NAME

setbuf, **setbuffer**, **setlinebuf**, **setvbuf** - stream buffering operations

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <stdio.h>

void
setbuf(*FILE * restrict stream, char * restrict buf*);

void
setbuffer(*FILE *stream, char *buf, int size*);

int
setlinebuf(*FILE *stream*);

int
setvbuf(*FILE * restrict stream, char * restrict buf, int mode, size_t size*);

DESCRIPTION

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a newline is output or input is read from any stream attached to a terminal device (typically stdin). The function fflush(3) may be used to force the block out early. (See fclose(3).)

Normally all files are block buffered. When the first I/O operation occurs on a file, malloc(3) is called, and an optimally-sized buffer is obtained. If a stream refers to a terminal (as stdout normally does) it is line buffered. The standard error stream stderr is always unbuffered. Note that these defaults may be altered using the stdbuf(1) utility.

The **setvbuf**() function may be used to alter the buffering behavior of a stream. The *mode* argument must be one of the following three macros:

_IONBF
: unbuffered

_IOLBF

line buffered

_IOFBF fully buffered

The *size* argument may be given as zero to obtain deferred optimal-size buffer allocation as usual. If it is not zero, then except for unbuffered files, the *buf* argument should point to a buffer at least *size* bytes long; this buffer will be used instead of the current buffer. If *buf* is not NULL, it is the caller's responsibility to free(3) this buffer after closing the stream. (If the *size* argument is not zero but *buf* is NULL, a buffer of the given size will be allocated immediately, and released on close. This is an extension to ANSI C; portable code should use a size of 0 with any NULL buffer.)

The **setvbuf**() function may be used at any time, but may have peculiar side effects (such as discarding input or flushing output) if the stream is ''active''. Portable applications should call it only once on any given stream, and before any I/O is performed.

The other three calls are, in effect, simply aliases for calls to **setvbuf**(). Except for the lack of a return value, the **setbuf**() function is exactly equivalent to the call

```
setvbuf(stream, buf, buf ? _IOFBF : _IONBF, BUFSIZ);
```

The **setbuffer**() function is the same, except that the size of the buffer is up to the caller, rather than being determined by the default BUFSIZ. The **setlinebuf**() function is exactly equivalent to the call:

```
setvbuf(stream, (char *)NULL, _IOLBF, 0);
```

RETURN VALUES

The **setvbuf**() function returns 0 on success, or EOF if the request cannot be honored (note that the stream is still functional in this case).

The **setlinebuf**() function returns what the equivalent **setvbuf**() would have returned.

SEE ALSO

stdbuf(1), fclose(3), fopen(3), fread(3), malloc(3), printf(3), puts(3)

STANDARDS

The **setbuf**() and **setvbuf**() functions conform to ISO/IEC 9899:1990 ("ISO C90").

HISTORY

The **setbuf**() function first appeared in Version 7 AT&T UNIX. The **setbuffer**() function first appeared in 4.1cBSD. The **setlinebuf**() function first appeared in 4.2BSD. The **setvbuf**() function first appeared

in 4.4BSD.

BUGS

setbuf() usually uses a suboptimal buffer size and should be avoided.