NAME

getcontext, setcontext - get or set the user context

SYNOPSIS

#include <ucontext.h>

int getcontext(ucontext_t *_ucp_);
int setcontext(const ucontext_t *_ucp_);

DESCRIPTION

In a System V-like environment, one has the two types _mcontext_t_ and _ucontext_t_ defined in _<ucontext.h>_ and the four functions **getcontext**(), **setcontext**(), makecontext(3), and swapcontext(3) that allow user-level context switching between multiple threads of control within a process.

The _mcontext_t_ type is machine-dependent and opaque. The _ucontext_t_ type is a structure that has at least the following fields:

```
typedef struct ucontext {
struct ucontext *uc_link;
sigset_t uc_sigmask;
stack_t uc_stack;
mcontext_t uc_mcontext;
} ucontext_t;
```

with _sigset_t_ and _stack_t_ defined in _<signal.h>_. Here _uc_link_ points to the context that will be resumed when the current context terminates (in case the current context was created using makecontext(3)), _uc_sigmask_ is the set of signals blocked in this context (see sigprocmask(2)), _uc_stack_ is the stack used by this context (see sigaltstack(2)), and _uc_mcontext_ is the machine-specific representation of the saved context, that includes the calling thread's machine registers.

The function **getcontext**() initializes the structure pointed at by _ucp_ to the currently active context.

The function **setcontext**() restores the user context pointed at by _ucp_. A successful call does not return. The context should have been obtained by a call of **getcontext**(), or makecontext(3), or passed as third argument to a signal handler.

If the context was obtained by a call of **getcontext**(), program execution continues as if this call just returned.

If the context was obtained by a call of makecontext(3), program execution continues by a call to the function _func_ specified as the second argument of that call to makecontext(3). When the function _func_ returns, we continue with the _uc_link_ member of the structure _ucp_ specified as the first argument of that call to makecontext(3). When this member is NULL, the thread exits.

If the context was obtained by a call to a signal handler, then old standard text says that program execution continues with the program instruction following the instruction interrupted by the signal. However, this sentence was removed in SUSv2, and the present verdict is the result is unspecified.

RETURN VALUE

When successful, **getcontext**() returns 0 and **setcontext**() does not return. On error, both return -1 and set _errno_ appropriately.

ERRORS

None defined.

ATTRIBUTES

Multithreading (see pthreads(7))

The **getcontext**() and **setcontext**() functions are thread-safe.

CONFORMING TO

SUSv2, POSIX.1-2001. POSIX.1-2008 removes the specification of **getcontext**(), citing portability issues, and recommending that applications be rewritten to use POSIX threads instead.

NOTES

The earliest incarnation of this mechanism was the setjmp(3)/ longjmp(3) mechanism. Since that does not define the handling of the signal context, the next stage was the sigsetjmp(3)/ siglongjmp(3) pair. The present mechanism gives much more control. On the other hand, there is no easy way to detect whether a return from **getcontext**() is from the first call, or via a **setcontext**() call. The user has to invent her own bookkeeping device, and a register variable won't do since registers are restored.

When a signal occurs, the current user context is saved and a new context is created by the kernel for the signal handler. Do not leave the handler using longjmp(3): it is undefined what would happen with contexts. Use siglongjmp(3) or **setcontext**() instead.

SEE ALSO

sigaction(2), sigaltstack(2), sigprocmask(2), longjmp(3), makecontext(3), sigsetjmp(3)

COLOPHON

This page is part of release 3.74 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at http://www.kernel.org/doc/man-pages/.