CONCEPTES AVANÇATS DE SISTEMES OPERATIUS (CASO)

Facultat d'Informàtica de Barcelona, Dept. d'Arquitectura de Computadors, curs 2014/2015 – 1Q

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.

Instal·lació de Qemu

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

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

Instal·lació de Debian/GNU Hurd

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-20140529.img
```

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

Un cop ha arrencat, entreu com a root i poseu-vos un password, que per defecte el sistema de la màquina virtual no en porta.

El teclat de la consola no estarà ben assignat, podeu fer:

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

i seleccionar el teclat espanyol.Haureu de rebootar per que la nova configuració es carregui.

Amb l'opció "-net user,hostfwd=tcp::5555-:22" redirigiu 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. Feu:

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

Per a que això funcioni haureu de permetre-ho en el fitxer de configuració del sshd:

/etc/ssh/sshd_config
…
PermitRootLogin yes
…

I fer-li un reset al servei: $ /etc/init.d/ssh restart

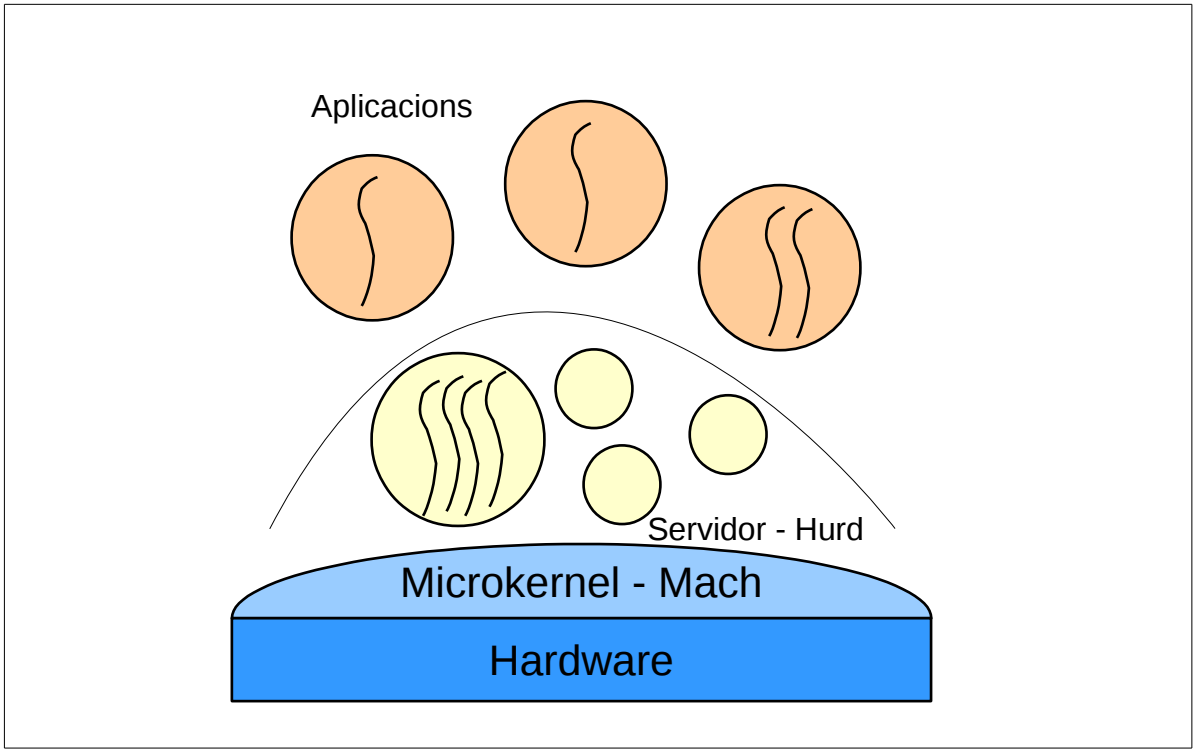
També podeu usar aquest port per realitzar transferències de fitxers entre el host i la màquina virtual:

$ scp -P 5555 <fitxer> root@localhost: $ scp -P 5555 root@localhost:<fitxer> .

I també podeu engegar l'entorn X-Windows: startx, però **pot resultar una mica lent**.

La interfície de Mach

A Hurd, tenim una estructura de sistema basada en microkernel:



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

– getpid(), **mach_task_self**()[1], **mach_thread_self**()[2], retornen la identificació del procés, task, thread.

– fork(), **task_create**[3](), **thread_create**()[4], per a crear processos, tasks i threads.

– mmap(), **vm_allocate**()[5], per demanar memòria.

– ...

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)

– thread (flux d'execució)

– task (entorn de procés)

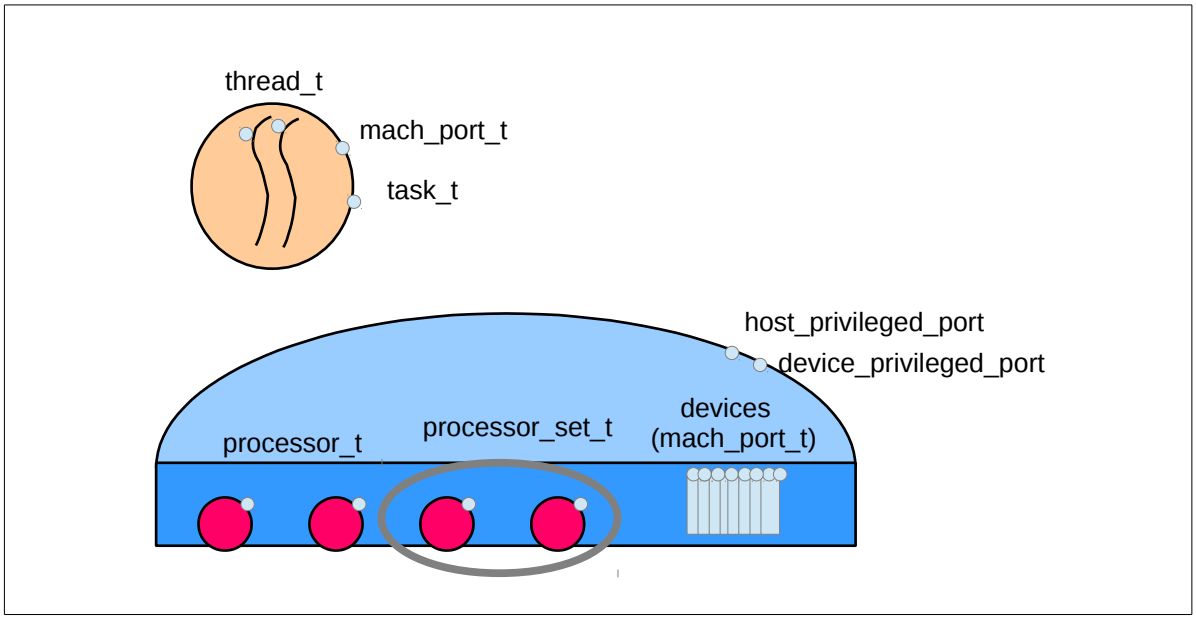
– host (gestió de la màquina)

– processor_set (conjunt de processadors)

– processor (processador)

– device (gestió de dispositiu)

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



1 ftp://ftp.cs.cmu.edu/afs/cs/project/mach/public/doc/osf/kernel_interface.ps (pàg 194, o bé transformeu el fitxer a PDF per poder buscar cadenes de text)
2 << (pàg. 161)
3 << (pàg. 195)
4 << (pàg. 166)
5 << (pàg. 74)

Exemple: com obtenir la llista de processadors. Per aconseguir la llista de processadors cal utilitzar la crida **host_processors**[6]. Aquesta crida té la següent interfície:

