Files

Systems Programming

JOHANNES KEPLER
UNIVERSITÄT LINZ

JKU

Michael Roland
Michael Sonntag
Institute of Networks and Security

INSTITUTE
OF NETWORKS
AND SECURITY

From the lecture part…

System calls for file manipulation

System call	RAX (cmd.)	RDI (parameter 1)	RSI (parameter 2)	RDX (parameter 3)	RAX (return value)
SYS_OPEN	2	Pointer to filename	Flags (O_RDONLY, …)	Create mode (e.g. 0666)	File descriptor or error number
SYS_READ	0	File descriptor	Pointer to data buffer	Max. number of bytes to read	Actual number of bytes read or error number
SYS_WRITE	1	File descriptor	Pointer to data buffer	Number of bytes to write	Actual number of bytes written or error number
SYS_CLOSE	3	File descriptor	---	---	0 (success) or error number

Example: toupper.s – Helper constants

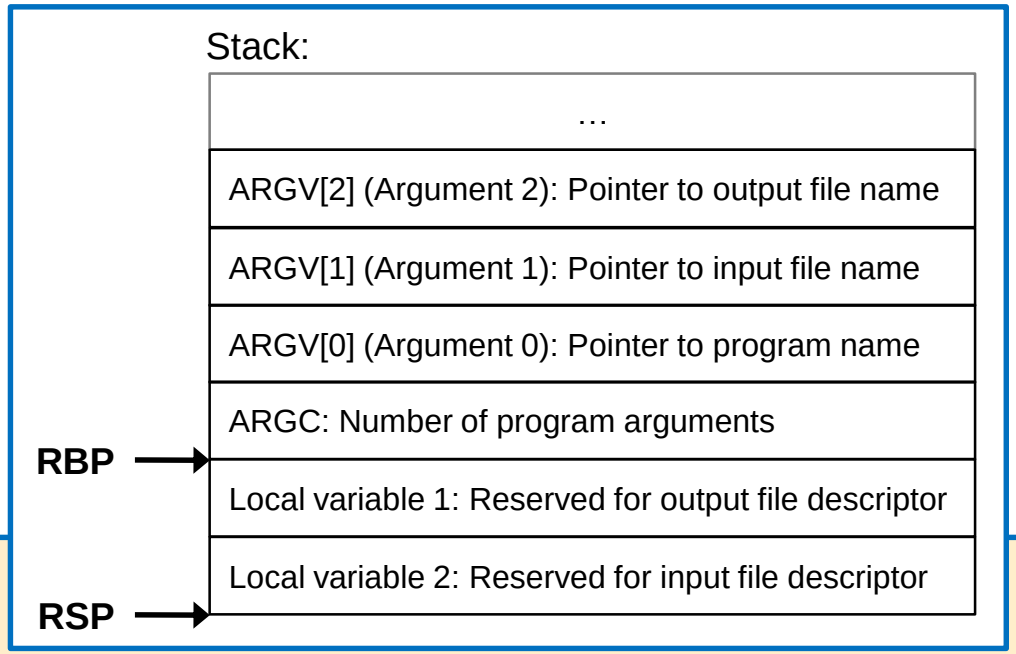
```
# System call numbers
.equ SYS_OPEN, 2
.equ SYS_READ, 0
.equ SYS_WRITE, 1
.equ SYS_CLOSE, 3
.equ SYS_EXIT, 60

.equ O_RDONLY, 0                      # Open file options - read-only
.equ O_CREAT_WRONLY_TRUNC, 03101      # Open file options - these are:
                                      # CREAT  - create file if not exising
                                      # WRONLY - only write to this file
                                      # TRUNC  - destroy current contents

.equ O_PERMS, 0666                    # Read & Write perms. for everyone

# End-of-file result status
.equ END_OF_FILE, 0     # This is the return value of read()
                        # which means we've hit the end of
                        # the file
```

