

Common Derived Types for SPARC V8 and V9

Tim Marsland and Jonathan Chew

1.0 Introduction

A number of the system derived types will change in the include files that will be shared between 32-bit and 64-bit versions of the applications environment because of one of the following reasons:

1. The basic difference in C data type models between ILP32 and LP64.
2. The opportunity to expand some of the types for the future.
3. To fix any types that were not properly defined before.

There is a trade-off between making space for future developments, and reducing interoperability and compatibility. We've attempted to make some decisions here, and explain our rationale. Most of the types are the scalar types from `<sys/types.h>`. These types are used in many fundamental system interfaces, so it's important that we get this correct from the beginning. Our approach has been a conservative one, largely because we want to reduce the binary compatibility and interoperability problems to a minimum

The rest of this document lists the system derived types to be changed, each with its current and proposed types and a short description. The definitions used by SGI and DEC are shown for comparison. This comparison is interesting because some customers have already ported code to Irix and OSF/1.

1.1 C++ and derived types

C++ presents some peculiarly tricky problems for us. Our goal here is to change our implementation of the derived types of the system so that they grow (or don't grow) appropriately when moving from an ILP32 to an LP64 compilation environment. In the ILP32 compilation environments, C compilers allow 32-bit quantities to be called **int** or **long**, so that we can change the implementation of today's derived types without any source or binary compatibility problems.

However, C++ treats C derived types simply as aliases; that is, it explicitly does not capture anything but the underlying type implementation. A function or method *e.g.* **gloop()** that looks like this in source form

```
void gloop(size_t);
```

embeds the *implementation* of **size_t** (currently **unsigned int**) and not **size_t** into any object file that either implements or consumes **gloop()**. So, if we were to change the implementation of **size_t**, then unless all objects that are aware of the definition of **gloop()** were recompiled, applications will break.

We propose to deal with this problem using the following form of header guard:

```
#if !defined(_LP64) && defined(__cplusplus)
typedef ...          /* today's version, unchanged */
#else
typedef ...          /* changed version */
#endif /* !_LP64 && __cplusplus */
```

This ensures that both C and C++ programs created in a 32-bit compilation environment to be binary compatible with older releases.

1.2 Large Files Compilation Environments

For ILP32 compilation, all the large file compilation environments must be preserved unaltered. For the LP64 compilation environment, we propose to adopt the effective types and interfaces of `_FILE_OFFSET_BITS == 64`. We do not propose to create a “small files” compilation environment as a subset of the LP64 environment, and it will be a compilation error to set `_LP64` and `_FILE_OFFSET_BITS == 32`.

