Practica 2

2. **Quin processador indica que tenim el programa proc.c? Busqueu a <mach/machine.h> els codis de "CPU Type" i "CPU Subtype".**

   El CPU type es 19 i el CPI subtype es 1. Mirant al machines.h veiem que es l'ultim valor possible; podem deduïr que no sap quin tipus de CPU hi ha darrera.

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.**

   Mirant el kernel interface em trobat la definicio de les caracteristiques que ens diu el executable proc.

   En primer lloc veiem els ports privilegiats(0x35 i 0x36) que hi han. Tambe veiem que actualment tenim un unic procesador, amb un identificador (0x37)

   A part del CPU type & subtype tenem un bolean que ens diu si la CPU esta *running*, un altre bolean que diu si aquest es el procesador master () i un numero d'Slot ().

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?**

   El compilador diu que li faltan arguments, es suficientment clar per buscar l'error. Al final, s'utilitza el gcc per compilar el codi C.

5. **Un cop arreglat el problema de la pregunta anterior, comproveu que el programa memorymanagement.c funciona correctament. Aquest programa usa processor_info 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. Responeu:**

   1. **Quanta memòria assigna al procés cada crida a processor_info?**

      La crida a host_processor retorna un processor_t, que mirant el *sizeof* obtenim que ocupa 4 bytes.

   2. **Quanta memòria assigna al procés cada crida a vm_map?**

      8192 bytes. Estan assignat a la interficie de la cirda vm_map.

   3. **Quines adreces ens dóna el sistema en cada crida (processor_info i vm_map)?**

      En el cas de processor_info, la adreça es assignada pel sistema operatiu(ja que realment esta retornant una objecte de tipus processor_t).

      En el cas de vm_map, el punter del parametre *address* esta inicialitzat a NULL i el sistema operatiu ens busca el primer lloc que acompleix el tamany especificat; cal destacar que al document de kernel interface expresa que l'adreça d'inici de la regió de memoria s'arrodoneix a la seguent pàgina si no hi cap a la actual.

   4. **Són pàgines consecutives? (pista: us ajudarà, incrementar el número d'iteracions que fa el programa... per veure la seqüència d'adreces més clara)**

      No, l'explicació es la mateixa que a la pregunta 3.

   5. **Quines proteccions podem demanar a l'assignar memòria a un procés Mach? (pista: veieu el fitxer)**

      Les proteccions possibles son:

      - VM_PROT_READ

- VM_PROT_WRITE
- VM_PROT_EXECUTE

Existeixen dos parametres de protecció, un de max_protection(máxim de permisos que s'apliquen a tot el programa) i un de cur_protection (on defineix el que es pot fer amb la regió.

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?**

   L'error es Segmentation Fault.

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.**

   vm_protect(mach_task_self(), (vm_address_t) &p);

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. Anomeneu-lo 'mps' per aprofitar que ja tenim definida la seva compilació el 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**

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

void printErrorSysCall(int res){

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

task_array_t  myTaskList;
mach_msg_type_number_t count = 0;

int main(int argc, char* argv){

        int res;
        mach_port_t host_privileged_port;
        mach_port_t namePortDefProc;
        device_t device_privileged_port;


        res = get_privileged_ports(&host_privileged_port,
&device_privileged_port);
        printErrorSysCall(res);
```

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

        res = processor_set_default(host_privileged_port, &namePortDefProc);
        printf("Getting the default set at: 0x%x\n", namePortDefProc);
        printErrorSysCall(res);

        //grab permision to the port:
        mach_port_t processor_set;
        res = host_processor_set_priv(host_privileged_port, namePortDefProc,
&processor_set);
        printErrorSysCall(res);
        printf("Get the processor_set 0x%x\n", processor_set);

        //return a list of task assigned to a processor set:
        res = processor_set_tasks(processor_set, &myTaskList, &count);
        printErrorSysCall(res);
        printf("Finded %u tasks in list, @ default processor set\n", count);
        printf("List in 0x%x\n", myTaskList);

        for(int i = 0; i< count; ++i) Print_Task_info(myTaskList[i]);

        return 0;
}
```

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.**

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

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. La informació específica de com és l'estat d'un thread en la nostra arquitectura i386 la trobareu a : struct i386_thread_state, i #defines i386_THREAD_STATE(flavor), i i386_THREAD_STATE_COUNT

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

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);
```

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**