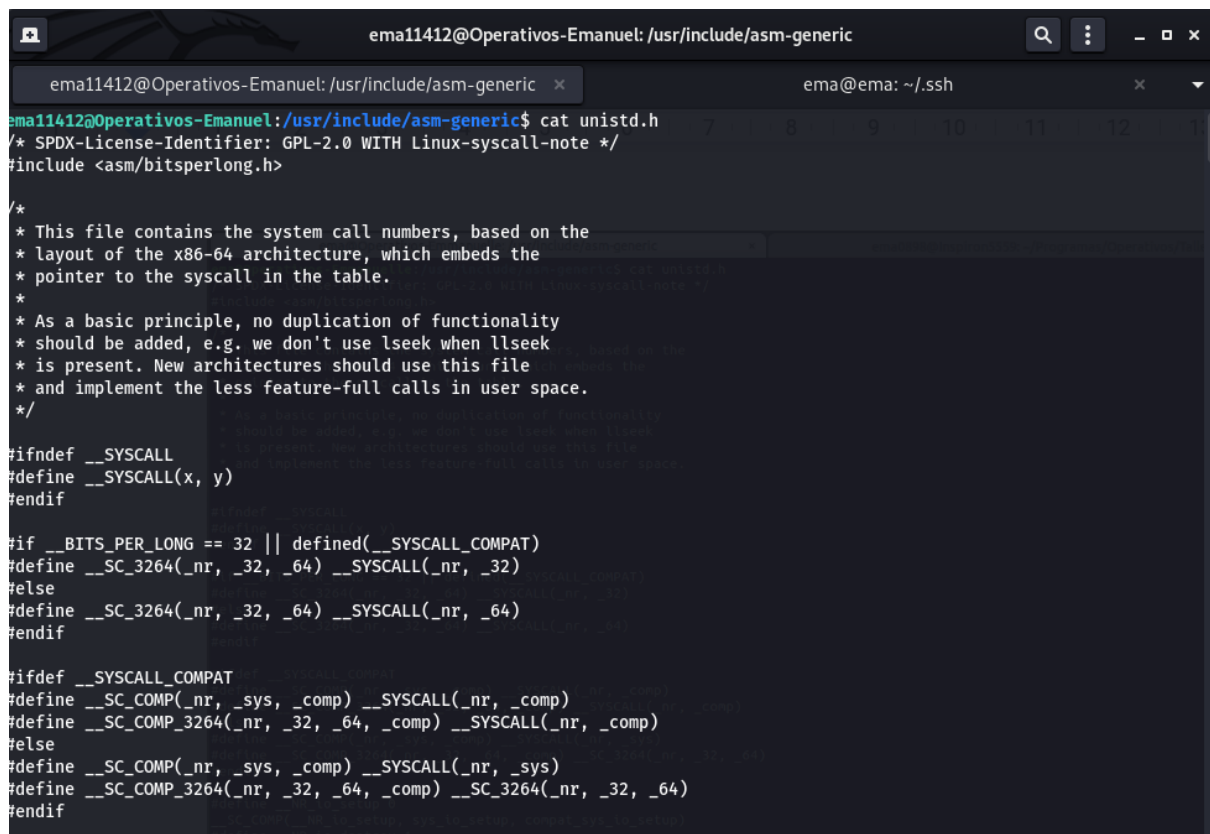


Preguntas guía:

1. Es una función o método el cual funciona para solicitar un proceso del sistema operativo, esto provee una interfaz entre los programas y el sistema.
2. Se diferencian en que la interrupción salta esporádicamente mientras que se tiene activo un proceso, lo cual suspende temporalmente este, la llamada al sistema se invoca a gusto del usuario.
3. Tenemos las siguientes:
 - a. read: Leer un archivo
 - b. write: Escritura
 - c. close: cerrar un fichero
 - d. wait: Espera que un proceso hijo termine, toma su salida y devuelve el pid del hijo que termina.
 - e. getpid: devuelve el id del proceso solicitante
 - f. fork: crea proceso idéntico al padre
 - g. exit: fin de ejecución del proceso
 - h. kill: envía señal a un proceso
 - i. brk(address): fija tamaño de segmento de datos a dirección
 - j. pause: suspende el solicitante hasta la próxima señal
4. Posee más de 200 llamadas al sistema promedio pero por ejemplo linux 3.7 tiene 393 llamadas

Ejercicios máquina virtual.

