64-bit Assembly in Linux

Or 'Building a better penguin'.

Through a Unix, Darkly

This text will amke use of Linux as a platform for developing software using assembly language. This includes Linux utilities and the Linux command-line or shell. Some information could be used in other Unix based OS's like BSD or MacOS. However you will need to be mindful of available cpu instructions and the calling convention of available C libraries (not to mention the other API's of any other libraries called).

For this reason lets start with the C Calling convention and then move on to Linux System Calls.

Call to Order

There are two types of function calls in Linux, the C calling convention and the System Call (which is also C style). The reason for the difference is that the system and a C Library function use different registers for holding parameters or arguments to a function. But this is a minor difference and the two calling methods are just about identical.

Before calling a function, you must load the registers with the parameters a function will operate on.

```
# Loading sacred registers for functions

movb $value,%ah      # Move 8-bit  value into register ah
movw $value,%ax      # Move 16-bit value into register ax
movl $value,%eax     # Move 32-bit value into register eax
movq $value,%rax     # Move 64-bit value into register rax

# Call the function after loading parameters
call function
```

This example does not show the order of the arguments, while unchanged between System Calls and C Calls, I will show the order in the next sections. Here is shown that you need to specify the size of the parameters when storing them in registers.

Note that in most cases you cannot push or pop an 8-bit value (byte) or register from the stack, but you can move bytes into and out of registers and variables. The above example is for completeness.

Calling Mr. Function, Mr. Function

When writing C library functions (or any functions) you have to preserve the value in the registers used by the function. For the C library these registers are : rbx, rsp, rbp, r12 through r15 ; when calling a C function these registers must be pushed and rsp adjusted for the new position. The parameters or arguments passed to a function are 'mov'ed to the appropriate register. All examples assume a 64-bit value in a r- prefix register.

```
extern fName        # Need to tell assembler that the function
                    # is found elsewhere, externally

# Moving data for functions arguments
# Use as many as needed in the specified registers
# Arguments after sixth, are passed on the stack.

movq $1stArg,%rdi   # First  Argument
movq $2ndArg,%rsi   # Second Argument
movq $3rdArg,%rdx   # Third  Argument
movq $4thArg,%rcx   # Fourth Argument; for syscall use %r10
movq $5thArg,%r8    # Fifth  Argument
movq $6thArg,%r9    # Sixth  Argument
pushq $10thArg      # Tenth   Arg; must be in reverse order
pushq $9thArg       # Ninth   Arg
pushq $8thArg       # Eigth   Arg
pushq $7thArg       # Seventh Arg

# Preserve sacred regs for C functions
# What is otherwise called a function prolog
fName:              # Function label, not needed for C library
                    # calls.
pushq %rbp          # Preserve old base pointer
movq %rsp,%rbp      # Copy new address into pointer
pushq %rbx          #
pushq %r12          # Preserve the registers used otherwise
pushq %r13          # preserve additional registers you need
pushq %r14          #
pushq %r15          #

# Do function stuff here
```

Values are 'push'ed in reverse order with the last value first. Up to ten arguments can be made with the first six stored in registers, the remaining four must be passed in values that are push'ed to the stack.

```
# Done with C function sacred regs
# What is otherwise called a function epilog

# Function stuff done here

popq %r15           #
popq %r14           # Pop in reverse order of push
popq %r13           #
popq %r12           #
popq %rbx           #
movq  %rbp,%rsp     # Restore pointers
popq  %rbp          # Restore base pointer

# Return to program that called the function
ret

# Remove arguments pushed before function called
popq $7thArg        # Pop in reverse order of push
popq $8thArg        #
popq $9thArg        #
popq $10thArg       #

end                 # End of program
```

If you are going to use any other register for the function it must be added to this list. You push them after %rdi and pop them before %rdi; in the correct order. You can only pop/push 16-bit, 32-bit and 64-bit registers.

Floating values are stored in xmm0 (first arg or function result) to xmm7 (eigth arg) and not the general purpose registers.

In place of restoring the stack pointers yourself (%rsp and %rbp), you may use the instruction ' leave ' before using ' ret ' instead. Arguments must still be removed as normal.

Note that everything between the last argument pushed and the ret instruction is used by the call instruction.

```
# Store arguments for function

# Function is called

call function
# <--- Function returns here

# Clean up after function is done, here
```

C Library calls reads parameters from the following order of registers. Note that syscall uses rcx and therefore you replace rcx as an argument with r10 instead. RBX, RBP, RSP and R12 to R15 are used by the system, if they are needed they must be preserved (Saved). C Library calls can take up to ten parameters.

