



Linux Ubuntu 26.04 Manual Pages on command 'makecontext.3'

\$ man makecontext.3

makecontext(3) Library Functions Manual makecontext(3)

NAME

makecontext, swapcontext - manipulate user context

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <ucontext.h>

void makecontext(ucontext_t *ucp, typeof(void (int arg0, ...)) *func,
                int argc, ...);

int swapcontext(ucontext_t *restrict oucp,
               const ucontext_t *restrict ucp);
```

DESCRIPTION

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

The `makecontext()` function modifies the context pointed to by `ucp` (which was obtained from a call to `getcontext(3)`). Before invoking `makecontext()`, the caller must allocate a new stack for this context and assign its address to `ucp->uc_stack`, and define a successor context and assign its address to `ucp->uc_link`.

When this context is later activated (using `setcontext(3)` or `swapcontext()`) the function `func` is called, and passed the series of integer (`int`) arguments that follow `argc`; the caller must specify the number of these arguments in `argc`. When this function returns, the

The `swapcontext()` function saves the current context in the structure pointed to by `oucp`, and then activates the context pointed to by `ucp`.

When successful, `swapcontext()` does not return. (But we may return later, in case `oucp` is activated, in which case it looks like `swapcontext()` returns 0.) On error, `swapcontext()` returns -1 and sets `errno` to indicate the error.

ENOMEM Insufficient stack space left.

For an explanation of the terms used in this section, see `attributes(7)`.

? Interface	? Attribute	? Value	
-------------	-------------	---------	--

? makecontext() ? Thread safety ? MT-Safe race:ucp ?

? swapcontext() ? Thread safety ? MT-Safe race:oucp race:ucp ?

None.

glibc 2.1. SUSv2, POSIX.1-2001. Removed in POSIX.1-2008, citing portability issues, and recommending that applications be rewritten to use POSIX threads instead.

The interpretation of `ucp->uc_stack` is just as in `sigaltstack(2)`, namely, this struct contains the start and length of a memory area to be used as the stack, regardless of the direction of growth of the stack. Thus, it is not necessary for the user program to worry about this direction.

text(), to permit this on some 64-bit architectures (e.g., x86-64).

EXAMPLES

The example program below demonstrates the use of `getcontext(3)`, `makecontext()`, and `swapcon?text()`. Running the program produces the following output:

```
$ ./a.out

main: swapcontext(&uctx_main, &uctx_func2)

func2: started

func2: swapcontext(&uctx_func2, &uctx_func1)

func1: started

func1: swapcontext(&uctx_func1, &uctx_func2)

func2: returning

func1: returning

main: exiting
```

Program source

```
#include <err.h>

#include <stdio.h>

#include <stdlib.h>

#include <ucontext.h>

static ucontext_t uctx_main, uctx_func1, uctx_func2;

static void

func1(void)

{

    printf("%s: started\n", __func__);

    printf("%s: swapcontext(&uctx_func1, &uctx_func2)\n", __func__);

    if (swapcontext(&uctx_func1, &uctx_func2) == -1)

        err(EXIT_FAILURE, "swapcontext");

    printf("%s: returning\n", __func__);

}

static void

func2(void)

{

    printf("%s: started\n", __func__);

    printf("%s: swapcontext(&uctx_func2, &uctx_func1)\n", __func__);
```

```

    if (swapcontext(&uctx_func2, &uctx_func1) == -1)
        err(EXIT_FAILURE, "swapcontext");
    printf("%s: returning\n", __func__);
}

int
main(int argc, char *argv[])
{
    char func1_stack[16384];
    char func2_stack[16384];
    if (getcontext(&uctx_func1) == -1)
        err(EXIT_FAILURE, "getcontext");
    uctx_func1.uc_stack.ss_sp = func1_stack;
    uctx_func1.uc_stack.ss_size = sizeof(func1_stack);
    uctx_func1.uc_link = &uctx_main;
    makecontext(&uctx_func1, func1, 0);
    if (getcontext(&uctx_func2) == -1)
        err(EXIT_FAILURE, "getcontext");
    uctx_func2.uc_stack.ss_sp = func2_stack;
    uctx_func2.uc_stack.ss_size = sizeof(func2_stack);
    /* Successor context is f1(), unless argc > 1 */
    uctx_func2.uc_link = (argc > 1) ? NULL : &uctx_func1;
    makecontext(&uctx_func2, func2, 0);
    printf("%s: swapcontext(&uctx_main, &uctx_func2)\n", __func__);
    if (swapcontext(&uctx_main, &uctx_func2) == -1)
        err(EXIT_FAILURE, "swapcontext");
    printf("%s: exiting\n", __func__);
    exit(EXIT_SUCCESS);
}

```

SEE ALSO

sigaction(2), sigaltstack(2), sigprocmask(2), getcontext(3), sigsetjmp(3)