

CONCEPTES AVANÇATS DE SISTEMES OPERATIUS (CASO)

Facultat d'Informàtica de Barcelona, Dept. d'Arquitectura de Computadors, curs 2019/20 – 2Q

Pràctiques de laboratori

Mach i GNU Hurd

Material

Un cop teniu el vostre sistema funcionant correctament, farem la instal·lació del sistema operatiu Debian/GNU Hurd en un entorn virtualitzat mitjançant Qemu (Alternativa 1) o VirtualBox (Alternativa 2).

Una instal·lació ja feta de Hurd (versió comprimida, cal descomprimir-la amb `tar -xvzf`, el fitxer que se n'extreu es dirà `debian-hurd-20171101.img`):

<http://people.debian.org/~sthibault/hurd-i386/debian-hurd.img.tar.gz>

Els exemples de codi sobre Mach que podeu obtenir de la web de l'assignatura

<http://docencia.ac.upc.edu/FIB/grau/CASO/labs-2019-20-2q/codi-lab-hurd.tar.bz2>

Una petita història sobre Hurd:

<http://www.linuxuser.co.uk/features/whatever-happened-to-the-hurd-the-story-of-the-gnu-os>

Una consideració que fa plantejar si aquesta pràctica serà útil en el futur...

https://www.gnu.org/software/hurd/history/port_to_another_microkernel.html

Alternativa 1: Instal·lació sobre Qemu

Abans de continuar, decidiu si fareu servir Qemu o VirtualBox. Si us decidiu per Qemu, continueu per aquesta secció. Si us decidiu per VirtualBox, aneu a la secció de l'Alternativa 2.

Instal·leu el Qemu per x86_64 (64-bits) i i386 (32-bits). Podeu fer la instal·lació de binaris pel vostre sistema:

```
$ sudo apt-get install qemu-kvm
```

Un cop instal·lat, comproveu que podeu executar les comandes `qemu-system-i386` i `qemu-system-x86_64`.

Instal·lació de Debian/GNU Hurd en Qemu

Ara instal·larem Debian/GNU Hurd per executar-lo dins l'entorn virtual del qemu en mode 32 bits. Seguiu les instruccions de:

<http://www.gnu.org/software/hurd/hurd/running/qemu.html>

Per engegar el qemu amb Hurd, feu servir una comanda com aquesta:

```
$ qemu-system-i386 -m 1024 -net nic,model=rtl8139 -net user,hostfwd=tcp::5555-:22 \
    -drive cache=writeback,index=0,media=disk,file=debian-hurd-20200101.img \
    -vga vmware
```

Alternativament (com a root):

```
$ kvm -m 1024 -net nic,model=rtl8139 -net user,hostfwd=tcp::5555-:22 \
      -drive cache=writeback,index=0,media=disk,file=debian-hurd-20200101.img \
      -vga vmware
```

Podeu posar-la en un shell script per facilitar arrencar-lo més endavant.

Alternativa 2: Instal·lació sobre VirtualBox

(Funciona per Ubuntu 19.10)

Instal·leu Virtual Box d'Oracle: https://www.virtualbox.org/wiki/Linux_Downloads

Un cop instal·lat, comproveu que disposeu de les comandes VirtualBox i vbox-img.

Instal·lació de Debian/GNU Hurd en VirtualBox

Abans d'instal·lar Debian/GNU Hurd, caldrà transformar la imatge que ens hem baixat del seu disc a un format que VirtualBox pugui entendre. El fitxer `debian-hurd-20200101.img` està en format “raw”, és una còpia directa dels bytes d'un disc:

```
$ file debian-hurd-20200101.img
```

```
debian-hurd-20200101 .img: DOS/MBR boot sector
```

La passarem a format Virtual Disk Image (“vdi”) amb aquesta comanda:

```
$ vbox-img convert --srcformat RAW --srcfilename debian-hurd-20200101.img \
                  --dstformat VDI --dstfilename debian-hurd-20200101.img.vdi
```

```
$ file debian-hurd-20200101.img.vdi
```

```
debian-hurd-20200101.img.vdi: VDI Image version 1.1 (<<< Oracle VM VirtualBox Disk Image >>>),
3146776576 bytes
```

Ara entreu al VirtualBox i usant el seu entorn gràfic, creeu una màquina virtual amb aquest fitxer resultat (`debian-hurd-20200101.img.vdi`), **connectat al controlador IDE** (al costat d'un CD-ROM – Optical drive).