Archivo con llamadas al sistema:



```
ema11412@Operativos-Emanuel: /usr/include/asm-generic$ cat unistd.h
/* SPDX-License-Identifier: GPL-2.0 WITH Linux-syscall-note */
#include <asm/bitsperlong.h>

/*
 * This file contains the system call numbers, based on the
 * layout of the x86-64 architecture, which embeds the
 * pointer to the syscall in the table.
 *
 * As a basic principle, no duplication of functionality
 * should be added, e.g. we don't use lseek when llseek
 * is present. New architectures should use this file
 * and implement the less feature-full calls in user space.
 */

#ifndef __SYSCALL
#define __SYSCALL(x, y)
#endif

#if __BITS_PER_LONG == 32 || defined(__SYSCALL_COMPAT)
#define __SC_3264(_nr, _32, _64) __SYSCALL(_nr, _32)
#else
#define __SC_3264(_nr, _32, _64) __SYSCALL(_nr, _64)
#endif

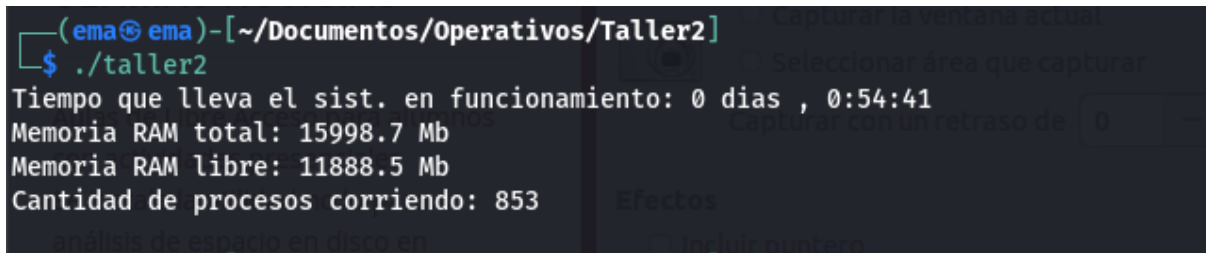
#ifndef __SYSCALL_COMPAT
#define __SC_COMP(_nr, _sys, _comp) __SYSCALL(_nr, _comp)
#define __SC_COMP_3264(_nr, _32, _64, _comp) __SYSCALL(_nr, _comp)
#else
#define __SC_COMP(_nr, _sys, _comp) __SYSCALL(_nr, _sys)
#define __SC_COMP_3264(_nr, _32, _64, _comp) __SC_3264(_nr, _32, _64)
#endif
```

Figura 1: Llamadas al sistema de la VM

Llamadas al sistema.

Llamada	Parámetros	Puntero
read	fd, buf, nbytes	63
write	fd, buf, n	64
fork	void	1079
execve	path, argv, envp	221
exit	status	93