2.0 List of Type Changes

TABLE 1. Proposed Changes to Common System Derived Types

Derived Type	Current	Proposed	SGI 32-bit	SGI 64-bit	DEC
<code>blkcnt_t</code> ¹	<code>long</code>	<i>unchanged</i>			
<code>blkcnt_t</code> ²	<code>longlong_t</code>	<i>unchanged</i>			
<code>blkcnt64_t</code> ³	<code>longlong_t</code>	<i>unchanged</i>			
<code>clock_t</code>	<code>long</code>	<code>int</code>	<code>long</code>	<code>int</code>	<code>int</code>
<code>daddr_t</code>	<code>long</code>	<code>int</code>	<code>long</code>	<code>long</code>	<code>int</code>
<code>dev_t</code>	<code>ulong_t</code>	<code>unsigned int</code>	<code>unsigned long</code>	<code>__uint32_t</code>	<code>int</code>
<code>diskaddr_t</code>	<code>longlong_t</code>	<i>unchanged</i>			
<code>fsblkcnt_t</code> ¹	<code>ulong_t</code>	<i>unchanged</i>			
<code>fsblkcnt_t</code> ²	<code>u_longlong_t</code>	<i>unchanged</i>			
<code>fsblkcnt64_t</code> ³	<code>u_longlong_t</code>	<i>unchanged</i>			
<code>fsfilcnt_t</code> ¹	<code>ulong_t</code>	<i>unchanged</i>			
<code>fsfilcnt_t</code> ²	<code>u_longlong_t</code>	<i>unchanged</i>			
<code>fsfilcnt64_t</code> ³	<code>u_longlong_t</code>	<i>unchanged</i>			
<code>gid_t</code>	<code>uid_t</code>	<i>unchanged</i>	<code>long</code>	<code>__int32_t</code>	<code>uint_t</code>
<code>hostid_t</code>	<code>long</code>	<i>omit</i>	<code>long</code>	<code>long</code>	
<code>id_t</code>	<code>long</code>	<code>int</code>	<code>long</code>	<code>__uint32_t</code>	<code>int</code>
<code>ino_t</code>			<code>unsigned long</code>	<code>unsigned long</code>	<code>uint_t</code>
<code>ino_t</code> ¹	<code>ulong_t</code>	<i>unchanged</i>			
<code>ino_t</code> ²	<code>u_longlong_t</code>	<i>unchanged</i>			
<code>ino64_t</code> ³	<code>u_longlong_t</code>	<i>unchanged</i>			
<code>len_t</code>	<code>u_longlong_t</code>	<i>unchanged</i>			
<code>major_t</code>	<code>ulong_t</code>	<code>unsigned int</code>	<code>ulong_t</code>	<code>__uint32_t</code>	<code>uint_t</code>
<code>minor_t</code>	<code>ulong_t</code>	<code>unsigned int</code>	<code>ulong_t</code>	<code>__uint32_t</code>	<code>uint_t</code>
<code>mode_t</code>	<code>ulong_t</code>	<code>unsigned int</code>	<code>unsigned long</code>	<code>__uint32_t</code>	<code>uint_t</code>
<code>nlink_t</code>	<code>ulong_t</code>	<code>unsigned int</code>	<code>unsigned long</code>	<code>__uint32_t</code>	<code>ushort</code>
<code>off_t</code>			<code>long</code>	<code>long</code>	<code>long</code>
<code>off_t</code> ¹	<code>long</code>	<i>unchanged</i>			

Derived Type	Current	Proposed	SGI 32-bit	SGI 64-bit	DEC
off_t ²	longlong_t	<i>unchanged</i>			
off64_t ³	longlong_t	<i>unchanged</i>			
offset_t	longlong_t	<i>unchanged</i>			
paddr_t	ulong_t	<i>omit?</i>	unsigned long	unsigned long	int
pid_t	long	int	long	__int32_t	int
ptrdiff_t	int	intptr_t	int	__int64_t	long
rlim_t					unsigned long
rlim_t ¹	unsigned long	<i>unchanged</i>			
rlim_t ²	u_longlong_t	<i>unchanged</i>			
rlim64_t	u_longlong_t	<i>unchanged</i>			
size_t	uint_t	unsigned long	unsigned int	unsigned long	unsigned long
speed_t	unsigned long	unsigned int			unsigned int
ssize_t	int	long	unsigned int	unsigned long	long
tcflag_t	unsigned long	unsigned int			unsigned int
time_t	long	int	long	int	int
u_offset_t	u_longlong_t	<i>unchanged</i>			
uid_t	long	int	long	__int32_t	uint_t
wchar_t	long	int			unsigned int

¹ Large Files - `_FILE_OFFSET_BITS == 32`

² Large Files - `_FILE_OFFSET_BITS == 64`

³ Large Files - `#ifdef _LARGEFILE64_SOURCE`

3.0 Descriptions

3.1 blkcnt_t

LFS The type **blkcnt_t** is defined as an extended signed integral type used for file block counts. See p14 of [3].

In the LP64 compilation environment, **blkcnt_t** will be implemented as a **long**.

3.2 blkcnt64_t

LFS The type **blkcnt64_t** is defined as an extended signed integral type used for 64-bit file block counts and is part of the transitional extensions for large files. See p19-20 of [3].

This type is unchanged.

