use an 8Kbyte blocksize.
- The filesystem must be made on a partition that is less than 2G in size and is between 0 and 4G (OBP cannot read beyond 4G on a disk).
- The OBP file reader cannot follow symbolic links, so filenames must be hard links, like the standard /os/mach, /os/tunables, and /os/server.

netlsd does not work

Problem

/etc/netls/netlsd does not work correctly.

Workaround

None. netlsd will be fixed in a future release of SPP-UX.

ps hangs

Problem

`ps` and other commands using the `CNX_SYSINFO` system calls may hang, causing all subsequent calls to `CNX_SYSINFO` to hang.

Workaround

The hanging command may be suspended or killed. Should that fail, the login session should be closed and a new login session established.

Large files

SPP-UX allows you to create and manipulate files up to one terabyte minus 512 bytes in length. (One terabyte is 2^{40} bytes.) However, because not all utilities and applications handle files larger than two gigabytes, you must be aware of the conditions for working with such files.

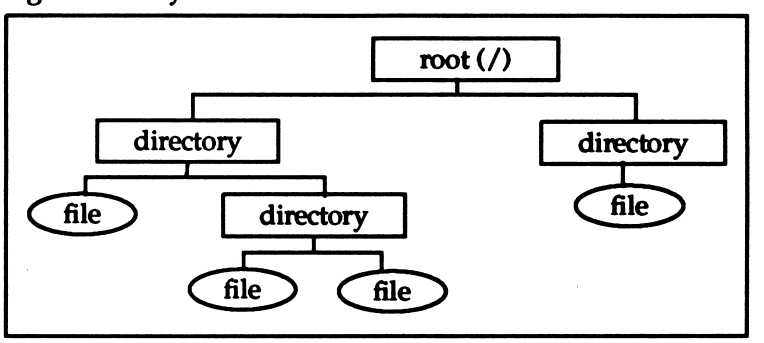
In this release notice, the term *large files* refers to files larger than two gigabytes. This information is presented in the following sections:

- The first section, "File systems and large files," explains factors of your file system environment that are important when working with large files.
- The second section, "SPP-UX utilities and large files," explains conditions that apply to utilities in a large files environment. This section includes operator-oriented utilities such as `dump` and `tar`.
- The third section, "Programming with large files," covers strategies for programmers who wish to write applications to create or manipulate large files. This section includes descriptions of special system calls for use with large files.

File systems and large files

File systems are structures that computers use to organize and store information. In SPP-UX, the file system is a hierarchical tree as illustrated in Figure 1.

Figure 1 File system hierarchical tree



Each file is connected or related to another starting with the root (/) directory file and continues downward through a virtually unlimited number of files and directories.

The term file system can refer to the entire file hierarchy or to a subsection of the file tree. Here, the term file system refers to a subsection of the file tree; a collection of files, directories, and file management structures that are assigned by the system manager to a certain portion of the disk system.

File system characteristics

Step 1 To find out what file system you are working in, enter

```
bdf .
```

Figure 2 shows sample output from the `bdf` command.

Figure 2 Sample `df` output

```
% bdf .
File system     kbytes      used      avail   capacity      Mounted on
/dev/st3        4140152     2980928   745232  80%            /test
```

Step 2 Look at the far right column of the `bdf` output for the name of your file system.

If the far left column of the `bdf` output starts with *machinename*:, instead of `/dev/`*xxx*, then you are working in a file system that is NFS-mounted from remote host *machinename*. If this is the case, you cannot create or manipulate large files in this file system.

The Network File System (NFS) allows file systems from other machines to be remote mounted on your machine; you access normal files in NFS-mounted file systems just like you access files that really live on your home machine. However, the current supported NFS protocol limits the size of files that can be accessed via NFS to two gigabytes or less.

FD_MAGIC_2 and FSF_LARGEFILES

The same large file restrictions apply to local file systems that do not have the `FD_MAGIC_2` magic number and `FSF_LARGEFILES` feature bit set in the filesystem superblock. Note, that filesystems created prior to this release will by default not contain the proper magic number/feature bit pair to allow large files. In order to create a filesystem that allows large files you must create the filesystem with the `cnx_newfs`(1M) or `cnx_mkfs`(1M) provided with this release. By default `mkfs` will not create a filesystem that allows the creation of large files.

Note

OBP does not support booting a large filesystem as root. The root device must be made with the magic number— FS_MAGIC_LFN— to boot successfully.

Before creating or manipulating a large file, make sure you will not place it in a file system that cannot hold it. Non-large-file compatible file system types and the restrictions they impose are described in the following sections. To determine if a filesystem

has the appropriate magic number/feature bit pair execute the `cnx_dumpfs` command:

Step 1 Enter

`cnx_dumpfs -s /filesystem`

Figure 3 shows sample output from the `cnx_dumpfs` command.

Figure 3 Sample of `cnx_dumpfs` output

```
% cnx_dumpfs -s /filesystem
magic FD_MAGIC_2
featurebits     FSF_LARGEFILES  FSF_LFN
```

Step 2 Verify that the magic number is `FD_MAGIC_2` and that the featurebits field contains `FSF_LARGEFILES`.