Arranqueu la màquina usant el botó “Start”.

Configuració bàsica de Debian/GNU Hurd

Un cop ha arrencat el Hurd en un dels dos entorns de virtualització, entreu com a **root** i **poseu-vos un password**, que per defecte el sistema de la màquina virtual no en porta. Hi ha també un usuari **demo**, que podeu fer servir per fer les proves com a usuari normal no administrador. **Poseu-li password també**, per poder entrar des del host per ssh.

El teclat de la consola no estarà ben assignat, des de l'usuari “root” podeu fer:

```
$ dpkg-reconfigure keyboard-configuration
```

i seleccionar el teclat espanyol. Entreu a l'eina de configuració i feu-ho així:

- Seleccioneu el teclat Generic 105-key (Intl) PC
- A keyboard layout seleccioneu l'última opció: other
- Busqueu el teclat “Spanish” i seleccioneu-lo
- Seleccioneu l'opció Spanish - Catalan
- Accepteu els valors per defecte per les tecles AltGr, Compose i la combinació Ctrl-Alt-Backspace (compte perquè aquesta darrera l'agafarà el gestor del host i fa acabar el gestor d'X-Windows, amb la qual cosa podeu perdre feina feta).

Per carregar la nova configuració del teclat, feu:

- `/etc/init.d/hurd-console restart`

Canvieu també el fitxer `/etc/ssh/sshd_config`, de forma que la línia 32:

```
PermitRootLogin prohibit_password
```

passi a ser:

```
PermitRootLogin yes
```

ALTERNATIVA 1 (Qemu) Amb l'opció “-net user,hostfwd=tcp::5555-:22” que li heu passar al qemu, aquest redirigeix el port 5555 del host al port d'SSH de la màquina virtual, amb la qual cosa, podeu usar *secure shell* per connectar-vos-hi.

ALTERNATIVA 2 (VirtualBox) Per fer el mateix tipus de redirecció de port en el VirtualBox, entreu a la secció “Network” i a l’Adapter 1” afegiu un “Port Forwarding”, com aquest:

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
Rule 1	TCP		5555		22

Ara des del host, podeu fer:

```
$ ssh -p 5555 root@localhost # opció recomanada per treballar i disposar de diverses sessions
```

Si no funciona, assegureu-vos que heu canviat correctament el `/etc/ssh/sshd_config`, indicant:
`PermitRootLogin yes`

També podeu usar aquest port per realitzar transferències de fitxers entre el host i la màquina virtual:

Host a Debian/GNU Hurd
\$ scp -P 5555 <fitxer> root@localhost:

Debian/GNU Hurd cap al Host
\$ scp -P 5555 root@localhost:<fitxer> ./

És recomanable treballar amb aquestes connexions remotes.

Copieu el codi d'exemple que us donem (codi-lab-hurd.tar.bz2)¹, on teniu també el Makefile que us permetrà compilar el codi font de tots els exercicis.

La interfície de Mach

A Hurd, tenim una estructura de sistema basada en microkernel (Figura 1).

En aquesta estructura hi ha dues interfícies ben diferenciades: Hurd (compatible amb UNIX/Linux) i Mach. Veieu-ne alguns exemples (indiquem la interfície de Mach en negreta):

- getpid(), **mach_task_self()**², **mach_thread_self()**³, retornen la identificació del procés, task, thread.
- fork(), **task_create**⁴(), **thread_create**⁵(), per a crear processos, tasks i threads.
- mmap(), **vm_allocate**⁶(), per demanar memòria.
- ...

1 <http://docencia.ac.upc.edu/FIB/grau/CASO/lab2014/codi-lab-hurd.tar.bz2>

2 http://docencia.ac.upc.edu/FIB/grau/CASO/slides2015/kernel_interface.pdf (pàg 194)

3 << (pàg. 161)

4 << (pàg. 195)

5 << (pàg. 166)

6 << (pàg. 74)