Example: toupper.s – Data buffer

```
.section .bss
# This is where the data is loaded
# into from the data file and written
# from into the output file. It should
# never exceed 16,000 for various
# reasons.
.equ BUFFER_SIZE, 500
.lcomm BUFFER_DATA, BUFFER_SIZE
```

Example: toupper.s – Helper constants

Remember: OS puts command line arguments (actually only pointers to those strings) on stack




```
    .section .text

    # STACK POSITIONS
    .equ ST_SIZE_RESERVE, 16 # Space for local variables
    # Note: Offsets are RBP-based, which is set immediately at program start
    .equ ST_FD_IN, -16       # Local variable for input file descriptor
    .equ ST_FD_OUT, -8       # Local variable for output file descriptor
    .equ ST_ARGC,    0       # Number of arguments (integer)
    .equ ST_ARGV_0,  8       # Name of program (address = pointer to string)
    .equ ST_ARGV_1, 16       # Input file name (address = pointer to string)
    .equ ST_ARGV_2, 24       # Output file name (address = pointer to string)
```

Example: toupper.s (4)

```
_start:
    ###INITIALIZE PROGRAM###
    movq %rsp, %rbp              # Create new stack frame
    subq $ST_SIZE_RESERVE, %rsp # Allocate file descriptor space on stack

    ###CHECK PARAMETER COUNT###
    cmpq $3, ST_ARGC(%rbp)
    je open_files

    movq $-1, %rdi               # Our return value for parameter problems
    movq $SYS_EXIT, %rax
    syscall
```

Example: toupper.s (5)

```
open_files:
open_fd_in:
    ###OPEN INPUT FILE###
    movq ST_ARGV_1(%rbp), %rdi   # Input filename into %rdi
    movq $O_RDONLY, %rsi         # Read-only flag
    movq $O_PERMS, %rdx          # This doesn't really matter for reading
    movq $SYS_OPEN, %rax         # Specify "open"
    syscall                      # Call Linux

    cmpq $0, %rax                # Check success
    jl exit                      # In case of error simply terminate
store_fd_in:
    movq  %rax, ST_FD_IN(%rbp)   # Save the returned file descriptor
```

Example: toupper.s (6)

```
open_fd_out:
    ###OPEN OUTPUT FILE###
    movq ST_ARGV_2(%rbp), %rdi          # Output filename into %rdi
    movq $O_CREAT_WRONLY_TRUNC, %rsi    # Flags for writing to the file
    movq $O_PERMS, %rdx                 # Permissions for new file (if created)
    movq $SYS_OPEN, %rax                # Open the file
    syscall                             # Call Linux

    cmpq $0, %rax                       # Check success
    jl close_input                      # In case of error close input file
                                        # (already open!)
store_fd_out:
    movq %rax, ST_FD_OUT(%rbp)          # Store the file descriptor
```

Example: toupper.s (7)

```
read_loop_begin:
    ###READ IN A BLOCK FROM THE INPUT FILE###
    movq ST_FD_IN(%rbp), %rdi      # Get the input file descriptor
    movq $BUFFER_DATA, %rsi        # The location to read into
    movq $BUFFER_SIZE, %rdx        # The size of the buffer
    movq $SYS_READ, %rax
    syscall                        # Size of buffer read is returned in %rax
    ###EXIT IF WE'VE REACHED THE END###
    cmpq $END_OF_FILE, %rax        # Check for end of file marker (or error)
    je end_loop                    # If found, go to the end
    jl close_output                # On error just terminate

continue_read_loop:
    ###CONVERT THE BLOCK TO UPPER CASE###
    movq $BUFFER_DATA, %rdi        # Location of the buffer
    movq %rax, %rsi                # Size of the buffer
    pushq $-1                      # Dummy value for stack alignment
    pushq %rax                     # Store bytes read for write check
    call convert_to_upper
```