To disallow large files in a given file system, use the `-N` option of `cnx_mkfs` or `cnx_newfs`. If the filesystem is created with the `-N` option, `cnx_mkfs` will not set the `FSF_LARGEFILES` featurebit and thus disables the ability to create large files on the resulting filesystem. The `-N` option is available only when invoking `cnx_mkfs` or `cnx_newfs`. A hard link has been created between `mkfs` and `cnx_mkfs` to retain backwards compatibility with Hewlett-Packard's utility set.

SPP-UX utilities and large files

Some SPP-UX utilities work on large files in the same manner as they do regular files. Others place restrictions on large file operations. The following sections explain how some SPP-UX utilities interact with large files.

General use utilities

Not all utilities are practical for use with large files. For example, you are not likely to try to examine a large file using `more`. The SPP-UX utilities most likely to be useful with large files have been modified to ensure that they work with large files. These utilities are

- `cat`
- `chmod`
- `cp`
- `dd`
- `find`
- `ftp`
- `ls`

- `mount`
- `mv`
- `rcp`
- `rm`
- `tail`

Use of all utilities is subject to the file system restrictions described in the section "File Systems and Large Files". An attempt to use any utility to place a large file in a file system where it is not allowed may result in truncation of the file.

For example, suppose you want to move large file test1 from file system /foo to file system /bar using `mv`. If /bar cannot hold large files, `mv /foo/test1 /bar` returns a write failed message. /bar/test1 contains only the first two gigabytes of test1.

Note

Once the large file aware utilities have been installed on your system you cannot revert to an older version of the operating system without removing the utilities. Common symptoms of a operating system/ utilities mismatch are failure to boot with "bad system call" messages. This occurs the utilities try to execute the new system calls that are not supported in older versions of the operating system.

Shells and pipes

The `sh` and `csh` shells and their variants cannot manipulate large files. This means that functions such as shell redirection do not work for large files.

Pipes do work with large files. Using pipes in combination with various utilities, you can replicate many shell functions. For instance:

```
cat fname1 > fname2
```

does not work because it relies on the shell to write into fname2. However,

```
cat fname1 | dd of=fname2
```

has the same result and is valid for large files.

You can also use pipes to feed large files to some utilities that otherwise could not handle them. For instance,

```
cat fname | grep pattern
```

works, even though

```
grep pattern fname
```

does not.

File system utilities

Several utilities used for creating and maintaining file systems are able to work with large files. Many of these are used by the system manager and require root permission. The file system utilities are

- `fsck`
- `fsirand`
- `mkfs, cnx_mkfs`
- `ncheck`
- `newfs, cnx_newfs`
- `dumpfs, cnx_dumpfs`

`dump`, `restore`, `fbackup`, and `frecover`, used by operators and users to backup and restore file systems, also understand large files. These utilities handle large files just as they do smaller files.

Three other file system utilities have limited usefulness with large files. `tar`, `pax`, and `cpio` have a limit of 11 octal digits built into their format; they can be used to backup files up to eight gigabytes in size. If any of these utilities encounters a file larger than eight gigabytes, it issues a warning and skips that file.

Holes in large files

A hole is a region of a file that has not been written to but has been bypassed with the `seek` system call. The presence of the hole is recorded, but its content is not stored on disk and it does not occupy any disk space.

A large file containing a hole may fit on a particular file system, while a file of the same size without the hole may not. `cp` and `mv` fill in holes with zeroes. The following examples explain how you can detect this.

For example, the file myfile has a size of 10 megabytes as reported by `ls`, but is only taking up 32 kilobytes of actual disk space, according to `du`:

```
% ls -l myfile
-rw------- 1 joe 10485760 Sep 10 15:26 myfile
% du -a myfile
32      myfile
```

This discrepancy occurs because myfile contains a hole. If myfile is copied or moved to another directory, the holes are filled with zeros:

```
% cp myfile bigfile
% ls -l bigfile
-rw------- 1 joe 10485760 Sep 10 15:30 bigfile
% du -a bigfile
10304   bigfile
```

Programming with large files

Two actions allow a program to read or write beyond the two gigabyte boundary in a file.

- Setting the `O_LARGEFILE` flag when a file is opened. To do so use the `open()` system call, the `fopen()` system call, or the `open64()` library routine,

  ```
  fd=open("largefilename", O_RDWR | O_LARGEFILE);
  ```

  or

  ```
  FD=fopen("largefilename", "rwl");
  ```

  or

  ```
  fd=open64("largefilename", O_RDWR);
  ```

- Setting the same flag with the `fcntl()` system call.

  ```
  fd=open("largefilename", O_RDWR);
  flag=fcntl(fd, F_GETFL, O);
  fcntl(fd, F_SETFL, flag!FLARGEFILE);
  ```

Note

You must use `fcntl` properly when working with large files. Shortcut methods may lose the large file flag.

For example, setting Async I/O on a large file with a call structure such as the following

```
fcntl(fd, F_SETFL, FASIO)
```

loses the `O_LARGEFILE` flag. However,

```
flag=fcntl(fd, E_GETFL, O);
flag|=FASIO
fcntl(fd, F_SETFL, flag);
```

keeps the setting of `O_LARGEFILE`.

