OS Homework 1 Re-Do

Taylor B. Morris
December 1, 2016

1. (a) The activation record is the data structure which stores information used by a function: the return address, parameters, and local variables and is found on the stack.

   (b) malloc allocates on the heap, so the value of theSum would be an address on the heap.

   (c) As i is a local variable, it would be stored on the stack.

   (d) As low is a global variable, it would be stored in the heap.

   (e) The instruction to call limit (line 11) would be in the code.

   (f) Defined symbols are symbols declared within a program. Undefined symbols are external symbols with no definition within the program.

   (g)

symbol	defined/undefined
low	defined
high	defined
main	defined
malloc	undefined
limit	undefined
printf	undefined

2. (c) file runs filesystem tests, magic tests, and language tests. Assuming object file named a.out for magic file list

      i. if $HOME/.magic.mgc or $HOME/.magic exists, will be used in preference to /usr/share/misc/magic.mgc and /usr/share/misc/magic/*
      ii. /usr/share/misc/magic.mgc
      iii. /usr/share/misc/magic/*
      iv. a.out
      v. elf.h
      vi. a.out.h
      vii. exec.h

   (d) tictac is identified as executable, and it was statically linked.

   (e) strace runs the command after it, aka file with tictac as input and prints out the functions called. Then, grep open goes through and only shows the lines from strace's output where open was called.

      i. "/usr/lib64/tls/x86_64/libmagic.so.1" - not successful
      ii. "/usr/lib64/tls/libmagic.so.1" - not successful
      iii. "/usr/lib64/x86_64/libmagic.so.1" - not successful
      iv. "/usr/lib64/libmagic.so.1"

v. "/usr/lib64/tls/libz.so.1" - not successful
vi. "/usr/lib64/libz.so.1"
vii. "/usr/lib64/tls/libc.so.6" - not successful
viii. "/usr/lib64/libc.so.6"
ix. "/usr/lib/locale/locale-archive"
x. "/etc/magic.mgc" - not successful
xi. "/etc/magic"
xii. "/usr/share/misc/magic.mgc"
xiii. "/usr/lib64/gconv/gconv-modules.cache"
xiv. "tictac"

(f) blind.o is an ELF relocatable, or object file, not yet compiled to be executable.

(g) i. atoi - undefined symbol
ii. g - uninitialized data, treated as undefined, but appears in heap.
iii. limit - defined, appears in code section

(j) We do not have the source code for limit; it was defined in blind.o.

(l) argv[1] is a string, or char[].

(m) It took 6 steps after the breakpoint. The values in the argument list are: the integer low at -10, integer high at 10, and str with a value of "10".

(n) x reformats an input, in this case from hex to string. The value printed out was "10"

(o) The address of the statement immediately following limit is 0x0000000000400e3c. The statement will be executed 3 times.

3. The user program calls the system call as if it were a user procedure. The user procedure then calls a kernel stub. The kernel stub stores the arguments to their registers and the system call's code to a special register - this is later refered to by the kernel to decide which call to use - and traps to the kernel. The kernel then refers to the arguments and system call code stored earlier to execute the call before returning control to the handler to hand back to the user program.

4. Execution in user mode is very restricted versus execution in system mode. In user mode, the addresses which can be accessed are limited by special base and bound registers or by virtual addresses, which keep the user process from affecting memory in locations it shouldn't reach. Additionally, the user mode cannot access certain privileged instructions and must turn over control to the system to call these instructions. These privileged instructions add a layer of security, keeping the user mode process from performing potentially harmful operations or accessing private data.

   If the user mode process needs a privileged instruction to be executed, it hands over the control to the system mode. In the x86 system, the mode switch takes the following steps: first, the operating sytem stops interrupts from happening during the mode switch (deferring them instead until later); next, the system saves key values from

registers to temporary hardware registers; then, it switches the stack pointer to the kernel stack using a special hardware register's value; once on the kernel stack, the key values are pushed onto the stack; at this point, the system saves the error code (if there is one) to the top of the stack, and a dummy variable to the top of the stack if there isn't an error code; finally, the hardware changes the program counter to the interrupt handler. Because these operations are hard operations in the hardware, they cannot be tampered with by user processes.

5. Kill will be raised from PID 1000. Kill is sent to PID 2000. The handler receives what system call it is, along with the arguments sent into the call. For this instance, the system call is a kill routine. Assuming process 2000 is a currently running process: Once in the handler, the operating system must transfer the process state that had been saved from PID 2000 when PID 2000 became in active to its signal stack and exits kernel mode and places the program pointer at the beginning of the process signal handler.

6. (a) Switch stack pointer to process interrupt stack
   (b) Push program counter and other key values onto stack
   (c) Push register values onto stack
   (d) Switch to new context
   (e) Pop register values from new context's interrupt stack
   (f) Pop program counter and other key values off stack
   (g) Switch program counter to being the new context

7. The output from this program is deterministic with the output

   intA 2 intB 4
   intA 2 intB 4