Example: toupper.s (8)

```
write_out_begin:
    ###WRITE THE BLOCK OUT TO THE OUTPUT FILE###
    movq ST_FD_OUT(%rbp), %rdi      # File to use
    movq $BUFFER_DATA, %rsi         # Location of  buffer
    movq %rax, %rdx                 # Buffer size (=number of bytes read)
    movq $SYS_WRITE, %rax
    syscall                         # Note: Check how much was written!

    ###CHECK WRITE SUCCESS###
    popq %rbx                       # Retrieve number of bytes read
    addq $8, %rsp                   # Remove stack alignment space
    cmpq %rax, %rbx                 # Compare number read to written
    jne close_output                # If not the same, terminate program

    ###CONTINUE THE LOOP###
    jmp read_loop_begin
```

Example: toupper.s (9)

```
end_loop:                   # No special error handling, so success and error
close_output:               # are the same: we just close both files
    ###CLOSE THE FILES###
    # NOTE - we don't need to do error checking on these, because
    #        error conditions don't signify anything special here
    movq ST_FD_OUT(%rbp), %rdi
    movq $SYS_CLOSE, %rax
    syscall

close_input:
    movq ST_FD_IN(%rbp), %rdi
    movq $SYS_CLOSE, %rax
    syscall

exit:
    ###EXIT###
    movq $0, %rdi
    movq $SYS_EXIT, %rax
    syscall
```

Example: toupper.s (10)

```
#####FUNCTION convert_to_upper
#PURPOSE:    This function actually does the conversion to upper case for a block
#INPUT:      The first parameter (rdi) is the location of the block of memory to convert
#            The second parameter (rsi) is the length of that buffer
#OUTPUT:     This function overwrites the current buffer with the upper-casified version.
#VARIABLES:
#            %rax  - beginning of buffer
#            %rbx  - length of buffer (old value must be saved!)
#            %rdi  - current buffer offset
#            %r10b - current byte being examined (%r10b is the first byte of %r10)
# Note: This variable assignment is for exemplary purposes only and very suboptimal!

    .equ  LOWERCASE_A, 'a'          # The lower boundary of our search
    .equ  LOWERCASE_Z, 'z'          # The upper boundary of our search
    .equ  UPPER_CONVERSION, 'A' - 'a'
                                    # Conversion: Difference upper/lower case
convert_to_upper:
    pushq %rbp                      # Prepare stack
    movq  %rsp, %rbp
    pushq %rbx                      # Save RBX
    ###SET UP VARIABLES###
    movq  %rdi, %rax
    movq  %rsi, %rbx
    movq  $0, %rdi
```

Example: toupper.s (11)

```
    # If a buffer with zero length was given us, just leave
    cmpq $0, %rbx
    je end_convert_loop
convert_loop:
    movb (%rax,%rdi,1), %r10b     # Get the current byte
    # Go to the next byte unless it is between 'a' and 'z'
    cmpb $LOWERCASE_A, %r10b
    jl next_byte
    cmpb $LOWERCASE_Z, %r10b
    jg next_byte
    # Otherwise convert the byte to uppercase
    addb $UPPER_CONVERSION, %r10b
    movb %r10b, (%rax,%rdi,1)     # And store it back
next_byte:
    incq %rdi                     # Next byte
    cmpq %rdi, %rbx               # Continue unless we've reached the end
    jne convert_loop
end_convert_loop:
    movq %rdi, %rax         # Store number of chars converted as return value
    popq %rbx
    movq %rbp, %rsp
    popq %rbp
    ret
```

JOHANNES KEPLER
UNIVERSITÄT LINZ

INSTITUTE
OF NETWORKS
AND SECURITY

https://www.ins.jku.at

THANK YOU FOR YOUR ATTENTION!

Slides by: Michael Roland and Michael Sonntag
michael.roland@ins.jku.at, michael.sonntag@ins.jku.at
+43 (732) 2468 - 4137
S3 235 (Science park 3, 2nd floor)

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Österreich
www.jku.at