3.3 clock_t

POSIX **clock_t** is defined to be capable of representing all integer values from 0 to the number of ticks in 24 hours. See p34, and p366 of [1].

X/Open **clock_t** used for system times in clock ticks. See p838 of [2].

This type is used for system times in clock ticks by **times(2)** and **clock(3C)**, and used as time in Hz in the kernel. The **times(2)** system call returns the calling process' system and user times and accumulated system and user times for its children in clock ticks. If **clock_t** is a signed 32-bit integer, it will overflow in a little more than a year if there are 60 ticks per second or less than a year if there are 100. For the accumulated children times, it may even take less time to overflow. Depending on the average uptime, this might be a concern, but one year is a pretty long time. Besides, it is probably better to use **proc(4)** to monitor process times anyway.

Overflow will occur much sooner with the C library routine, **clock(3C)**, because it returns the amount of CPU time used in microseconds since the first call to it in the calling process. It will overflow a signed 32-bit **clock_t** in about 2147 seconds which is about 36 minutes. This is an existing problem, and it seems more like a bug than a feature to use **clock_t** for microseconds instead of ticks as it was intended.

The relatively small amount of time that it takes to overflow **clock_t** with **clock(3C)** is troubling, but DEC and SGI both left **clock_t** as an **int**. For now, **clock_t** will be left as a 32-bit quantity.

3.4 daddr_t

The type **daddr_t** is the disk block address type. It is commonly used as a 512-byte block number, so it can address file systems and devices up to 1 terabyte.

Only one system call uses **daddr_t** — **ustat(2)** returns a data structure containing a field which is the total number of free blocks on a filesystem. However, there are several **dkio(7I)**, **fdio(7I)**, **hdio(7I)**, and **mtio(7I)** ioctls which embed **daddr_t**, and of course it is used in persistent data structures saved to disk and tape, as well as the programs that manipulate them *e.g.* **ufsdump**, **ufsrestore**, **format**, **fsck**, **mkfs**, **tar**, *etc.*

The Large Files Summit has made allowances for 64-bit file block counts *via* **stat(2)** by defining **blkcnt_t**. DEC has left **daddr_t** as 32 bits and SGI has made it 64 bits.

For now, we intend to leave **daddr_t** as 32 bits, but its type implementation should change from **long** to **int** to be 32 bits in both ILP32 and LP64 environments. Since 1 terabyte filesystems can be constructed by concatenation today, we think that this problem will need to be solved in the 32-bit world anyway, so any disk block addresses that need to be more than 32 bits should use **diskaddr_t**.

3.5 dev_t

POSIX The type **dev_t** is used for device numbers. It must be arithmetic and may be made big enough to accommodate host-locality considerations of networked systems. See p32, p321 of [1].

X/Open **dev_t** is used for device IDs. See p838 of [2].

The existing device number is partitioned into a 14-bit major number space, and an 18-bit minor number space. There does not appear to be any pressing need to grow **dev_t** from 32 to 64 bits now.

Some reviewers of this document have observed that a device driver which partitions the minor number into spaces to simplify minor number decoding may well be able to use a bigger **dev_t**. But that convenience has to be balanced against the compatibility problems created for 32-bit applications. Both DEC and SGI have kept **dev_t** 32 bits in size.

We recommend that **dev_t** remain 32 bits and its implementation be changed from **ulong_t** to **unsigned int** to be 32 bits in both ILP32 and LP64 worlds.

3.6 diskaddr_t

The type **diskaddr_t** is the 64-bit disk block address type used by Sun. It is used by the **quot** command to represent a block number. This seems to be the better type on which to base 64-bit disk block numbers in both ILP32 and LP64 environments.

3.7 fsblkcnt_t

LFS The type **fsblkcnt_t** is defined as an extended unsigned integral type and used for file system block counts. See p14 of [3].

This type was introduced for large files. It should become a 64-bit quantity in the LP64 compilation environment.

