Àlex Aguilera

Miguel Fernández

## Práctica Mach i GNU Hurd

**RESPUESTAS:**

**5. Un cop arreglat el problema de la pregunta anterior, comproveu que el programa memorymanagement.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. Responeu:**

**5.1. Quanta memòria assigna al procés cada crida a host\_processors?**

Cada llamada de host\_processors, asigna 0x1000.

**5.2. Quanta memòria assigna al procés cada crida a vm\_map?**

Cada llamada de vm\_map, asigna 0x2000.

**5.3. Quines adreces ens dóna el sistema en cada crida (host\_processors i vm\_map)?**

Para host\_processors: 0x1033000, 0x12e5000, 0x12e8000, 0x12eb000 y para vm\_map: 0x12e3000, 0x12e6000, 0x12e9000, 0x12ec000.

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

Se puede observar que son consecutivas y se van entrelazando, menos la primera que no sabemos porque lo hace diferente. Hemos aumentado a 32 iteraciones para verlo un poco mejor.

**5.5. Quines proteccions podem demanar a l’assignar memòria a un procés Mach? (pista: veieu el fitxer <mach/vm\_prot.h)**

Protecciones:

#define VM\_PROT\_NONE ((vm\_prot\_t) 0x00)

#define VM\_PROT\_READ ((vm\_prot\_t) 0x01) /\* read permission \*/

#define VM\_PROT\_WRITE ((vm\_prot\_t) 0x02) /\* write permission \*/

#define VM\_PROT\_EXECUTE ((vm\_prot\_t) 0x04)

#define VM\_PROT\_DEFAULT (VM\_PROT\_READ|VM\_PROT\_WRITE)

#define VM\_PROT\_ALL (VM\_PROT\_READ|VM\_PROT\_WRITE|VM\_PROT\_EXECUTE)

#define VM\_PROT\_NO\_CHANGE ((vm\_prot\_t) 0x08)

#define VM\_PROT\_COPY ((vm\_prot\_t) 0x10)

**5.6. 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?**

Nos sale un Segmentation Fault. Para ponerlo como solo lectura, hemos

modificado la función vm\_map, poniendo "VM\_PROT\_READ".

**5.7. 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.**

Para solucionarlo con la función vm\_protect(), hemos añadido lo siguiente al

código:

res = vm\_protect (mach\_task\_self(), (vm\_address\_t) p, 8192, 0, VM\_PROT\_DEFAULT);

**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).**

Código en ex8.c. No aparece ningún error de compilación, pero al ejecutar nos sale un Segmentation Fault. Creemos que se debe a que no el hijo no tiene permisos para ejecutar el printf.

**9. Observar que en el fitxer tenim dues definicions de funcions interessants per resoldre el problema de la pregunta anterior: … però cap de les dues soluciona el problema…**

La función mach\_setup\_thread() le da al thread una pila y la configura para que se ejecute con el PC cuando se reanude.

Mientras que, mach\_setup\_tls() le da al thread una zona TLS (thread-local storage)

**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).**

Código en ex10.c.

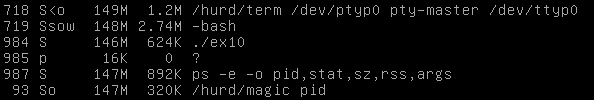


Imagen del ps mientras se ejecuta ex10

**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().**

Código en ex12.c. Se han creado dos procesos:. El hijo envia el proceso y el padre lo recibe. No se ha conseguido que se comuniquen y el proceso padre no llega a recibir el mensaje. Se ha probado hardcodeando el port en ambos pero no se ha tenido éxito.