

**NAME**

unw\_create\_addr\_space -- create address space for remote unwinding

**SYNOPSIS**

```
#include <libunwind.h>
```

```
unw_addr_space_t unw_create_addr_space(unw_accessors_t *ap, int byteorder);
```

**DESCRIPTION**

The `unw_create_addr_space()` routine creates a new unwind address-space and initializes it based on the call-back routines passed via the `ap` pointer and the specified `byteorder`. The call-back routines are described in detail below. The `byteorder` can be set to 0 to request the default byte-order of the unwind target. To request a particular byte-order, `byteorder` can be set to any constant defined by `<endian.h>`. In particular, `__LITTLE_ENDIAN` would request little-endian byte-order and `__BIG_ENDIAN` would request big-endian byte-order. Whether or not a particular byte-order is supported depends on the target platform.

**CALL-BACK ROUTINES**

Libunwind uses a set of call-back routines to access the information it needs to unwind a chain of stack-frames. These routines are specified via the `ap` argument, which points to a variable of type `unw_accessors_t`. The contents of this variable is copied into the newly-created address space, so the variable must remain valid only for the duration of the call to `unw_create_addr_space()`.

The first argument to every call-back routine is an address-space identifier (`as`) and the last argument is an arbitrary, application-specified void-pointer (`arg`). When invoking a call-back routine, libunwind sets the `as` argument to the address-space on whose behalf the invocation is made and the `arg` argument to the value that was specified when `unw_init_remote(3)` was called.

The synopsis and a detailed description of every call-back routine follows below.

**CALL-BACK ROUTINE SYNOPSIS**

```
int find_proc_info(unw_addr_space_t as,
                  unw_word_t ip, unw_proc_info_t *pip,
                  int need_unwind_info, void *arg);
void put_unwind_info(unw_addr_space_t as,
                    unw_proc_info_t *pip, void *arg);
int get_dyn_info_list_addr(unw_addr_space_t as,
                          unw_word_t *dilap, void *arg);
int access_mem(unw_addr_space_t as,
              unw_word_t addr, unw_word_t *valp,
```

```

        int write, void *arg);
int access_reg(unw_addr_space_t as,
               unw_regnum_t regnum, unw_word_t *valp,
               int write, void *arg);
int access_fpreg(unw_addr_space_t as,
                 unw_regnum_t regnum, unw_fpreg_t *fpvalp,
                 int write, void *arg);
int resume(unw_addr_space_t as,
           unw_cursor_t *cp, void *arg);
int get_proc_name(unw_addr_space_t as,
                  unw_word_t addr, char *bufp,
                  size_t buf_len, unw_word_t *offp,
                  void *arg);

```

## FIND\_PROC\_INFO

Libunwind invokes the `find_proc_info()` call-back to locate the information need to unwind a particular procedure. The `ip` argument is an instruction-address inside the procedure whose information is needed. The `pip` argument is a pointer to the variable used to return the desired information. The type of this variable is `unw_proc_info_t`. See `unw_get_proc_info(3)` for details. Argument `need_unwind_info` is zero if the call-back does not need to provide values for the following members in the `unw_proc_info_t` structure: `format`, `unwind_info_size`, and `unwind_info`. If `need_unwind_info` is non-zero, valid values need to be returned in these members. Furthermore, the contents of the memory addressed by the `unwind_info` member must remain valid until the info is released via the `put_unwind_info` call-back (see below).

On successful completion, the `find_proc_info()` call-back must return zero. Otherwise, the negative value of one of the `unw_error_t` error-codes may be returned. In particular, this call-back may return `-UNW_ESTOPUNWIND` to signal the end of the frame-chain.

## PUT\_UNWIND\_INFO

Libunwind invokes the `put_unwind_info()` call-back to release the resources (such as memory) allocated by a previous call to `find_proc_info()` with the `need_unwind_info` argument set to a non-zero value. The `pip` argument has the same value as the argument of the same name in the previous matching call to `find_proc_info()`. Note that libunwind does *not* invoke `put_unwind_info` for calls to `find_proc_info()` with a zero `need_unwind_info` argument.

## GET\_DYN\_INFO\_LIST\_ADDR

Libunwind invokes the `get_dyn_info_list_addr()` call-back to obtain the address of the head of the dynamic unwind-info registration list. The variable stored at the returned address must have a type of `unw_dyn_info_list_t` (see `_U_dyn_register(3)`). The `dliap` argument is a pointer to a variable of type

`unw_word_t` which is used to return the address of the dynamic unwind-info registration list. If no dynamic unwind-info registration list exist, the value pointed to by `dliap` must be cleared to zero. Libunwind will cache the value returned by `get_dyn_info_list_addr()` if caching is enabled for the given address-space. The cache can be cleared with a call to `unw_flush_cache()`.