3.8 fsblkcnt64_t

LFS The type **fsblkcnt64_t** is defined as an extended unsigned integral type used for 64-bit file system block counts. It is part of the transitional extensions for large files. See p19-20 of [3].

This type is unchanged.

3.9 fsfilcnt_t

LFS The type **fsfilcnt_t** is defined as an extended unsigned integral type used for file system file counts. See p14 of [3].

This type was introduced for large files. It should become a 64-bit quantity in the LP64 compilation environment.

3.10 fsfilcnt64_t

LFS The type **fsfilcnt64_t** is defined as an extended unsigned integral type used for 64-bit file system file counts. It is part of the transitional extensions for large files. See p19-20 of [3].

This type is unchanged.

3.11 gid_t

POSIX **gid_t** is the type used for group IDs. See p32 of [1].

X/Open **gid_t** is the type used for group IDs. See p838 of [2].

It should remain 32 bits in size, and its implementation should be changed from **long** to **int** so that it holds 32 bits in both ILP32 and LP64 compilation environments. For justification, see **uid_t** below.

3.12 hostid_t

The type **hostid_t** is used to uniquely define a particular node on an RFS network. It seems to be obsolete cruft, so we propose to omit it altogether from both compilation environments.

3.13 id_t

X/Open The type **id_t** is used as a general identifier and can be used to contain at least a **pid_t**, **uid_t**, or a **gid_t**. See p838 of [2].

It is used for various kinds of identifiers such as process, process group, session, scheduling class, user, and group IDs. The actual type must be the same for all, since some system calls, *e.g.* **sigsend(2)**, take argu-

ments that may be any of these types. The enumeration type **idtype_t**, defined in `<sys/procset.h>`, is used to indicate what type of ID is being specified.

Since it is a general form of system identifier, it should track the type of the things that it represents. As long as nothing it represents changes, it should remain 32 bits and have its implementation changed from **long** to **int** to be 32 bits in both ILP32 and LP64.

3.14 ino_t

POSIX **ino_t** is the type used for file serial numbers. See p32 of [1]. File serial numbers are defined to be a per file system unique identifier for a file. See p15 of [1].

X/Open **ino_t** is the type used for file serial numbers. See p838 of [2].

It should grow to 64 bits, both because of the LFS specification, and because NFS3 and XFS both use 64-bit inode numbers already.

3.15 ino64_t

LFS The type **ino64_t** is defined as an extended unsigned integral type used for file serial numbers and is part of the transitional extensions for large files. See p19-20 of [3].

This type is unchanged.

3.16 len_t

The type **len_t** is used for 64-bit lengths.

3.17 major_t

The type **major_t** is used for the major part of the device number. It should remain 32 bits, and its implementation should change from **ulong_t** to **unsigned int** so it will hold 32 bits in both ILP32 and LP64.

3.18 minor_t

The type **minor_t** is used for the minor part of the device number. It should remain 32 bits, and its implementation should change from **ulong_t** to **unsigned int** so it will hold 32 bits in both ILP32 and LP64.

3.19 mode_t

POSIX The type **mode_t** is used for some file attributes like file type and access permissions. In both 4.3BSD and SVID, it is an **unsigned short** and historically only the low-order 16 bits are significant. See p32 and p321 of [1].

X/Open The type **mode_t** is used for file attributes. See p838 of [2].

There does not appear to be any reason to change its size, but its implementation needs to be changed from **ulong_t** to **unsigned int** so it will hold 32 bits in both ILP32 and LP64.

3.20 nlink_t

POSIX The type **nlink_t** is used for link counts. The link count to be the number of directories referring to a given file. It is part of the **stat** structure as the number of links for the file. Originally, it was introduced to replace **short** for the **st_nlink** field in the **stat** structure because **short** was thought to be too small. See p16, p32 and p321 of [1].

X/Open The type **nlink_t** is used for link counts. See p838 of [2].

The current 32-bit version of **nlink_t** seems adequate in size, so there is no need to change it. Its implementation should be changed from **ulong_t** to **unsigned int** so it will hold 32 bits in both ILP32 and LP64.