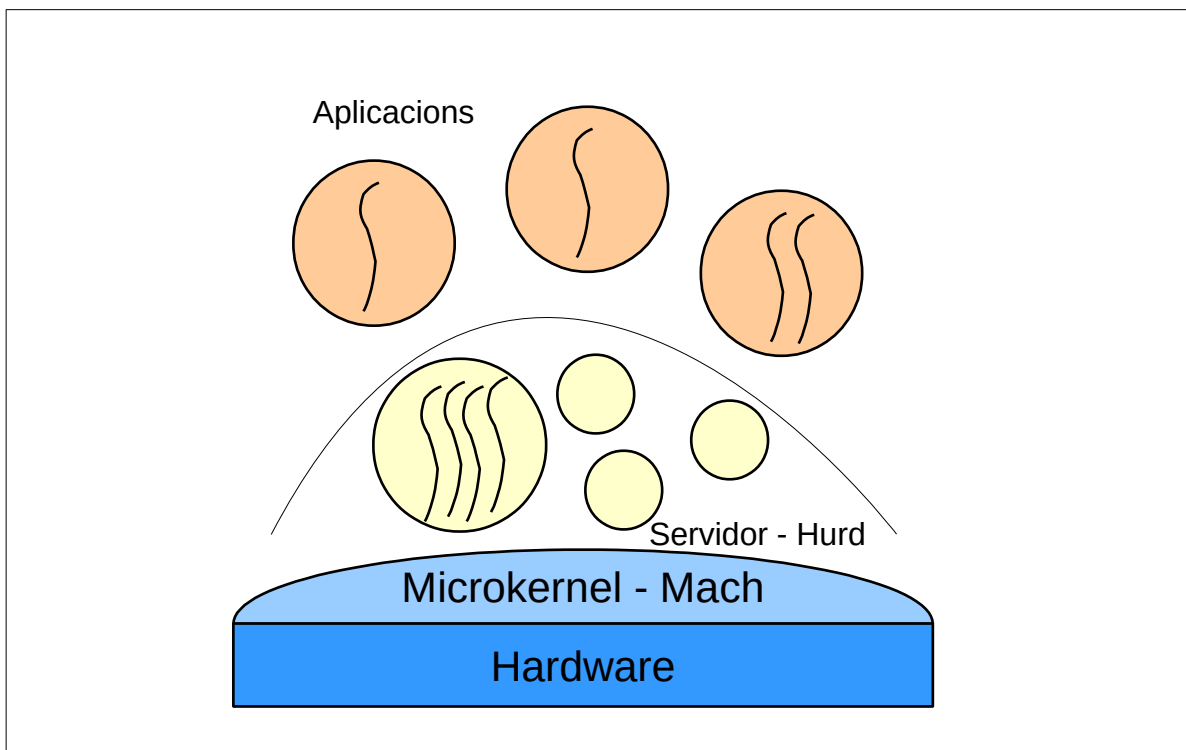


Figura 1: Estructura del sistema operatiu Mach-Hurd.

En Mach, totes les abstraccions es representen per un identificador de **tipus port**. Un port és una entitat a la qual es poden enviar missatges. D'aquesta forma cada entitat té un servidor que llegeix els missatges enviats als ports que implementa i d'aquesta forma es poden fer operacions sobre elles. Com podeu veure en el manual del Kernel Interface, hi ha algunes entitats més que en UNIX:

- port (comunicacions)
- vm (virtual memory, o gestió de l'espai d'adreces)
- memory_object (mapeig de dades sobre l'espai d'adreces, o gestió de la memòria virtual)
- task (entorn de procés)
- host (gestió de la màquina)
- processor_set (conjunt de processadors)
- processor (processador)
- device (gestió de dispositiu)
- thread (flux d'execució)

Aquesta seria la representació de les abstraccions del sistema, incloent els seus ports identificadors:

