NAME

setjmp, longjmp, _setjmp, _longjmp – non-local goto

SYNOPSIS

/usr/ucb/cc [*flag* ...] *file* ...

#include <setjmp.h>

int setjmp(*env***)**
jmp_buf *env***;**

void longjmp(*env***,** *val***)**
jmp_buf *env***;**
int *val***;**

int _setjmp(*env***)**
jmp_buf *env***;**

void _longjmp(*env***,** *val***)**
jmp_buf *env***;**
int *val***;**

DESCRIPTION

setjmp( ) and **longjmp( )** are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

setjmp( ) saves its stack environment in *env* for later use by **longjmp( )**. A normal call to **setjmp( )** returns zero. **setjmp( )** also saves the register environment. If a **longjmp( )** call will be made, the routine which called **setjmp( )** should not return until after the **longjmp( )** has returned control (see below).

longjmp( ) restores the environment saved by the last call of **setjmp( )**, and then returns in such a way that execution continues as if the call of **setjmp( )** had just returned the value *val* to the function that invoked **setjmp( )**; however, if *val* were zero, execution would continue as if the call of **setjmp( )** had returned one. This ensures that a ''return'' from **setjmp( )** caused by a call to **longjmp( )** can be distinguished from a regular return from **setjmp( )**. The calling function must not itself have returned in the interim, otherwise **longjmp( )** will be returning control to a possibly non-existent environment. All memory-bound data have values as of the time **longjmp( )** was called. The CPU and floating-point data registers are restored to the values they had at the time that **setjmp( )** was called. But, because the **register** storage class is only a hint to the C compiler, variables declared as **register** variables may not necessarily be assigned to machine registers, so their values are unpredictable after a **longjmp( )**. This is especially a problem for programmers trying to write machine-independent C routines.

setjmp( ) and **longjmp( )** save and restore the signal mask while **_setjmp( )** and **_longjmp( )** manipulate only the C stack and registers.

None of these functions save or restore any floating-point status or control registers.

EXAMPLES

The following example uses both **setjmp( )** and **longjmp( )** to return the flow of control to the appropriate instruction block:

```
#include <stdio.h>
#include <setjmp.h>
#include <signal.h>
#include <unistd.h>
jmp_buf env; static void signal_handler();

main( ) {
        int returned_from_longjump, processing = 1;
        unsigned int time_interval = 4;
        if ((returned_from_longjump = setjmp(env)) != 0)
            switch (returned_from_longjump)     {
              case SIGINT:
                printf("longjumped from interrupt %d\n",SIGINT);
                break;
              case SIGALRM:
                printf("longjumped from alarm %d\n",SIGALRM);
                break;
            }
        (void) signal(SIGINT, signal_handler);
        (void) signal(SIGALRM, signal_handler);
        alarm(time_interval);
        while (processing)        {
          printf(" waiting for you to INTERRUPT (cntrl-C) ...\n");
          sleep(1);
        }                                  /* end while forever loop */
}

static void signal_handler(sig)
int sig; {
        switch (sig)     {
          case SIGINT:              ...   /* process for interrupt */
                                    longjmp(env,sig);
                                      /* break never reached */
          case SIGALRM:             ...   /* process for alarm */
                                    longjmp(env,sig);
                                      /* break never reached */
          default:                  exit(sig);
        }
}
```

When this example is compiled and executed, and the user sends an interrupt signal, the output will be:

longjumped from interrupt

Additionally, every 4 seconds the alarm will expire, signalling this process, and the output will be:

longjumped from alarm

SEE ALSO

cc(1B), **sigvec**(3B), **setjmp**(3C), **signal**(3C)

NOTES

Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

BUGS

setjmp( ) does not save the current notion of whether the process is executing on the signal stack. The result is that a **longjmp( )** to some place on the signal stack leaves the signal stack state incorrect.

On some systems **setjmp( )** also saves the register environment. Therefore, all data that are bound to registers are restored to the values they had at the time that **setjmp( )** was called. All memory-bound data have values as of the time **longjmp( )** was called. However, because the **register** storage class is only a hint to the C compiler, variables declared as **register** variables may not necessarily be assigned to machine registers, so their values are unpredictable after a **longjmp( )**. When using compiler options that specify automatic register allocation (see **cc**(1B)), the compiler will not attempt to assign variables to registers in routines that call **setjmp( )**.

longjmp( ) never causes **setjmp( )** to return zero, so programmers should not depend on **longjmp( )** being able to cause **setjmp( )** to return zero.