The following sections explain tools available to programmers who wish to manipulate large files. Later sections explain some restrictions to bear in mind when operating on large files.

FORTRAN support for large files

Transparent support for large files is available with Convex FORTRAN V9.2. Large formatted and unformatted files may be read and written. The FORTRAN I/O interface extends transparently to large files.

Direct access files are limited to 2**31-1 RECORDS. The actual size of such a file is then limited by the size of individual records. This restriction is due to the use of the `INTEGER*4` data type for the REC= specifier in READ and WRITE statements for direct access files and the NEXTREC= specifier in INQUIRE statements.

FORTRAN programs must be re-linked with V9.2 in order to read and write large files. All FORTRAN programs linked with the V9.2 library have large file access.

System calls for use with large files

Many system calls take a file offset as an argument. 64-bit versions of many of these system calls are available for use with large files. Several libc routines are available for use with large files also: `creat64()`, `fseek64()`, `ftell64()`, `fopen()`, `fgetpos64()`, `fsetpos64`, and `open64()`.

Each of these functions has a 32-bit counterpart whose name is the same except for the ending 64 (`lseek64()`, `lseek()`; `fseek64()`, `fseek()`; etc.). Except as noted, the 64-bit function has the same functionality as its 32-bit counterpart, but takes a 64-bit offset.

The new system calls and libraries live in the static library /usr/convex/all/spp1/libc.a.

- `lseek64()`— seek on a file descriptor using 64 bit offsets. A successful call to `lseek64()` sets the `O_LARGEFILE` bit.
- `fseek64()`— seek on a stream using 64 bit offsets. Implemented with `lseek64()`.
- `stat64()`— perform a `stat()` on a large file, returning a structure with 64 bit offsets. Uses the `stat64_t` structure; declares the `st_size` field as an `off64_t`.
- `fstat64()`— perform an `fstat()` on a file descriptor, returning a structure with 64 bit offsets.
- `lstat64()` — `lstat()` with 64 bit offsets.

- `ftell64()` — stream position as a 64 bit offset.
- `fgetpos64()` — get stream position as a 64 bit offset.
- `fsetpos64()` — set stream position as a 64 bit offset.
- `open64()`— open a file, with O_LARGEFILE bit implied.
- `creat64()`— equivalent to open64(path, `O_CREAT | O_TRUNC | O_WRONLY`, mode).
- `truncate64()` — `truncate()` using 64 bit offsets.
- `ftruncate64()` — `ftruncate()` using 64 bit offsets.

The existing versions of `read()` and `write()` work with large files, but must have the `O_LARGEFILE` bit set in order to function past two gigabytes into a file.

Note that `open()` will fail when attempting to open an existing large file when the `O_LARGEFILE` bit is not set.

Limited system calls

The `mmap()`, `getrlimit()`, and `setrlimit()` system calls require 32-bit file offsets and do not understand large files. They do not adversely affect large files, but their usefulness with respect to large files is limited.

`mmap()` maps only the first two gigabytes of a file. `setrlimit()` can limit file size with byte granularity up to two gigabytes. It may, as always, set the file size limit to `RLIM_INFINITY` to allow access to any size file. `setrlimit()` cannot set, and `getrlimit()` cannot understand, limits above two gigabytes.

Include file values

The following include files contain definitions useful in programming with large files:

- <sys/cnx_types.h> contains the typedef `off64_t`, which defines the possible offsets, and thus the maximum size, of a file. It is defined as a long long.
- <sys/cnx_stat.h> contains the structure, `struct stat64`, used by `stat64()`.
- The `open()` and `fcntl()` flag `O_LARGEFILE` is defined in <sys/cnx_fcntl.h>.
- Function prototypes for the new syscalls and libraries are defined in the following:
  - <sys/cnx_fcntl.h>
  - <sys/cnx_stat.h>

- <sys/cnx_stdio.h>
- <sys/cnx_unistd.h>

File system considerations

Although the stated limit for large files is one terabyte minus 512 bytes, some combinations of block size and number of free blocks make it impossible to write a file that large. On file systems with block sizes less than 32K, you must use triple indirect block pointers to write a large file of maximum size. In such a file system (block size < 32K), the availability of block pointers determines the maximum file size you may write.

Existing programs and large files

The behavior of existing programs with large files is determined by the behavior of the system calls those programs use. Recompiling an old executable in a large file environment has no effect because it still uses the same system calls.

The danger in using an existing program with a large file is to the file, not the program. A program that does not understand large files cannot open a large file. The following paragraphs explain how several system calls behave if they encounter a large file.

`lseek()`
: Fails on a large file if the file offset is already greater than two gigabytes or if a successful seek would cause the offset to be greater than two gigabytes. It ignores the `O_LARGEFILE` bit.

`open()`
: Fails when attempting to open a large file.

`stat()`
: Fails when performed on a large file.

`truncate()` and `ftruncate()`
: Truncate a large file at the two-gigabyte boundary.

Remember that these limitations also apply if a program that understands large files calls another program that does not.