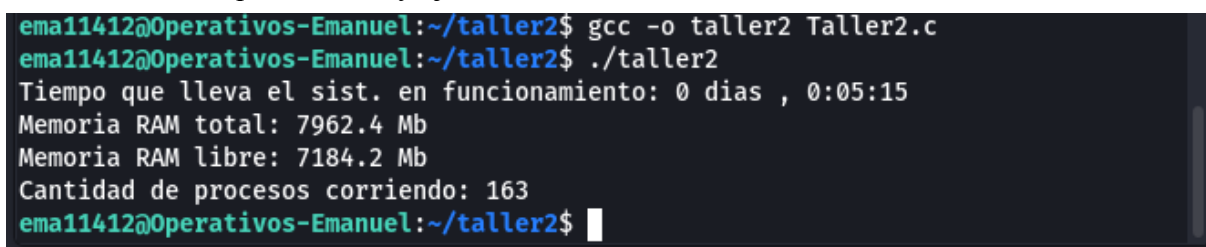
Ejecución del archivo Taller2.c

A terminal window with a dark background. The prompt is (ema@ema)-[~/Documentos/Operativos/Taller2]. The user enters ./taller2. The output shows system statistics: 'Tiempo que lleva el sist. en funcionamiento: 0 dias , 0:54:41', 'Memoria RAM total: 15998.7 Mb', 'Memoria RAM libre: 11888.5 Mb', and 'Cantidad de procesos corriendo: 853'.

```
(ema@ema)-[~/Documentos/Operativos/Taller2]
$ ./taller2
Tiempo que lleva el sist. en funcionamiento: 0 dias , 0:54:41
Memoria RAM total: 15998.7 Mb
Memoria RAM libre: 11888.5 Mb
Cantidad de procesos corriendo: 853
```

Figura 2: Ejecución del Taller2.c en maquina local

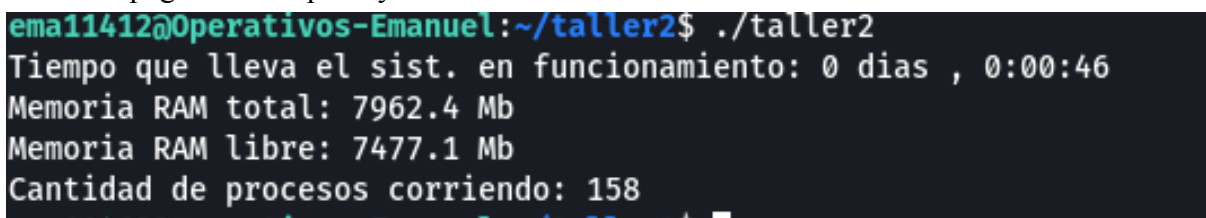
Conexión con maquina virtual y ejecución del archivo Taller2.c

A terminal window with a dark background. The prompt is ema11412@Operativos-Emanuel:~/taller2\$. The user enters gcc -o taller2 Taller2.c, then ./taller2. The output shows system statistics: 'Tiempo que lleva el sist. en funcionamiento: 0 dias , 0:05:15', 'Memoria RAM total: 7962.4 Mb', 'Memoria RAM libre: 7184.2 Mb', and 'Cantidad de procesos corriendo: 163'.

```
ema11412@Operativos-Emanuel:~/taller2$ gcc -o taller2 Taller2.c
ema11412@Operativos-Emanuel:~/taller2$ ./taller2
Tiempo que lleva el sist. en funcionamiento: 0 dias , 0:05:15
Memoria RAM total: 7962.4 Mb
Memoria RAM libre: 7184.2 Mb
Cantidad de procesos corriendo: 163
ema11412@Operativos-Emanuel:~/taller2$
```

Figura 3: Ejecución del Taller2.c en VM

Una vez apagada la maquina y vuelta a encender

A terminal window with a dark background. The prompt is ema11412@Operativos-Emanuel:~/taller2\$. The user enters ./taller2. The output shows system statistics: 'Tiempo que lleva el sist. en funcionamiento: 0 dias , 0:00:46', 'Memoria RAM total: 7962.4 Mb', 'Memoria RAM libre: 7477.1 Mb', and 'Cantidad de procesos corriendo: 158'.

```
ema11412@Operativos-Emanuel:~/taller2$ ./taller2
Tiempo que lleva el sist. en funcionamiento: 0 dias , 0:00:46
Memoria RAM total: 7962.4 Mb
Memoria RAM libre: 7477.1 Mb
Cantidad de procesos corriendo: 158
```

Figura 4: Ejecución del Taller2.c en VM luego de reiniciar.

Como se puede ver en las anteriores se nos muestra la memoria RAM total y la libre por el sistema en ese momento, a diferencia con la VM mi maquina local tiene muchos mas procesos ya que esta haciendo mas cosas ademas de lo basico del sistema, la VM solo esta encendida haciendo lo basico en el sistema.