3.21 off_t

POSIX The type **off_t** is used for file sizes and offsets. See p32 of [1].

X/Open The type **off_t** is used for file sizes and offsets. See p838 of [2].

In the LP64 compilation environment, **off_t** should grow to 64 bits for system to grow naturally *e.g.* the 64-bit system call **lseek(2)** will naturally take a 64-bit file offset.

3.22 off64_t

LFS The type **off64_t** is used for file sizes and is part of the transitional extensions for large files. See p19 of [3].

This types is unchanged.

3.23 offset_t

The type **offset_t** is used to represent a 64-bit file offset. It is intended to be the file offset type used within the kernel, though it has also escaped from the kernel into the return value of, and an argument to **llseek(2)**. It should remain unchanged as a 64-bit quantity.

3.24 paddr_t

The type **paddr_t** is the physical address type. It has been used in a number of confusing ways.

Since it is so rarely used correctly to mean “a physical address”, we propose to omit it altogether from the LP64 compilation environment.

3.25 pid_t

POSIX **pid_t** is used for process IDs and process group IDs. See p32 of [1].

X/Open As above. See p838 of [2].

Both DEC and SGI have left this type as 32 bits.

It is currently implemented using **long** and should be an **int** so that it will hold 32 bits for both ILP32 and LP64 compilation environments.

3.26 ptrdiff_t

X/Open Signed integral type of the result of subtracting two pointers

ptrdiff_t should be implemented using **intptr_t** instead of **int**.

3.27 rlim_t

X/Open **rlim_t** is an unsigned integral type used for limit values

This will be implemented as a 64-bit quantity in the LP64 compilation environment.

3.28 rlim64_t

LFS The type **rlim64_t** must be an extended unsigned arithmetic type that can represent correctly a non-negative value of an **off64_t**

This type is unchanged.

3.29 `size_t`

X/Open The type `size_t` is used for the sizes of objects. See p838 of [2].

For the LP64 compilation environment, it needs to be a 64-bit quantity because it has to be able to represent the biggest size in the system. Many functions such as `malloc(3c)` need this capability from `size_t`. Often, `size_t` is used to hold the value of the C `sizeof` operator, which is also 64 bits.

So, `size_t` should be a 64-bit quantity and its type should be changed from `uint_t` to `unsigned long` to be 32 bits in ILP32 and 64 bits in LP64.

3.30 `speed_t`

POSIX `speed_t` shall be an unsigned integral type. See p175 of [1].

X/Open Used for terminal baud rates. See p846 of [2].

This type should stay 32-bit and should be implemented using `unsigned int`.

3.31 `ssize_t`

POSIX The type `ssize_t` is used by functions that return a count of bytes (memory space) or an error indication and should be capable of holding values in the range from -1 to `{SSIZE_MAX}` inclusive. See p32 of [1].

X/Open `ssize_t` is used for count of bytes or an error indication. See p838 of [2].

Like `size_t`, `ssize_t` should be a 64-bit quantity for all the same reasons. Its implementation needs to be changed from `int` to `long` for it to grow naturally from 32 bits in ILP32 to 64 bits in LP64.

3.32 `tflag_t`

POSIX `tflag_t` shall be an unsigned integral type. See p175 of [1].

X/Open Used for terminal modes. See p846 of [2].

This type should stay 32-bit and should be implemented as an `unsigned int`.

3.33 `time_t`

X/Open The type `time_t` is used for time in seconds. See p838 of [2].

As a 32-bit quantity, it will overflow in the year 2038.

However changing the size of the representation will cause more disruption to existing usage of `time_t`.

It is very widely used. The file system stores it on disk and `ufsdump` puts it on tape. Both DEC and SGI have left it as a 32-bit quantity.

For these reasons, we do not propose to change the size of the representation of `time_t`, but its type will be changed from `long` to `int` for it to be 32 bits in both ILP32 and LP64.