On successful completion, the `get_dyn_info_list_addr()` call-back must return zero. Otherwise, the negative value of one of the `unw_error_t` error-codes may be returned.

## ACCESS\_MEM

Libunwind invokes the `access_mem()` call-back to read from or write to a word of memory in the target address-space. The address of the word to be accessed is passed in argument `addr`. To read memory, libunwind sets argument `write` to zero and `valp` to point to the word that receives the read value. To write memory, libunwind sets argument `write` to a non-zero value and `valp` to point to the word that contains the value to be written. The word that `valp` points to is always in the byte-order of the host-platform, regardless of the byte-order of the target. In other words, it is the responsibility of the call-back routine to convert between the target's and the host's byte-order, if necessary.

On successful completion, the `access_mem()` call-back must return zero. Otherwise, the negative value of one of the `unw_error_t` error-codes may be returned.

## ACCESS\_REG

Libunwind invokes the `access_reg()` call-back to read from or write to a scalar (non-floating-point) CPU register. The index of the register to be accessed is passed in argument `regnum`. To read a register, libunwind sets argument `write` to zero and `valp` to point to the word that receives the read value. To write a register, libunwind sets argument `write` to a non-zero value and `valp` to point to the word that contains the value to be written. The word that `valp` points to is always in the byte-order of the host-platform, regardless of the byte-order of the target. In other words, it is the responsibility of the call-back routine to convert between the target's and the host's byte-order, if necessary.

On successful completion, the `access_reg()` call-back must return zero. Otherwise, the negative value of one of the `unw_error_t` error-codes may be returned.

## ACCESS\_FPREG

Libunwind invokes the `access_fpreg()` call-back to read from or write to a floating-point CPU register. The index of the register to be accessed is passed in argument `regnum`. To read a register, libunwind sets argument `write` to zero and `fpvalp` to point to a variable of type `unw_fpreg_t` that receives the read value. To write a register, libunwind sets argument `write` to a non-zero value and `fpvalp` to point to the variable of type `unw_fpreg_t` that contains the value to be written. The word that `fpvalp` points to is always in the byte-order of the host-platform, regardless of the byte-order of the target. In other words, it is the responsibility of the call-back routine to convert between the target's and the host's byte-order,

if necessary.

On successful completion, the `access_fpreg()` call-back must return zero. Otherwise, the negative value of one of the `unw_error_t` error-codes may be returned.

## RESUME

Libunwind invokes the `resume()` call-back to resume execution in the target address space. Argument `cp` is the unwind-cursor that identifies the stack-frame in which execution should resume. By the time libunwind invokes the resume call-back, it has already established the desired machine- and memory-state via calls to the `access_reg()`, `access_fpreg`, and `access_mem()` call-backs. Thus, all the call-back needs to do is perform whatever action is needed to actually resume execution.

The resume call-back is invoked only in response to a call to `unw_resume(3)`, so applications which never invoke `unw_resume(3)` need not define the resume callback.

On successful completion, the `resume()` call-back must return zero. Otherwise, the negative value of one of the `unw_error_t` error-codes may be returned. As a special case, when resuming execution in the local address space, the call-back will not return on success.

## GET\_PROC\_NAME

Libunwind invokes the `get_proc_name()` call-back to obtain the procedure-name of a static (not dynamically generated) procedure. Argument `addr` is an instruction-address within the procedure whose name is to be obtained. The `bufp` argument is a pointer to a character-buffer used to return the procedure name. The size of this buffer is specified in argument `buf_len`. The returned name must be terminated by a NUL character. If the procedure's name is longer than `buf_len` bytes, it must be truncated to `buf_len-1` bytes, with the last byte in the buffer set to the NUL character and `-UNW_ENOMEM` must be returned. Argument `offp` is a pointer to a word which is used to return the byte-offset relative to the start of the procedure whose name is being returned. For example, if procedure `foo()` starts at address `0x40003000`, then invoking `get_proc_name()` with `addr` set to `0x40003080` should return a value of `0x80` in the word pointed to by `offp` (assuming the procedure is at least `0x80` bytes long).

On successful completion, the `get_proc_name()` call-back must return zero. Otherwise, the negative value of one of the `unw_error_t` error-codes may be returned.

## RETURN VALUE

On successful completion, `unw_create_addr_space()` returns a non-NULL value that represents the newly created address-space. Otherwise, NULL is returned.

## THREAD AND SIGNAL SAFETY

`unw_create_addr_space()` is thread-safe but *not* safe to use from a signal handler.

**SEE ALSO**

`_U_dyn_register(3)`, `libunwind(3)`, `unw_destroy_addr_space(3)`, `unw_get_proc_info(3)`,  
`unw_init_remote(3)`, `unw_resume(3)`

**AUTHOR**

David Mosberger-Tang

Email: [dmosberger@gmail.com](mailto:dmosberger@gmail.com)

WWW: <http://www.nongnu.org/libunwind/>.