Assuming 64-bit CPU (match registers to argument size) :

	64-bit	32-bit	16-bit	8-bit	Saved	Example	Example
Call/Result	rax	eax	ax	ah	No	write ($1)	exit ($60)
First Arg	rdi	edi	di	dil	No	Write location	-
Second	rsi	esi	si	sil	No	String location	-
Third/Result	rdx	edx	dx	dl	No	Amount to write	-
Fourth	rcx	ecx	cx	cl	No	-	-
Base	rbx	ebx	bx	bl	Yes	-	-
Base	rbp	ebp	bp	-	Yes	-	-
Stack	rsp	esp	sp	-	Yes	-	-
Fifth	r8	r8d	r8w	r8b	No	-	-
Sixth	r9	r9d	r9w	r9b	No	-	-
Temp	r10	r10d	r10w	r10b	No	-	-
Temp	r11	r11d	r11w	r11b	No	-	-
	r12	r12d	r12w	r12b	Yes	-	-
	r13	r13d	r13w	r13b	Yes	-	-
	r14	r14d	r14w	r14b	Yes	-	-
	r15	r15d	r15w	r15b	Yes	-	-
1st/Return	-	xmm0	-	-	No	-	-
2nd/Return	-	xmm1	-	-	No	-	-
Float Arg	-	xmm2	-	-	No	-	-
Float Arg	-	xmm3	-	-	No	-	-
Float Arg	-	xmm4	-	-	No	-	-
Float Arg	-	xmm5	-	-	No	-	-
Float Arg	-	xmm6	-	-	No	-	-
Float Arg	-	xmm7	-	-	No	-	-
Temp	-	xmm8-15	-	-	No	-	-

Read Mr. Hyde's books, he explains the stack in greater detail than here and more accurately; though I don't recall any mention of the C library. His text is more dedicated to Assembly code and discusses C functions much later near the end of his book (AoA).

Calling Mr. Linux, Mr. Linux ...

Linux System calls (syscall) reads parameters from the following order of registers. Note that syscall uses rcx and therefore you replace rcx as an argument with r10 instead. RBX, RBP, RSP and R12 to R15 are used by the system, if they are needed they must be preserved (Saved). Syscalls can only take six parameters.

Assuming 64-bit CPU (match registers to argument size) :

	64-bit	32-bit	16-bit	8-bit	Saved	Example	Example
Call/Result	rax	eax	ax	ah	No	write ($1)	exit ($60)
First Arg	rdi	edi	di	dil	No	Write location	-
Second	rsi	esi	si	sil	No	String location	-
Third/Result	rdx	edx	dx	dl	No	Amount to write	-
Fourth	r10	r10d	r10w	r10b	No	-	-
Base	rbx	ebx	bx	bl	Yes	-	-
Base	rbp	ebp	bp	-	Yes	-	-
Stack	rsp	esp	sp	-	Yes	-	-
Fifth	r8	r8d	r8w	r8b	No	-	-
Sixth	r9	r9d	r9w	r9b	No	-	-
1st/Return	-	xmm0	-	-	No	-	-
2nd/Return	-	xmm1	-	-	No	-	-
Float Arg	-	xmm2	-	-	No	-	-
Float Arg	-	xmm3	-	-	No	-	-
Float Arg	-	xmm4	-	-	No	-	-
Float Arg	-	xmm5	-	-	No	-	-
Float Arg	-	xmm6	-	-	No	-	-
Float Arg	-	xmm7	-	-	No	-	-
Temp	-	xmm8-15	-	-	No	-	-

```
# No need for function label or extern symbol
# Unless you want to give your exit call a function
# of its own.

# Do not preserve registers before calling syscall
# Unless you use a register used by syscall

movq    $60, %rax    # Load %rax with system call number
movq    $0, %rdi     # Load %rdi with exit status
syscall              # Make the system call

# No ret or leave instruction is used,
# no clean up of registers.
# Unless you used a register used by the syscall instruction
```

There are about 300 Linux System Calls. These calls use the C-style syntax but are calls to functions built into the Linux kernel and not to a library file (' .so ' or shared object file).

As shown above there is no need to preserve or clean up registers or stack unless you need to preserve registers used by the syscall. If so, then treat as a normal call to a C library function.