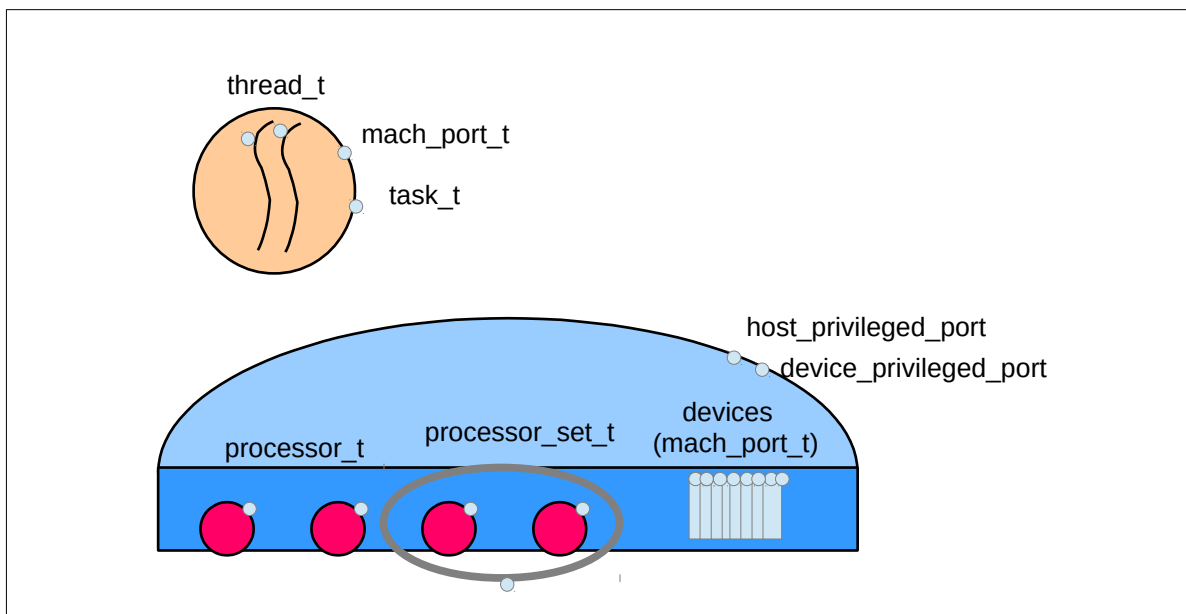


Figura 2: Identificació de cada abstracció del sistema operatiu

Exemple: com obtenir la llista de processadors, veure també el programa de la Figura 3. Per aconseguir la llista de processadors cal utilitzar la crida **host_processors**⁷. Aquesta crida té la següent interfície:

```
kern_return_t host_processors (mach_port_t      host_priv,
                              processor_array_t* processor_list,
                              mach_msg_type_number_t* processor_count);
```

On *host_priv* és el *host_privileged_port*. La crida torna la llista de processadors en una taula (*array*) de memòria reservada des del sistema, en l'espai d'adreces del procés, i el número de processadors que controla el sistema. Les crides a Mach que tornen informació reservant memòria d'aquesta manera les podeu identificar en el manual del Kernel Interface perquè:

- El paràmetre que retorna la informació està especificat com “[out pointer to dynamic array of <tipus de dades elemental que retorna>]”
- El paràmetre que retorna la quantitat d'elements retornats s'especifica com “[out scalar]”. Si la memòria l'ha de proporcionar l'aplicació, llavors aquest darrer paràmetre és “[in/out scalar]” i s'utilitza en entrada per indicar al sistema la quantitat d'elements que caben en la memòria proporcionada i en sortida del sistema, aquest ens indica quants elements hi ha copiat realment.

Hi ha una crida a sistema especial per obtenir els ports privilegiats, en particular el *host_privileged_port*: *get_privileged_ports*. Aquesta crida va al servidor de Hurd, que comprova si el procés té privilegis suficients per tornar-li o no els ports privilegiats. En el nostre cas, si el procés que fa la crida no és de l'administrador (root), la crida no li retornarà els ports, sino aquest error: *Error getting privileged ports (0x40000001), Operation not permitted*. **Recordeu que és molt important comprovar els errors que ens poden tornar les crides que fa el nostre programa.**

Gestió de memòria

En Mach, hi ha crides a sistema que, tot i no estar relacionades amb la gestió de la memòria, retornen nova memòria assignada al procés. La crida **host_processors** n'és un exemple. Les podeu distingir

⁷ http://docencia.ac.upc.edu/FIB/grau/CASO/slides2015/kernel_interface.pdf (pàg 227)

perquè estan definides amb un paràmetre “out” (de sortida), com aquesta, en format MIG de <mach/mach_host.defs>:

```
routine host_processors (      host_priv      : host_priv_t;
                             out processor_list : processor_array_t);
```

que es tradueix en (<mach/mach_host.h>):

```
kern_return_t host_processors (      mach_port_t      host_priv,
                                   processor_array_t * processor_list,
                                   mach_msg_type_number_t * processor_listCnt);
```

on ... processor_listCnt és un simple punter a un enter sense signe:

```
typedef unsigned int mach_msg_type_number_t;
```

