NAME

unw_resume -- resume execution in a particular stack frame

SYNOPSIS

```
#include <libunwind.h>

int unw_resume(unw_cursor_t *cp);
```

DESCRIPTION

The unw_resume() routine resumes execution at the stack frame identified by cp. The behavior of this routine differs slightly for local and remote unwinding.

For local unwinding, unw_resume() restores the machine state and then directly resumes execution in the target stack frame. Thus unw_resume() does not return in this case. Restoring the machine state normally involves restoring the ‘‘preserved’’ (callee-saved) registers. However, if execution in any of the stack frames younger (more deeply nested) than the one identified by cp was interrupted by a signal, then unw_resume() will restore all registers as well as the signal mask. Attempting to call unw_resume() on a cursor which identifies the stack frame of another thread results in undefined behavior (e.g., the program may crash).

For remote unwinding, unw_resume() installs the machine state identified by the cursor by calling the access_reg and access_fpreg accessor callbacks as needed. Once that is accomplished, the resume accessor callback is invoked. The unw_resume routine then returns normally (that is, unlikely for local unwinding, unw_resume will always return for remote unwinding).

Most platforms reserve some registers to pass arguments to exception handlers (e.g., IA-64 uses r15-r18 for this purpose). These registers are normally treated like ‘‘scratch’’ registers. However, if libunwind is used to set an exception argument register to a particular value (e.g., via unw_set_reg()), then unw_resume() will install this value as the contents of the register. In other words, the exception handling arguments are installed even in cases where normally only the ‘‘preserved’’ registers are restored.

Note that unw_resume() does *not* invoke any unwind handlers (aka, ‘‘personality routines’’). If a program needs this, it will have to do so on its own by obtaining the unw_proc_info_t of each unwound frame and appropriately processing its unwind handler and language-specific data area (lsda). These steps are generally dependent on the target platform and are regulated by the processor-specific ABI (application-binary interface).

RETURN VALUE

For local unwinding, unw_resume() does not return on success. For remote unwinding, it returns 0 on

success. On failure, the negative value of one of the errors below is returned.

THREAD AND SIGNAL SAFETY

unw_resume() is thread-safe. If cursor cp is in the local address-space, this routine is also safe to use from a signal handler.

ERRORS

UNW_EUNSPEC
: An unspecified error occurred.

UNW_EBADREG
: A register needed by unw_resume() wasn't accessible.

UNW_EINVALIDIP
: The instruction pointer identified by cp is not valid.

UNW_BADFRAME
: The stack frame identified by cp is not valid.

SEE ALSO

libunwind(3libunwind), unw_set_reg(3libunwind), sigprocmask(2)

AUTHOR

David Mosberger-Tang
Email: **dmosberger@gmail.com**
WWW: **http://www.nongnu.org/libunwind/**.