For the System Call Table, the following abbreviations are used (? indicates an optional value) :
addr – address or location of data (as in memory or descriptor)
start – address or location to start operation
len – length or duration of operation
path – location of file (as in directory paths)
flags – metadata or information on status of bits
ptr – pointer to structure, structure (like an array) will have to be parsed
count – number of fields in a structure or number of structures
time – time to wait or time-out duration
method – how to apply flags or offsets
code – device specific binary
arg – device specific arguments

For more information on any call see " $ man 2 call " where call is one of the functions in the table.

Call	rax	rdi	rsi	rdx	r10	r8	r9
read	$0	addr	start	len			
write	$1	addr	start	len			
open	$2	path	flags				
close	$3	addr					
stat	$4	path	ptr				
fstat	$5	addr	ptr				
lstat	$6	path	ptr				
poll	$7	ptr	count	time			
lseek	$8	addr	len	method			
mmap	$9	addr	len	flags	flags	addr	len
mprotect	$10	addr	len	flags			
munmap	$11	addr	len				
brk	$12	addr					
sigaction	$13	flag	ptr	ptr			
sigprocmask	$14	flag	ptr	ptr			
sigreturn	$15	-	-	-	-	-	-
ioctl	$16	addr	code	arg			
pread	$17	addr	start	len	len		
pwrite	$18	addr	start	len	len		
readv	$19	addr	ptr	count			
writev	$20	addr	ptr	count			
access	$21	addr	path	method	flags		
pipe	$22	ptr					
select	$23	count	ptr	ptr	ptr		
sched_yield	$24	-	-	-	-	-	-
mremap	$25	addr	len	len	flags	?addr	
msync	$26						
mincore	$27						
madvise	$28						
shmget	$29						

shmat	$30						
shmctl	$31						
dup	$32						
dup2	$33						
pause	$34						
nanosleep	$35						
getitimer	$36						
alarm	$37						
setitimer	$38						
getpid	$39						
sendfile	$40						
socket	$41						
connect	$42						
accept	$43						
sendto	$44						
recvfrom	$45						
sendmsg	$46						
recvmsg	$47						
shutdown	$48						
bind	$49						
listen	$50						
getsockname	$51						
getpeername	$52						
socketpair	$53						
setsockopt	$54						
getsockopt	$55						
clone	$56						
fork	$57						
vfork	$58						
execve	$59						
exit	$60	status					
wait4	$61						
kill	$62						
uname	$63						

semget	$64						
semop	$65						
semctl	$66						
shmdt	$67						
msgget	$68						
msgsnd	$69						
msgrcv	$70						
msgctl	$71						
fcntl	$72						
flock	$73						
fsync	$74						
fdatasync	$75						
truncate	$76						
ftruncate	$77						
getdents	$78						
getcwd	$79						
chdir	$80						
fchdir	$81						
rename	$82						
mkdir	$83						
rmdir	$84						
creat	$85						
link	$86						
unlink	$87						
symlink	$88						
readlink	$89						
chmod	$90						
fchmod	$91						
chown	$92						
fchown	$93						
lchown	$94						
umask	$95						
gettimeofday	$96						
getrlimit	$97						

getrusage	$98						
sysinfo	$99						
times	$100						
ptrace	$101						
getuid	$102						
syslog	$103						
getgid	$104						
setuid	$105						
setgid	$106						
geteuid	$107						
getegid	$108						
setpgid	$109						
getppid	$110						
getpgrp	$111						
setsid	$112						
setreuid	$113						
setregid	$114						
getgroups	$115						
setgroups	$116						
setresuid	$117						
getresuid	$118						
setresgid	$119						
getresgid	$120						
getpgid	$121						
setfsuid	$122						
setfsgid	$123						
getsid	$124						
capget	$125						
capset	$126						
rt_sigpending	$127						
rt_sigtimedwait	$128						
rt_sigqueuinfo	$129						
rt_sigsuspend	$130						
sigaltstack	$131						

utime	$132						
mknod	$133						
uselib	$134						
personality	$135						
ustat	$136						
statfs	$137						
fstatfs	$138						
sysfs	$139						
getpriority	$140						
setpriority	$141						
sched_setparam	$142						
sched_getparam	$143						
sched_setscheduler	$144						
sched_getscheduler	$145						
sched_get_priority_max	$146						
sched_get_priority_min	$147						
sched_rr_get_interval	$148						
mlock	$149						
munlock	$150						
mlockall	$151						
munlockall	$152						
vhangup	$153						
modify_ldt	$154						
pivot_root	$155						
_sysctl	$156						
prctl	$157						
arch_pctl	$158						
adjtimex	$159						
setrlimit	$160						
chroot	$161						
sync	$162						
acct	$163						
settimeofday	$164						
mount	$165						

