```
--- c:\asgn1_net\setjmp.c ----------------------------------------------------
     1: /* setjmp.c: This program sets many variables, does a setjmp, and
     2:  * then does a longjmp to see what gets restored
     3:  */
     4:
     5: #include <stdio.h>
     6: #include <signal.h>
     7: #include <setjmp.h>
     8: #include <stdlib.h>
     9: #include <float.h>
    10: #include <math.h>
    11: #include <string.h>
    12:
    13: jmp_buf mark;              /* setjmp state data structure */
```

This header code sets the stack for our new procedure (main). First we save the old stack base pointer (bp) on the stack and then set the base pointer to the return address from our procedure at the top of the new stack frame. F0h (240) bytes of stack space are reserved for the local variables within the main procedure. All of this uninitialized space is filled with 0CCCCCCCC's in order to blank them out for debugging purposes. One slot is for initial padding. Var1 takes up the second. Three and Four are padding. Var2 makes five. Six and seven are padding. Eight is var3. Nine and ten are padding. jmpret is the eleventh. The next 196 bytes are padding between stack frames. This is done by the compiler in all user-defined functions since the setjmp test file was compiled in debug mode. (Note that the stack frame is set up such that passed parameters are on the stack followed by the return address - where the new ebp points - followed by the current function parameters. esp always points to the top of the stack. That is, to the last used stack location.)

```
    14:
    15: void main( void )
    16: {
00401000  push        ebp
00401001  mov         ebp,esp
00401003  sub         esp,0F0h
00401009  push        ebx
0040100A  push        esi
0040100B  push        edi
0040100C  lea         edi,[ebp+FFFFFF10h]
00401012  mov         ecx,3Ch
00401017  mov         eax,0CCCCCCCCh
0040101C  rep stos    dword ptr [edi]
    17:   int v1, v2, v3;
    18:   int jmpret;
    19:
```

Here we are initializing the v1 .. v3 which are local variables stored on the stack.

```
    20:   v1 = 2;
0040101E  mov         dword ptr [ebp-8],2
    21:   v2 = 3;
00401025  mov         dword ptr [ebp-14h],3
    22:   v3 = 4;
0040102C  mov         dword ptr [ebp-20h],4
    23:
    24:   /* Save stack environment for return in case of error. First
    25:    * time through, jmpret is 0, so true conditional is executed.
    26:    * After the longjmp occurs, jmpret will be set to -1 and false
    27:    * conditional will be executed.
    28:    */
```


Call the setjmp procedure. Setjmp is a macro that returns 0 the first time through and we are filling in the return value ahead of time here. Mark is the structure that we pass in to setjmp3. _setjmp3 is the name of the function that is actually called through setjmp. The result is placed in jmpret. (Results are always placed in eax by convention) The stack is advanced by eight bytes to restore it to its previous position.

```
    29:   jmpret = setjmp( mark );
00401033  push        0
00401035  push        41D180h
0040103A  call        00401200
0040103F  add         esp,8
00401042  mov         dword ptr [ebp-2Ch],eax
```

Compare the return value to 0. If it is not equal then jump ahead to the else clause.

```
    30:   if( jmpret == 0 ) {
00401045  cmp         dword ptr [ebp-2Ch],0
00401049  jne         00401083
```

Call the printf procedure passing in all of the v parameters.

```
    31:      printf("v1 = %d, v2 = %d, v3 = %d\n", v1, v2, v3);
0040104B  mov         eax,dword ptr [ebp-20h]
0040104E  push        eax
0040104F  mov         ecx,dword ptr [ebp-14h]
00401052  push        ecx
00401053  mov         edx,dword ptr [ebp-8]
00401056  push        edx
00401057  push        41816Ch
0040105C  call        00401130
00401061  add         esp,10h
```

Set all v parameters to 222

```
    32:      v1 = v2 = v3 = 222;
00401064  mov         dword ptr [ebp-20h],0DEh
0040106B  mov         eax,dword ptr [ebp-20h]
0040106E  mov         dword ptr [ebp-14h],eax
00401071  mov         ecx,dword ptr [ebp-14h]
00401074  mov         dword ptr [ebp-8],ecx
```

Call the longjmp procedure with a parameter input of -1.

```
    33:      longjmp(mark, -1);
00401077  push        0FFFFFFFFh
00401079  push        41D180h
0040107E  call        004010AC
    34:   }
    35:   else {
```

Print the v parameters

```
    36:      printf("v1 = %d, v2 = %d, v3 = %d\n", v1, v2, v3);
00401083  mov         eax,dword ptr [ebp-20h]
00401086  push        eax
00401087  mov         ecx,dword ptr [ebp-14h]
0040108A  push        ecx
0040108B  mov         edx,dword ptr [ebp-8]
0040108E  push        edx
0040108F  push        41816Ch
00401094  call        00401130
00401099  add         esp,10h
    37:   }
```



```
    38:
    39:   return;
    40: }
```