Al encenderla podemos ver un pequeño cambio, ya que algunos procesos estan terminando o iniciando pero es mínimo.

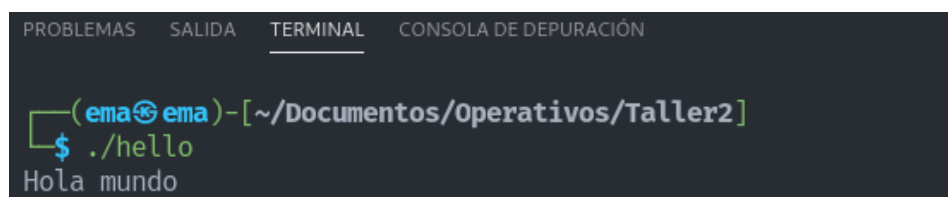
Imprimir hola mundo sin printf

```
#include <stdlib.h>
#include <unistd.h>

int main(){
    const char msg[] = "Hola mundo\n";
    write(STDOUT_FILENO, msg, sizeof(msg));
    return 0;
}
```

Figura 5: Código implementado hola mundo sin printf

Muestra en consola.



```
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

(ema@ema)~[~/Documentos/Operativos/Taller2]
$ ./hello
Hola mundo
```

Figura 6: Hola mundo en consola.

Llamada al sistema 'Hola SO version'

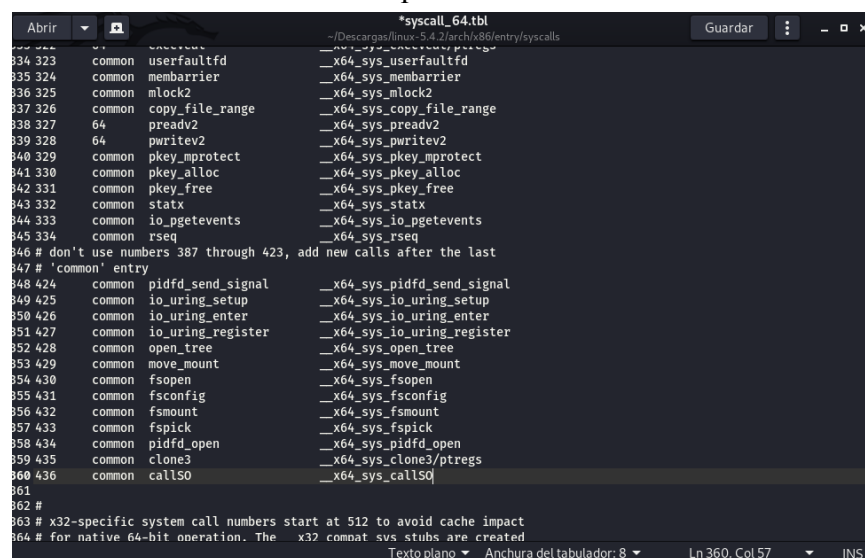
Se debe descargar la carpeta del siguiente link:

<https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.4.2.tar.xz>

Una vez descomprimida debemos de ir a la ruta:

