```
# Michael E Sparks, 10-13-16 -- Towers of Hanoi in IA32 assembly, for Linux

# Demonstrates stack pressure resulting from a recursive algorithm that isn't
# last-call optimized. This example is meant to harmonize with my Prolog
# solution for the ToH puzzle.

# The following assembly code could be extended to handle any arbitrary
# number of disks .ge. 1.

# I've used pure assembly here--no "cheating" by calling out to the
# C standard library!

# See /usr/include/x86_64-linux-gnu/asm/unistd_32.h for a list of everyday
# Linux system calls/ kernel services.

.section .rodata

digits: .string "0123456789ABCDEF"

announce:
    .ascii " disks:\n"
    .set announce_len, .-announce

nl: .byte 0xA # ASCII code for newline

arrow:
    .ascii " -> "
    .set arrow_len, .-arrow

left: # encode with 0xA
    .ascii "left"
    .set left_len, .-left

right: # encode with 0xB
    .ascii "right"
    .set right_len, .-right

center: # encode with 0xC
    .ascii "ctr"
    .set center_len, .-center


.section .text
.globl _start
_start:
    xor %esi, %esi # tracks how many disks to use in puzzle

series:
    incl %esi

    # Here, we consider using up to 4 disks.
    # As is, the code can readily accommodate up to 15 disks.
    # Modification would be necessary to accommodate
    # yet larger numbers (and then only for the announcement-
    # related instructions--actual calculations should
    # proceed just fine, run-time requirements aside.)
    cmpl $0x4, %esi
    ja cleanup

    # announce how many disks for current iteration
    movl $0x4, %eax
    movl $0x1, %ebx
    movl $digits, %edi
    leal (%edi,%esi), %ecx
    movl $0x1, %edx
    int $0x80

    movl $0x4, %eax
    movl $0x1, %ebx
```



```
    movl $announce, %ecx
    movl $announce_len, %edx
    int $0x80

    pushl %esi # preserve counter register's state prior to
               # invoking hanoi routine (which also uses it)

    # Setup initial invocation, a la hanoi(N,L,R,C) from Prolog.
    # Note that, ordinarily, args get pushed in reverse order.
    pushl $0xC # center (auxiliary peg)
    pushl $0xB # right (destination peg)
    pushl $0xA # left (source peg)
    pushl %esi # N (disk count)
    call hanoi

    # Keep in mind that Linux stacks grow down and
    # that offsets are calculated in units of bytes.

    # Note also that assembly is handled differently on BSD-
    # flavored *nix systems, including Mac OS X; hence, this
    # code Just. Ain't. Portable.

    addl $0x10, %esp # clean off the stack

    popl %esi # retrieve counter state

    # print blank line
    movl $0x4, %eax
    movl $0x1, %ebx
    movl $nl, %ecx
    movl $0x1, %edx
    int $0x80

    jmp series


.type hanoi, @function
hanoi:
    # preserve caller's stack frame
    pushl %ebp
    movl %esp, %ebp

    movl 0x8(%ebp), %esi # retrieve invocation's 'N' parameter
    cmpl $0x0, %esi
    jz leave_hanoi # short circuit if invoked by leaf node

    decl %esi # sets N1

    # setup the hanoi(N1,L,C,R) call (i.e., expand internal node)
    pushl 0x10(%ebp)
    pushl 0x14(%ebp)
    pushl 0xC(%ebp)
    pushl %esi
    call hanoi

    # we pushed four 32-bit values on the stack - clean them off!
    addl $0x10, %esp

    # process leaf node (i.e., print report)
    movl $0x4, %eax
    movl $0x1, %ebx

    movl 0xC(%ebp), %edi # determine what 'L' variable's grounded to
    cmpl $0xB, %edi
    jb LA # implies 'L' is bound to "left" = 0xA
    je LB # to "right"
    ja LC # to "center"
LA:
    movl $left, %ecx
```

```
    movl $left_len, %edx
    jmp Ldone
LB:
    movl $right, %ecx
    movl $right_len, %edx
    jmp Ldone
LC:
    movl $center, %ecx
    movl $center_len, %edx
    jmp Ldone
Ldone:
    int $0x80

    movl $0x4, %eax
    movl $0x1, %ebx
    movl $arrow, %ecx
    movl $arrow_len, %edx
    int $0x80

    movl 0x10(%ebp), %edi # determine what 'R' variable's grounded to
    cmpl $0xB, %edi
    jb RA # implies 'R' is bound to "left" = 0xA
    je RB # to "right"
    ja RC # to "center"
RA:
    movl $left, %ecx
    movl $left_len, %edx
    jmp Rdone
RB:
    movl $right, %ecx
    movl $right_len, %edx
    jmp Rdone
RC:
    movl $center, %ecx
    movl $center_len, %edx
    jmp Rdone
Rdone:
    int $0x80

    movl $0x4, %eax
    movl $0x1, %ebx
    movl $nl, %ecx
    movl $0x1, %edx
    int $0x80

    # We need to reset esi after the recursive invocation above
    movl 0x8(%ebp), %esi
    decl %esi # sets N1

    # setup the hanoi(N1,C,R,L) call (i.e., expand internal node)
    pushl 0xC(%ebp)
    pushl 0x10(%ebp)
    pushl 0x14(%ebp)
    pushl %esi
    call hanoi

    addl $0x10, %esp # clean off the stack frame
leave_hanoi:
    movl %ebp, %esp
    popl %ebp
    ret

cleanup:
    movl $0x1, %eax
    xor %ebx, %ebx
    int $0x80
```