La crida genèrica per demanar memòria és molt versàtil, similar al mmap de UNIX/Linux, aquesta és la seva interfície:

```
kern_return_t vm_map (      mach_port_t      target_task,
                           vm_address_t * address,          // where to allocate mem
                           vm_size_t      size,
                           vm_address_t alignment_mask,     // desired alignment
                           boolean_t anywhere,              // fixed address or not
                           mach_port_t memory_object,        // optional manager
                           vm_offset_t offset,               // inside memory_obj
                           boolean_t copy,                  // copy or shared
                           vm_prot_t cur_protection,
                           vm_prot_t max_protection,
                           vm_inherit_t inheritance          // for children tasks
                           );
```

```

#include <mach.h>
#include <mach_error.h>
#include <mach/mig_errors.h>
#include <mach/thread_status.h>
#include <stdio.h>
#include <stdlib.h>
#include <hurd.h>

// compile with gcc -D_GNU_SOURCE -O proc.c -o proc

processor_array_t processor_list = NULL;
mach_msg_type_number_t processor_listCnt = 0;

int main ()
{
    int res, i;
    mach_port_t host_privileged_port;
    device_t device_privileged_port;

    res = get_privileged_ports(&host_privileged_port, &device_privileged_port);
    if (res != KERN_SUCCESS) {
        printf ("Error getting privileged ports (0x%x), %s\n", res,
            mach_error_string(res));
        exit(1);
    }

    printf ("privileged ports: host 0x%x devices 0x%x\n",
        host_privileged_port, device_privileged_port);

    printf ("Getting processors at array 0x%x\n", processor_list);

    res = host_processors(host_privileged_port,
        &processor_list, &processor_listCnt);
    if (res != KERN_SUCCESS) {
        printf ("Error getting host_processors (0x%x), %s\n", res,
            mach_error_string(res));
        exit(1);
    }

    printf ("    processors at array 0x%x\n", processor_list);
    printf ("processor_listCnt %d\n", processor_listCnt);

    for (i=0; i < processor_listCnt; i++)
        printf ("processor_list[%d] 0x%x\n", i, processor_list[i]);
}

```

Figura 3: Exemple de codi per obtenir la llista de processadors.

Exercicis

1. Comproveu que el programa `proc.c` funciona correctament per l'usuari `root`, però dóna l'error indicat anteriorment si l'executa un usuari no privilegiat (podeu usar l'usuari “demo”).
2. Quin processador indica que tenim el programa `proc.c`? Busqueu a `<mach/machine.h>` els codis de “CPU Type” i “CPU Subtype”.
3. Expliqueu les altres característiques del processador que mostra `proc.c`. Obtingueu-les del fitxer `<mach/processor_info.h>`; localitzeu-hi l'estructura `processor_basic_info`.
4. Comproveu que el programa `memory-management.c` dóna errors al compilar... com els podeu arreglar? (pista: falta una coma (,) a la línia 60).

Són clars els missatges d'error que dóna el compilador GCC en aquesta situació?

5. Un cop arreglat el problema de la pregunta anterior, comproveu que el programa `memory-management.c` funciona correctament. Aquest programa usa **`host_processors`** i **`vm_map`** de forma intercalada, per demanar memòria 8 cops. L'ús de `processor_info` per demanar memòria queda fora del seu ús habitual, però funciona correctament. Responen:
 1. Quanta memòria assigna al procés cada crida a `host_processors`?
 2. Quanta memòria assigna al procés cada crida a `vm_map`?
 3. Quines adreces ens dóna el sistema en cada crida (`host_processors` i `vm_map`)?
 4. Són pàgines consecutives? (pista: us ajudarà, incrementar el número d'iteracions que fa el programa... per veure la seqüència d'adreces més clara)
 5. Quines proteccions podem demanar a l'assignar memòria a un procés Mach? (pista: veieu el fitxer `<mach/vm_prot.h>`)
 6. Canvieu el programa per a que la memòria demanada sigui de només lectura. Quin error us dóna el sistema quan executeu aquesta nova versió del programa?
 7. Després, afegiu una crida a **`vm_protect (...)`** per tal de desprotegir la memòria per escriptura i que el programa torni a permetre les escriptures en la memòria assignada. Proveu la nova versió i comproveu que ara torna a funcionar correctament.

6. [opcional] Feu un nou programa que actui com un 'ps', que llisti les tasks que estan corrent (o que estan aturades) en el sistema. Anomenau-lo 'mps' per aprofitar que ja tenim definida la seva compilació en el fitxer `Makefile`.

Ajuda, aquestes són les crides que heu de fer servir: `get_privileged_ports`, **`processor_set_default`**, **`host_processor_set_priv`**, **`processor_set_tasks`**, **`task_info`**. Podeu usar també la rutina `Print_Task_info` proporcionada en el fitxer `print-task-info.c`.

