LAB 2
SYSTEM CALLS

PROGRAMMAZIONE DI SISTEMA (JZ-ZZ)

2023/24

ANDREA PORTALURI

OBIETTIVI DEL LABORATORIO

- Eseguire programmi utente
- Implementare read, write e exit come system calls

ESEGUIRE UN PROGRAMMA UTENTE

I programmi utente in OS161 sono richiamabili mediante il comando:

```
>> p <path/del/programma>
```

(es., `p testbin/palin`)

Nella versione base di OS161 alcuni non sono eseguibili in modo corretto perchè manca il supporto per:

- System calls `read`, `write` e `_exit`
- Gestione della memoria virtuale con restituizone di memoria allocata
- Argomenti al `main` (`argc`, `argv`)

SYSTEM CALLS

Si esegua, a tale scopo, il programma *testbin/palin*.

>> p testbin/palin

```
andrea@andrea-VM: ~/os161/roo
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 55
Unknown syscall 3
Fatal user mode trap 4 sig 10 (Address error on load,
e00f)
panic: I don't know how to handle this
sys161: trace: software-requested debugger stop
sys161: Waiting for debugger connection...
```

Vengono effettuate le chiamate a SYS_write e SYS__exit ma non sono implementate

/kern/include/kern/syscall.h

```
21 //                                  -- Proces
22 #define SYS_fork          0
23 #define SYS_vfork         1
24 #define SYS_execv         2
25 #define SYS__exit         3
26 #define SYS_waitpid       4
27 #define SYS_getpid        5
28 #define SYS_getppid       6
29 //                                  (virtual
30 #define SYS_sbrk          7
31 #define SYS_mmap          8
32 #define SYS_munmap        9
33 #define SYS_mprotect      10
34
35 #define SYS_open          45
36 #define SYS_pipe          46
37 #define SYS_dup           47
38 #define SYS_dup2          48
39 #define SYS_close         49
40 #define SYS_read          50
41 #define SYS_pread         51
42 //#define SYS_readv       52
43 //#define SYS_preadv      53
44 #define SYS_getdirentry   54
45 #define SYS_write         55
46 #define SYS_pwrite        56
47 //#define SYS_writev      57
48 //#define SYS_pwritev     58
49 #define SYS_lseek         59
```

/kern/arch/mips/syscall/syscall.c

```
switch (callno) {
    case SYS_reboot:
        err = sys_reboot(tf->tf_a0);
    break;

    case SYS___time:
        err = sys___time((userptr_t)tf->tf_a0, (userptr_t)tf->tf_a1);
    break;

    /* Add stuff here */

    default:
        kprintf("Unknown syscall %d\n", callno);
        err = ENOSYS;
    break;
}
```

Manca l'implementazione delle syscalls

IMPLEMENTARE READ E WRITE

- Si chiede di implementare due funzioni `sys_write()` e `sys_read()` (ricalcando i prototipi di `read()` e `write()`) in un file *kern/syscall/file_syscalls.c* (limitare I/O per stdout e stdin e ricorrere alle funzioni `putch()` e `getch()`) gestendo le inclusioni necessarie.
- Modificare i file di configurazioni del kernel e *conf.kern* definendo e aggiungendo le opzioni necessarie (si consiglia di creare un nuovo kernel).
- Aggiungere e gestire il *case* per le syscalls implementate in */kern/arch/mips/syscall/syscall.c*

SYSTEM CALL READ

```
#include <types.h>
#include <kern/unistd.h>
#include <clock.h>
#include <copyinout.h>
#include <syscall.h>
#include <lib.h>

int sys_read(int fd, userptr_t buf_ptr, size_t size) {
    int i;
    char *p = (char *)buf_ptr;
    if (fd!=STDIN_FILENO) {
        kprintf("sys_read supported only to stdin\n");
        return -1;
    }
    for (i=0; i<(int)size; i++) {
        p[i] = getch();
        if (p[i] < 0)
            return i;
    }
return (int)size;
}
```

Limitare la lettura dal solo STDIN (altri tipi di fd verranno implementati successivamente)

SYSTEM CALL WRITE

```
#include <types.h>
#include <kern/unistd.h>
#include <clock.h>
#include <copyinout.h>
#include <syscall.h>
#include <lib.h>

int sys_write(int fd, userptr_t buf_ptr, size_t size) {
    int i;
    char *p = (char *)buf_ptr;
    if (fd!=STDOUT_FILENO && fd!=STDERR_FILENO) {
        kprintf("sys_write supported only to stdout\n");
        return -1;
    }
    for (i=0; i<(int)size; i++)
        putch(p[i]);
    return (int)size;
}
```

Limitare la scrittura ai soli STDOUT e STDERR

IMPLEMENTARE EXIT

- Si chiede di implementare la funzione `sys__exit()` (chiamata implicitamente al termine del `main`) in modo analogo a `sys_read()` e `sys_write()` in un file *kern/syscall/file_syscalls.c*
- Aggiungere il *case* per `SYS__exit` (con doppio underscore) in */kern/arch/mips/syscall/syscall.c*
- Realizzare una versione ridotta che sia almeno in grado di effetturare `as_destroy()` e chiusura del thread con `thread_exit()` (non si richiede il salvataggio dello stato).

SYSTEM CALL EXIT

```
#include <types.h>
#include <kern/unistd.h>
#include <clock.h>
#include <copyinout.h>
#include <syscall.h>
#include <lib.h>
#include <proc.h>
#include <thread.h>
#include <addrspace.h>

void sys__exit(int status) {
    struct addrspace *as = proc_getas();
    as_destroy(as);
    thread_exit();
    panic("thread_exit returned (this should not happen)\n");
    (void) status; // TODO: status handling
}
```

Recupera l'address space del processo corrente e lo libera

Non è richiesta il salvataggio dello stato (per ora)