32 bits Linux system calls

Hello? who's this?

© Government of Canada
This document is the property of the Government of Canada. It shall not be altered, distributed beyond its intended audience, produced, reproduced or published, in whole or in any substantial part thereof, without the express permission of CSE.

Full list for 32 bits is:

/usr/include/i386-linux-gnu/asm/unistd_32.h

© Government of Canada
This document is the property of the Government of Canada. It shall not be altered, distributed beyond its intended audience, produced, reproduced or published, in whole or in any substantial part thereof, without the express permission of CSE.

This file looks somewhat like this

```
#ifndef _ASM_X86_UNISTD_32_H
#define _ASM_X86_UNISTD_32_H 1

#define __NR_restart_syscall 0
#define __NR_exit 1
#define __NR_fork 2
#define __NR_read 3
#define __NR_write 4
#define __NR_open 5
#define __NR_close 6
#define __NR_waitpid 7
#define __NR_creat 8
#define __NR_link 9
#define __NR_unlink 10
#define __NR_execve 11
#define __NR_chdir 12
#define __NR_time 13
#define __NR_mknod 14
#define __NR_chmod 15
```

Each system call has its own number!

You can refer to this file if you are not sure what number is assigned to a specific system call.

Extract from:
/usr/include/i386-linux-gnu/asm/unistd_32.h

Documentation for system calls

man 2 <system_call_name>

I also like: https://syscalls.kernelgrok.com/

© Government of Canada
This document is the property of the Government of Canada. It shall not be altered, distributed beyond its intended audience, produced, reproduced or published, in whole or in any substantial part thereof, without the express permission of CSE.

Doing a call - Hello system calls

```
global main

section .data
    hello:
        db 'Hello system calls', 0x0a, 0

section .text

main:
        push ebp
        mov ebp, esp

        mov eax, 0x04   ; Write System Call
        mov ebx, 0x01   ; 0=STDIN, 1=STDOUT, 2=STDERR
        mov ecx, hello  ; Message
        mov edx, 0x13   ; Len
        int 0x80        ; Send the system call

        pop ebp
        ret
```

System call **number** goes into **eax**.

Parameters go into **ebx, ecx, edx, esi,edi, ebp** from first to sixth parameters.

When dealing with STDIN, STDOUT or STDERR, their respective value is 0, 1, 2.

0x80 is the **interrupt number** that allows to send a **software interrupt** for a system call into the kernel.

Calling convention details can be found using “man 2 syscall”

Calling convention based on architecture

arch/ABI	instruction	syscall #	retval	Notes
arm/OABI	swi NR	-	a1	NR is syscall #
arm/EABI	swi 0x0	r7	r0	
arm64	svc #0	x8	x0	
blackfin	excpt 0x0	P0	R0	
i386	int $0x80	eax	eax	
ia64	break 0x100000	r15	r8	See below
mips	syscall	v0	v0	See below
parisc	ble 0x100(%sr2, %r0)	r20	r28	
s390	svc 0	r1	r2	See below
s390x	svc 0	r1	r2	See below
sparc/32	t 0x10	g1	o0	
sparc/64	t 0x6d	g1	o0	
x86_64	syscall	rax	rax	See below
x32	syscall	rax	rax	See below

arch/ABI	arg1	arg2	arg3	arg4	arg5	arg6	arg7	Notes
arm/OABI	a1	a2	a3	a4	v1	v2	v3	
arm/EABI	r0	r1	r2	r3	r4	r5	r6	
arm64	x0	x1	x2	x3	x4	x5	-	
blackfin	R0	R1	R2	R3	R4	R5	-	
i386	ebx	ecx	edx	esi	edi	ebp	-	
ia64	out0	out1	out2	out3	out4	out5	-	
mips/o32	a0	a1	a2	a3	-	-	-	See below
mips/n32,64	a0	a1	a2	a3	a4	a5	-	
parisc	r26	r25	r24	r23	r22	r21	-	
s390	r2	r3	r4	r5	r6	r7	-	
s390x	r2	r3	r4	r5	r6	r7	-	
sparc/32	o0	o1	o2	o3	o4	o5	-	
sparc/64	o0	o1	o2	o3	o4	o5	-	
x86_64	rdi	rsi	rdx	r10	r8	r9	-	
x32	rdi	rsi	rdx	r10	r8	r9	-	

Extract from:
man 2 syscall

Let’s write some code!