7. [opcional] Feu un programa "mtask" que rebi una primera opció [-r|-s] i una llista de processos (pids) i els aturi (-s) o els deixi continuar executant-se (-r), usant les crides **task_suspend/task_resume**.

Ajuda: busqueu una crida a Hurd que us permeti passar d'un pid al port (task_t) que identifica la task.

Exemples: `mtask -r 84 105` # fa un `task_resume` de les tasks que pertanyen als processos 84 i 105

`mtask -s 58 206 87` # atura l'execució dels processos 58, 206 i 87.

8. Feu un programa que creï un flux (**thread_create**) i li canviï l'estat (uesp, eip) amb les crides **thread_get_state** i **thread_set_state**, per engegar-lo posteriorment (**thread_resume**).

Trobareu els tipus genèrics (independents de l'arquitectura) relacionats amb el context d'un flux en el fitxer `<mach/thread_status.h>`. La informació específica de com és l'estat d'un thread en la nostra arquitectura i386 la trobareu a `<mach/machine/thread_status.h>`: `struct i386_thread_state`, i `#defines i386_THREAD_STATE(flavor)`, i `i386_THREAD_STATE_COUNT`.

Feu que el flux executi una funció amb un bucle infinit i comproveu amb el 'top' que està consumint processador (el meu top diu %CPU 0.0, però el programa - thread - es situa dalt de tot), abans de destruir-lo (**thread_terminate**):

```
top - 18:21:45 up 10:57, 10 users, load average: 1.18, 0.87, 0.70
Tasks: 59 total, 1 running, 54 sleeping, 0 stopped, 0 zombie
%Cpu(s): 74.4 us, 0.0 sy, 0.0 ni, 25.6 id, 0.0 wa, 0.0 hi, 0.0 si
Kb Mem: 524280 total, 113028 used, 411252 free, 0 buffers
Kb Swap: 177148 total, 0 used, 177148 free, 0 cached
```

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1770 root 20 0 146m 728 0 R 0.0 0.1 0:00.00 thread
 3 root 20 0 417m 19m 0 S 0.0 3.9 0:00.04 ext2fs
24 root 20 0 130m 976 0 S 0.0 0.2 0:00.00 procfs
```

Ara feu que el thread faci un `printf(...)`. Per què us dóna un "bus error"? Podeu esbrinar què passa?

Pista: useu el gdb, i mireu quina instrucció està executant el thread que falla:

```
gdb> thread 1          # permet seleccionar un thread, el primer en aquest cas
gdb> x/10i $eip        # escriu les instruccions que hi ha a partir del PC del thread
gdb> x/10x $esp        # escriu les dades que hi ha a partir de l'adreça continguda en el registre esp
```

9. Observar que en el fitxer `<mach.h>` tenim dues definicions de funcions interessants per resoldre el problema de la pregunta anterior:

```
kern_return_t mach_setup_thread(task_t task, thread_t thread, void * pc,
                                vm_address_t * stack_base, vm_size_t * stack_size);

kern_return_t mach_setup_tls(thread_t thread);
```

... però cap de les dues soluciona el problema...

10. Feu un programa que creï una task (**task_create** / **task_terminate**), i li doni memòria (**vm_allocate**), per després copiar-li una pàgina de dades (**vm_write**).

Si heu fet la comanda 'mps' (de l'apartat 3), comproveu que la vostra task només té la memòria que li heu donat, haurieu d'obtenir una informació com:

```
virtual size 16384  # si li heu demanat 16KB (4 pàgines)
resident size 0
```

Comproveu que amb la comanda 'ps' aquesta task també es veu: `$ ps -e -o pid,stat,sz,rss,args`

```
PID Stat  SZ  RSS Args
1670 p    16K  0  ?
```

11. Feu un programa que accepti un pid i una adreça com a paràmetres, faci un **vm_read** de l'adreça donada en el procés donat i mostri la informació obtinguda.

Creieu que això mateix es pot fer en UNIX/Linux? I en Windows?

12. [opcional] Feu un programa que creï un procés amb *fork()* i faci que pare i fill es comuniquin amb un missatge de Mach, usant **mach_msg_send()** i **mach_msg_receive()**.
13. [opcional] Amplieu el programa de l'apartat 3, de forma que també mostri la informació bàsica dels fluxos de cada task.

Entregueu: Prepareu els programes i les respostes a les preguntes 2, 3, 4, 5, 8, 9, 10 i 11 per pujar-los al Racó.