```
kern_return_t host_processors    (mach_port_t              host_priv,
                                  processor_array_t*       processor_list,
```

```
#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]);
}
```

```
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. Hi ha una crida a sistema especial per obtenir els ports privilegiats:

6 ftp://ftp.cs.cmu.edu/afs/cs/project/mach/public/doc/osf/kernel_interface.ps (pàg 227) +
$ ps2pdf kernel_interface.ps >kernel_interface.pdf # per poder buscar text còmodament

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.

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.
2. Modifiqueu el programa proc.c per obtenir la informació del processador: processor_basic_info, estructura que trobareu al fitxer <mach/processor_info.h>.
3. Feu un nou programa que actui com un 'ps', que llisti les tasks que estan corrent (o que estan aturades) en el sistema. Anomeneu-lo 'mps'.

   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.
4. [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.
5. 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**). 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(...). Funciona correctament o us dóna un "bus error"? Podeu esbrinar què passa?
6. 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**).

   Comproveu amb el programa 'mps' (fet a l'apartat 3) que la vostra task només té la memòria que li heu donat, haurieu d'obtenir una informació com:

```
virtual size 16384
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  ?
```

7. Feu un programa que accepti un pid i una adreça com a parametres, 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?

8. [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**().

9. [opcional] Amplieu el programa de l'apartat 3, de forma que també mostri la informació bàsica dels fluxos de cada task.

Entregueu

Com a entrega de la pràctica, prepareu el programa i les respostes dels apartats 2, 3, 5, 6 i 7, per pujar-les al Racó.