3.34 `u_offset_t`

The type `u_offset_t` is used for an unsigned version of `offset_t` and introduced by the Large Files project. It should remain 64 bits in the LP64 compilation environment.

3.35 `uid_t`

POSIX The `uid_t` type is used for user IDs. See p32 of [1]. A user ID is a nonnegative integer that is used to identify a system user. See p25, p32 and p323 of [1].

X/Open The **uid_t** type is used for user IDs. See p838 of [2].

Both **uid_t** and **gid_t** are widely used in many interfaces which expect them to be 32 bits or smaller. They are embedded in many media formats created by archive utilities. Apart from **ufsdump**, only the **cpio -x** format allows full 32-bit uids and gids to be saved and restored. They are also stored on-disk in the file system and passed over the wire by NFS3 which only allows them to be unsigned 32-bit quantities. In addition, the NIS protocol only allows **uid_t** and **gid_t** to be 32-bit. Changing to 64 bits would also grow the **cred_t** structure in the kernel and create problems dealing with 32-bit and 64-bit applications issuing the **{s,g}et{r,e}{u,g}id(2)** family of system calls. Both DEC and SGI have left them as 32-bit quantities.

Since many things expect them to be 32 bits and there does not seem to be any need to grow them, **uid_t** and **gid_t** should stay 32 bits. Currently, they are implemented using **long** but should be of type **int** to be 32 bits in both ILP32 and LP64. They should stay as signed quantities because interfaces like **setreuid(2)** use a value of -1 to mean “don’t set this value”.

3.36 wchar_t

X/Open Integral type whose range of values can represent distinct wide character codes for all members of the largest character set specified among the locales supported by the compilation environment ...

wchar_t should stay as a 32-bit quantity, but be implemented using **int** instead of **long**

4.0 Issues

4.1 struct timeval

X/Open The **tv_usec** field of **struct timeval** is defined as an explicit **long**. See p834 of [2].

This is an unfortunate choice. The field only has to contain values between 0 and 999,999. Interestingly, DEC has made this field be an **int**, even though they claim compliance to a standard that mandates it being a **long**. We believe that this should be an **int** or a **useconds_t**.

4.2 long long

Since it is not an ANSI type, explicit use of the **long long** in interface definitions should be avoided. The LP64 compilation environment reduces the dependency on **long long** even further. For LP64 compilation environments, **long** or **int64_t** can be used instead of **long long**, though derived types like **off_t** should be used wherever possible to keep code as portable as possible.

4.3 Fixed-Width vs. Natural Types

When should fixed-width types be used instead of natural types, *e.g.* **int32_t** vs. **int**? Often, this boils down to issues of style rather than substance. Here’s some guidelines on when to use one or the other. Use explicit fixed-width types for:

- Over-the-wire protocols, though XDR is often a better way.
- Shared memory protocols
- CPU and device driver registers
- Providing ILP32 compatibility services from an LP64 program

but not for things that are naturally “an integer” *e.g.*

- simple counters in loops

- file descriptors

Also note that there is a performance penalty for simply making everything be 64-bit; while the CPU can manipulate 32-bit and 64-bit quantities just as quickly, moving more data takes longer, it fills the cache faster, and occupies more space on disk.

5.0 Summary

The net effect of the changes discussed in this paper is to obsolete **hostid_t** and **paddr_t**, grow **size_t**, **ssize_t** and **ptrdiff_t** to 64-bit quantities, and to co-opt the type changes of `_FILE_OFFSET_BITS == 64`, but otherwise leave the other system types as 32-bit quantities in the LP64 compilation environment.

6.0 References

- [1] IEEE Std 1003.1b-1993, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 1: Real-time — Extension [C Language]
- [2] X/Open CAE Specification — System Interfaces and Headers — Issue 4, Version 2
- [3] X/Open Submission: Adding Support for Arbitrary File Sizes to the Single UNIX Specification, March 20, 1996. See http://www.sas.com/standards/large.file/x_open/20Mar96.html.