umount	$166						
swapon	$167						
swapoff	$168						
reboot	$169						
sethostname	$170						
setdomainname	$171						
iopl	$172						
ioperm	$173						
create_module	$174						
init_module	$175						
delete_module	$176						
get_kern_syms	$177						
query_module	$178						
quotactl	$179						
nfsservctl	$180						
getpmsg	$181						
putpmsg	$182						
afs_syscall	$183						
tuxcall	$184						
security	$185						
gettid	$186						
readahead	$187						
setxattr	$188						
lsetxattr	$189						
fsetxattr	$190						
getxattr	$191						
lgetxattr	$192						
fgetxattr	$193						
listxattr	$194						
llistxattr	$195						
flistxattr	$196						
removexattr	$197						
lremovexattr	$198						
fremovexattr	$199						

tkill	$200						
time	$201						
futex	$202						
sched_setaffinity	$203						
sched_getaffinity	$204						
set_thread_area	$205						
io_setup	$206						
io_destroy	$207						
io_getevents	$208						
io_submit	$209						
io_cancel	$210						
get_thread_area	$211						
lookup_dcookie	$212						
epoll_create	$213						
epoll_ctl_old	$214						
epoll_wait_old	$215						
remap_file_pages	$216						
getdents64	$217						
set_tid_address	$218						
restart_syscall	$219						
semtimedop	$220						
fadvise64	$221						
timer_create	$222						
timer_settime	$223						
timer_gettime	$224						
timer_getoverrun	$225						
timer_delete	$226						
clock_settime	$227						
clock_gettime	$228						
clock_getres	$229						
clock_nanosleep	$230						
exit_group	$231						
epoll_wait	$232						
epoll_ctl	$233						

tgkill	$234						
utimes	$235						
vserver	$236						
mbind	$237						
set_mempolicy	$238						
get_mempolicy	$239						
mq_open	$240						
mq_unlink	$241						
mq_timedsend	$242						
mq_timedreceive	$243						
mq_notify	$244						
mq_getsetattr	$245						
kexec_load	$246						
waitid	$247						
add_key	$248						
request_key	$249						
keyctl	$250						
ioproi_set	$251						
ioprio_get	$252						
inotify_init	$253						
inotify_add_watch	$254						
inotify_rm_watch	$255						
migrate_pages	$256						
openat	$257						
mkdirat	$258						
mknodat	$259						
fchownat	$260						
futimesat	$261						
newfstatat	$262						
unlinkat	$263						
renameat	$264						
linkat	$265						
symlinkat	$266						
readlinkat	$267						

fchmodat	$268						
faccessat	$269						
pselect6	$270						
ppoll	$271						
unshare	$272						
set_robust_list	$273						
get_robust_list	$274						
splice	$275						
tee	$276						
sync_file_range	$277						
vmsplice	$278						
move_pages	$279						
utimensat	$280						
epoll_pwait	$281						
signalfd	$282						
timerfd_create	$283						
eventfd	$284						
fallocate	$285						
timerfd_settime	$286						
timerfd_gettime	$287						
accept4	$288						
signalfd4	$289						
eventfd2	$290						
epoll_create1	$291						
dup3	$292						
pipe2	$293						
inotify_init1	$294						
preadv	$295						
pwritev	$296						
rt_tgsigqueueinfo	$297						
perf_event_open	$298						
recvmmsg	$299						
fanotify_init	$300						
fanotify_mark	$301						

prlimit64	$302						
name_to_handle_at	$303						
open_by_handle_at	$304						
clock_adjtime	$305						
syncfs	$306						
sendmmsg	$307						
setns	$308						
getcpu	$309						
process_vm_readv	$310						
process_vm_writev	$311						
kcmp	$312						
finit_module	$313						
sched_setattr	$314						
sched_getattr	$315						
renameat2	$316						
seccomp	$317						
getrandom	$318						
memfd_create	$319						
kexec_file_load	$320						
bpf	$321						
execveat	$322						
userfaultfd	$323						
membarrier	$324						
mlock2	$325						
copy_file_range	$326						
preadv2	$327						
pwritev2	$328						
pkey_mprotect	$329						
pkey_alloc	$330						
pkey_free	$331						