Pop the saved variables off the stack and return (this will exit the program)

```
0040109C  jmp         004010A0
0040109E  jmp         004010A2
004010A0  xor         eax,eax
004010A2  pop         edi
004010A3  pop         esi
004010A4  pop         ebx
004010A5  mov         esp,ebp
004010A7  pop         ebp
004010A8  ret
```

Note: I have swapped the order of setjmp and longjmp here because I think it makes sense to comment the former before the latter.

```
--- No source file -----------------------------------------------------------
004011FE  int         3
004011FF  int         3
__setjmp3:

** Save selected register values into the jmp_buf data structure **

** Put the offset of the jmp_buf structure (mark) into edx **
00401200  mov         edx,dword ptr [esp+4]

** Save ebp in the first word of the jmp_buf structure **
00401204  mov         dword ptr [edx],ebp

** Put ebx into the second word of the jmp_buf structure **
00401206  mov         dword ptr [edx+4],ebx

** Save edi in the third word of the jmp_buf structure **
00401209  mov         dword ptr [edx+8],edi

** Save esi in the fourth word of the jmp_buf structure **
0040120C  mov         dword ptr [edx+0Ch],esi

** Save esp into the 5th word of the jmp_buf structure **
0040120F  mov         dword ptr [edx+10h],esp

** Save the return instruction address (eip value) into **
** the eax register **
00401212  mov         eax,dword ptr [esp]

** Put that value into the sixth word of the jmp_buf structure **
00401215  mov         dword ptr [edx+14h],eax

** This is a fixed value that is used as a type of sanity check **
** It is stored in the nineth word of the jmp_buf structure     **
00401218  mov         dword ptr [edx+20h],56433230h

** Tentatively set the exception unwinding code to be a null pointer **
** (Stored in the tenth word of the jmp_buf data structure           **
0040121F  mov         dword ptr [edx+24h],0

** Put the Registration into eax = first byte of the fs segment      **
** Note that the fs segment always points to the currently executing **
** thread's environment structure in Windows NT                      **
00401226  mov         eax,dword ptr fs:[00000000h]

** Put the registration value into the seventh position of the jmp_buf **
```



```
** structure                                                           **
0040122C  mov         dword ptr [edx+18h],eax

** Is the registration value equal to -1?                       **
0040122F  cmp         eax,0FFFFFFFFh

** If no then jmp to _s3_get_count **
00401232  jne         0040123D

** Record the Trylevel based on the thread information       **
** This is not something that we need to worry about.  It is **
** here to support extensions in C++                          **

** Record -1 as the try level in the eighth word of jmp_buf **
00401234  mov         dword ptr [edx+1Ch],0FFFFFFFFh

** jmp ahead to 00401278 **
0040123B  jmp         00401278

_s3_get_count:

** Put the second setjmp parameter into ecx.  (Apparently **
** that value is a 0 by default)  See if the value is,    **
** in fact, a 0.  If so then jump ahead to 0040120b)      **

** Put the second setjmp3 parameter into ecx **
0040123D  mov         ecx,dword ptr [esp+8]

** See if ecx is equal to 0 **
00401241  or          ecx,ecx

** If so jump to 0040124F **
00401243  je          0040124F

** If it is not, then grab the next higher stack position **
** and place it into eax                                  **
00401245  mov         eax,dword ptr [esp+0Ch]

** eax is the address of the unwindfunc that will be needed  **
** to handle exceptions later.  Save it in the tenth jmp_buf **
** position                                                  **
00401249  mov         dword ptr [edx+24h],eax

** Decrement eax by 1                                        **
0040124C  dec         ecx

** If it is not equal to 0 then jmp to _s3_save_trylevel **
0040124D  jne         00401257

_s3_default_trylevel:

** Put word at the 12'th byte above the address in eax into eax **
0040124F  mov         eax,dword ptr [eax+0Ch]

** Put the value into the eighth word of jmp_buf (the try level position) **
00401252  mov         dword ptr [edx+1Ch],eax

** Jump to _s3_done **
00401255  jmp         00401278

_s3_save_trylevel:

** Put the word at the 10'th byte above esp into eax **
00401257  mov         eax,dword ptr [esp+10h]
```

```
** Put that value into the try level (eighth word of the jmp_buf structure) **
0040125B  mov         dword ptr [edx+1Ch],eax

** Decrement ecx **
0040125E  dec         ecx

** If it equals zero then jump to _s3_done **
0040125F  je          00401278

** push esi **
00401261  push        esi

** push edi **
00401262  push        edi

** Load esi with the address of the 28'th byte above the stack **
00401263  lea         esi,[esp+1Ch]

** Load edi with the address of the 40'th byte above the stack **
00401267  lea         edi,[edx+28h]

** Compare ecx to 6 **
0040126A  cmp         ecx,6

