LAB 2
SYSTEM CALLS E GESTIONE DELLA MEMORIA

PROGRAMMAZIONE DI SISTEMA (JZ-ZZ)

2022/23

ANDREA PORTALURI

OBIETTIVI DEL LABORATORIO

- Eseguire programmi utente
- Implementare read, write e exit come system calls
- Primi passi nella gestione della memoria

ESEGUIRE UN PROGRAMMA UTENTE

I programmi utente in OS161 sono richiamabili mediante il comando:

```
>> p <programma>
```

(es., `p testbin/palin`)

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 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 il supporto per le syscalls

SYSTEM CALLS (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 IO su 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 CALLS (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` 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).

GESTIONE DELLA VM

OS161 base contiene un gestore della memoria virtuale che effettua unicamente allocazione contigua di memoria reale senza mai rilasciarla (*kern/arch/mips/vm/dumbvm.c*) tramite `getppages()` e `ram_stealmem()`.

GESTIONE DELLA VM

`ram_stealmem` *(kern/arch/mips/vm/ram.c)*

```
paddr_t ram_stealmem (unsigned long npages) {
    paddr_t paddr;
    size_t size = npages * PAGE_SIZE;
    if (firstpaddr + size > lastpaddr)
        return 0;
    paddr = firstpaddr;
    firstpaddr += size;
    return paddr;
}
```

`npages` è il numero di pagine richieste al SO

Controlla se il primo indirizzo fisico (`firstpaddr`) + la `size` richiesta è maggiore dell'ultimo indirizzo fisico disponibile (`lastpaddr`). Se sì, fallisce nell'allocazione di memoria

Se l'allocazione ha successo, la funzione ritorna l'indirizzo fisico iniziale e aggiorna il primo nuovamente disponibile.

GESTIONE DELLA VM

`getppages` *(kern/arch/mips/vm/dumbvm.c)*

```
static paddr_t
getppages (unsigned long npages) {
    paddr_t addr;
    spinlock_acquire(&stealmem_lock);
    addr = ram_stealmem(npages);
    spinlock_release(&stealmem_lock);
    return addr;
}
```

`getppages` chiede il numero di pagine fisiche e ritorna un puntatore in memoria.

`spinlock` permette la mutua esclusione in case di processi multipli che accedono in memoria.

GESTIONE DELLA VM

Si consiglia di realizzare:

- un vettore di flag (int o char) da inizializzare in `as_prepare_load()` e `alloc_kpages()`.
- la restituzione di memoria in `is_destroy()` e `free_kpages()`.
- modificare `getppages()` e/o `ram_stealmem()`.

Attenzione: `getppages()` viene chiamata sia da `kmalloc()` -> `alloc_kpages()` che da `as_prepare_load()`.