linux-5.4.2/arch/x86/entry/syscalls

Una vez acá se abre el archivo syscall_64.tbl, dependiendo podría ser 64 o 32. Aca se busca las llamadas al sistema se busca una que no este utilizando.



```
Abrir  [icon]  *syscall_64.tbl  Guardar  [icon]  [icon]  [icon]
~/Descargas/linux-5.4.2/arch/x86/entry/syscalls

324 common userfaultfd      _x64_sys_userfaultfd
325 common membarrier       _x64_sys_membarrier
326 common mlock2            _x64_sys_mlock2
327 common copy_file_range   _x64_sys_copy_file_range
328 64 preadv2               _x64_sys_preadv2
329 64 pwritev2              _x64_sys_pwritev2
329 common pkey_mprotect     _x64_sys_pkey_mprotect
330 common pkey_alloc        _x64_sys_pkey_alloc
331 common pkey_free         _x64_sys_pkey_free
332 common statx             _x64_sys_statx
333 common io_pgetevents     _x64_sys_io_pgetevents
334 common rseq              _x64_sys_rseq
46 # don't use numbers 387 through 423, add new calls after the last
47 # 'common' entry
48 424 common pidfd_send_signal _x64_sys_pidfd_send_signal
49 425 common io_uring_setup   _x64_sys_io_uring_setup
50 426 common io_uring_enter   _x64_sys_io_uring_enter
51 427 common io_uring_register _x64_sys_io_uring_register
52 428 common open_tree        _x64_sys_open_tree
53 429 common move_mount      _x64_sys_move_mount
54 430 common fsopen           _x64_sys_fsopen
55 431 common fsconfig          _x64_sys_fsconfig
56 432 common fsmount          _x64_sys_fsmount
57 433 common fspick           _x64_sys_fspick
58 434 common pidfd_open       _x64_sys_pidfd_open
59 435 common clone3           _x64_sys_clone3_ptregs
60 436 common callSO          _x64_sys_callSO
61
62 #
63 # x32-specific system call numbers start at 512 to avoid cache impact
64 # for native 64-bit operation. The x32 compat sys stubs are created

Texto plano  Anchura del tabulador: 8  Ln 360, Col 57  INS
```

Se agregó en la llamada 436.

Seguido de lo anterior se modifica el archivo.

/linux-5.4.2/include/linux/syscalls.h

Agregando al final del documento lo siguiente.

```
1402     return old;
1403 }
1404
1405 /* for __ARCH_WANT_SYS_IPC */
1406 long ksys_semtimedop(int semid, struct sembuf __user *tsops,
1407                     unsigned int nsops,
1408                     const struct __kernel_timespec __user *timeout);
1409 long ksys_semget(key_t key, int nsems, int semflg);
1410 long ksys_old_semctl(int semid, int semnum, int cmd, unsigned long arg);
1411 long ksys_msgget(key_t key, int msgflg);
1412 long ksys_old_msgctl(int msqid, int cmd, struct msqid_ds __user *buf);
1413 long ksys_msgrcv(int msqid, struct msgbuf __user *msgp, size_t msgsz,
1414                 long msgtyp, int msgflg);
1415 long ksys_msgsnd(int msqid, struct msgbuf __user *msgp, size_t msgsz,
1416                 int msgflg);
1417 long ksys_shmget(key_t key, size_t size, int shmflg);
1418 long ksys_shmctl(char __user *shmaddr);
1419 long ksys_old_shmctl(int shmid, int cmd, struct shmid_ds __user *buf);
1420 long compat_ksys_semtimedop(int semid, struct sembuf __user *tsems,
1421                             unsigned int nsops,
1422                             const struct old_timespec32 __user *timeout);
1423
1424 asmlinkage int sys_calls0(void);
1425
```

Luego de esto se modifica el archivo

/linux-5.4.2/kernel/sys.c

Ahí agregamos lo siguiente al final del archivo:

```
SYSCALL_DEFINE1(calls0, struct new_utsname __user *, name)
{
    struct new_utsname tmp;
    down_read(&uts_sem);
    memcpy(&tmp, utsname(), sizeof(tmp));
    up_read(&uts_sem);
    printk("Hola S0: %s\n", &tmp, &tmp → release);

    return 0;
}
#endif /* CONFIG_COMPAT */
```

Por último hacemos make de la carpeta, en la dirección /linux-5.4.2 con lo siguientes comandos

```
sudo make -jn
```

```
sudo make modules_install install -jn
```

Por último se prueba con el comando

```
sudo dmesg
```

Referencias:

González, O. (2018). What is system call in operating system?. Retrieved from <https://www.guru99.com/system-call-operating-system.html>

Linux Team. (2017). Linux programmer's manual sysinfo(). Retrieved from <http://man7.org/linux/man-pages/man2/sysinfo.2.html>