** If it is less than or equal then jump to _s3_save_data **
0040126D  jbe         00401274

** Otherwise, load 6 into ecx **
0040126F  mov         ecx,6

_s3_save_data:

** Copy the 6 words from the edi address into esi
00401274  rep movs    dword ptr [edi],dword ptr [esi]

** Restore edi to its old value **
00401276  pop         edi

** Restore esi to its old value **
00401277  pop         esi

_s3_done:

** Zero out eax (the return value) **
00401278  sub         eax,eax

** Return **
0040127A  ret
0040127B  int         3
0040127C  int         3
0040127D  int         3
0040127E  int         3
0040127F  int         3

--- No source file -------------------------------------------------------------
004010A9  int         3
004010AA  int         3
004010AB  int         3
_longjmp:

** Put the jmp_buf address into ebx **
004010AC  mov         ebx,dword ptr [esp+4]

** Copy the first word of the jmp_buf structure into ebp **
004010B0  mov         ebp,dword ptr [ebx]
```



```
** Copy the seventh (Registration) word into ebx
004010B2  mov         esi,dword ptr [ebx+18h]

** Is esi equal to the exception handling word of the thread **
** environment block?                                        **
004010B5  cmp         esi,dword ptr fs:[00000000h]

** Yes, so jump ahead to _lj_local_unwind **
004010BC  je          004010C7

** push esi onto the stack **
004010BE  push        esi

**Call the _global_unwind2 function **
004010BF  call        004015E4

** Restore the stack after our function call **
004010C4  add         esp,4

_lj_local_unwind:

** Compare esi with 0 **
004010C7  cmp         esi,0

** If it is equal to 0 then jump to _lj_no_unwind **
004010CA  je          004010FC

** Load the address of the nineth word (Cookie) of jmp_buf into eax **
004010CC  lea         eax,[ebx+20h]

** Push eax's value **
004010CF  push        eax

** Call rt_probe_read4 **
004010D0  call        004016E0

** or eax (the return value) with itself **
004010D5  or          eax,eax

** Jump to _lj_old_unwind if it was 0 **
004010D7  je          004010EF

** Put the cookie value back into eax **
004010D9  mov         eax,dword ptr [ebx+20h]

** Was it equal to our original constant? **
004010DC  cmp         eax,56433230h

** If not, go to _lj_old_unwind **
004010E1  jne         004010EF

** Put the tenth word of jmp_buf (unwindfunc) into eax **
004010E3  mov         eax,dword ptr [ebx+24h]

** or eax with itself **
004010E6  or          eax,eax

** If it equaled zero then jump to _lj_no_unwind **
004010E8  je          004010FC

** Push ebx onto the stack **
004010EA  push        ebx

** Call the unwindfunc **
```



```
004010EB  call        eax

** Jump to _lj_no_unwind **
004010ED  jmp         004010FC

_lj_old_unwind:

** Push the eighth word in jmp_buf into eax **
004010EF  mov         eax,dword ptr [ebx+1Ch]

** Push eax **
004010F2  push        eax

** Push esi **
004010F3  push        esi

** Call the _local_unwind2 function **
004010F4  call        00401626

** Restore esp from our function call **
004010F9  add         esp,8


_lj_no_unwind:

** Push 0 onto the stack **
004010FC  push        0

** Put the sixth word (eip) into eax **
004010FE  mov         eax,dword ptr [ebx+14h]

** Call NLG_Notify **
00401101  call        004016BA

** Push ebx into edx **
00401106  mov         edx,ebx

** Put the second word of jmp_buf into ebx **
00401108  mov         ebx,dword ptr [edx+4]

** Put the third word of jmp_buf into edi **
0040110B  mov         edi,dword ptr [edx+8]

** Put the fourth word of jmp_buf into esi **
0040110E  mov         esi,dword ptr [edx+0Ch]

** Set eax equal to the return value.  Compare eax       **
** with 1.  If eax is not equal to 1 then the resulting  **
** subtraction (eax - 1) will have the same sign or will **
** be set to 0.  If eax is equal to 0 then (0 - 1) will  **
** trigger the carry bit and the adc instruction will    **
** then increment the eax register since adc will be set **
** (eax = eax + 0 + cf)                                  **

** Put the return value (set as the second function parameter) into eax **
00401111  mov         eax,dword ptr [esp+8]

** Compare with 1 **
00401115  cmp         eax,1

** If eax were equal to 0 then bump it up to 1 (as described above) **
00401118  adc         eax,0

** Put the fifth word of jmp_buf into esp **
0040111B  mov         esp,dword ptr [edx+10h]
```



```
** Throw away the top word on the stack **
0040111E  add         esp,4

** Jump to the address stored in the sixth word of the jmp_buf structure **
00401121  jmp         dword ptr [edx+14h]

** Return from the longjmp function (this is never reached) **
00401124  ret
00401125  int         3
00401126  int         3
00401127  int         3
00401128  int         3
00401129  int         3
0040112A  int         3
0040112B  int         3
0040112C  int         3
0040112D  int         3
0040112E  int         3
0040112F  int         3
```