**PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ**
**FACULTAD DE CIENCIAS E INGENIERÍA**

# SISTEMAS OPERATIVOS

**4ta práctica (tipo a)**
**(Primer semestre de 2014)**

Horario 0781: prof. V. Khlebnikov

Duración: 1 h. 50 min.
Nota: No se puede usar ningún material de consulta.
**La presentación, la ortografía y la gramática influirán en la calificación.**
Puntaje total: 20 puntos

---

**Pregunta 1 (4 puntos – 20 min.)** (*MOS4E, Chapter 3, Problem 21*) Below is an execution trace of a program fragment for a computer with 512-byte pages. The program is located at address 1020, and its stack pointer is at 8192 (to the last pushed value; the stack grows toward 0). Give the page reference string generated by this program. Each instruction occupies 4 bytes (1 word) including immediate constants. Both instruction and data references count in the reference string.

Load word 6144 into register 0
Push register 0 onto the stack
Call a procedure at 5120, stacking the return address
Subtract the immediate constant 16 from the stack pointer
Compare the actual parameter to the immediate constant 4
Jump if equal to 5152

**Pregunta 2 (2 puntos – 10 min.)** (*MOS3E, Chapter 3*) Si en el algoritmo *aging* los 2 contadores son 00100000 y 00101000, ¿qué página será eliminada? ¿Esta misma página sería eliminada en el algoritmo LRU? Y, si ambos contadores son 0, ¿cómo se comportarán el algoritmo *aging* y LRU?

**Pregunta 3 (2 puntos – 10 min.)** (*MOS3E, Chapter 3*) ¿Qué es *thrashing*? ¿Qué es *locality of reference*?

**Pregunta 4 (4 puntos – 20 min.)** (*MOS3E, Chapter 3, Problem 3*) In this problem you are to compare the storage needed to keep track of free memory using a bitmap versus using a linked list. The 2-GB memory is allocated in units of *n* bytes. For the linked list, assume that memory consists of an alternating sequence of segments and holes, each 4-MB. Also assume that each node in the linked list needs a 64-bit memory address, a 32-bit length, and a 32-bit next-node field. How many bytes of storage is required for each method? Which one is better?

**Pregunta 5 (8 puntos – 40 min.)** (*Computer Forensic Blog - Converting Virtual into Physical Addresses, by* Andreas Schuster) While analysing a memory dump, sooner or later you'll have to convert a virtual into a physical address. This can be a challenging task when it's done for the first time. This article will guide you through the process. Please have a look at the following excerpt taken from a debugger session:

```
kd> !process 0 0
**** NT ACTIVE PROCESS DUMP ****
PROCESS 829516a0  SessionId: 0  Cid: 0008    Peb: 00000000  ParentCid: 0000
    DirBase: 00030000  ObjectTable: 82976108  TableSize: 235.
    Image: System

PROCESS 8246c6a0  SessionId: 0  Cid: 00a4    Peb: 7ffdf000  ParentCid: 0008
    DirBase: 04e4b000  ObjectTable: 8246cd88  TableSize: 33.
    Image: SMSS.EXE

PROCESS 82442020  SessionId: 0  Cid: 00bc    Peb: 7ffdf000  ParentCid: 00a4
    DirBase: 07409000  ObjectTable: 824ec9a8  TableSize: 399.
    Image: CSRSS.EXE

PROCESS 824038c0  SessionId: 0  Cid: 00d0    Peb: 7ffdf000  ParentCid: 00a4
    DirBase: 07dce000  ObjectTable: 82404ae8  TableSize: 422.
    Image: WINLOGON.EXE
```

Did you notice that with the exception of the system process all processes seem to refer to the same Process Environment Block (Peb)? The Peb is a large structure which among other things refers to the program file loaded into memory, so

obviously this can't be true. The solution to this problem is strikingly simple - the Peb's address `0x7ffdf000` is a virtual address! But that doesn't help you much if all you've got is a memory dump. To read the Peb you'll have to convert the virtual address into a physical address and turn this into an offset into the dump file.

The following example will guide you through the process. Once again I will use the first image from the DFRWS Memory Analysis Challenge. First, it's free and publicly available. Second, the image was generated with dd. Therefore an offset into the image file equals an address in physical memory. Other file types would require some knowledge about the file internals and some mathematics to calculate back and forth between offsets and physical addresses.

Let's have a look at the Peb of UMGR32.EXE. As shown in the Intel manual, your starting point to convert virtual into physical addresses is the CPU's CR3 control register. Its value will be different for every process because each process has got a virtual address space of its own. The proper value for CR3 is saved in a variable named `DirectoryTableBase` of the `EPROCESS` structure. I've slightly modified Ptfinder to report this value (column title is "PDB").

```
> ptfinder.pl --nothreads dfrws2005-physical-memory1.dmp

No.  Type PID    TID    Time created        Offset     PDB        Remarks
---- ---- ------ ------ ------------------- ---------- ---------- ---------------
   4 Proc    668        2005-06-05 00:55:08 0x0095f020 0x075a7000 UMGR32.EXE
```

Let's find out the (virtual) address of the Peb. Therefore open the dump in a hex editor and seek to offset `0x0095f020`. From there proceed to the Peb's offset (`0x1b0` for Windows 2000). There you'll find the byte sequence `0x00 0xf0 0xfd 0x7f`. So the Peb's address is `0x7ffdf000`.
Write down this address as a binary number and get the most significant 10 digits. They'll give you an index into the Page Directory. In this example the index is `...` . (**Pregunta a)** (**2 puntos**) - **What is the index?**)

As seen above the Page Directory starts at physical address `0x075a7000`. Each Page Directory Entry (PDE) is 4 Bytes in size. So the address for entry no. ... is `...` . (**Pregunta b)** (**2 puntos**) - **What is the address?**)

The PDE reads `0x05cee067`. First thing to look at is the Page Size (PS) flag (bit 7). If set, the address belongs to a 4 MB page. Windows uses large pages only for the kernel's code. So not surprisingly our address belongs to a small (that is 4 KB) page. As the Present (P) flag (bit 0) indicates, this page is present in physical memory. So we can proceed examining the memory.

Because the Peb's address belongs to a small page the next 10 bits (21 to 12) are to be interpreted as an index into a Page Table. The remaining 12 bits are an offset into the memory page. (**Pregunta c)** (**2 puntos**) - **What are the Page Table index and the offset?**)

The Page Table starts at address `0x5cee * 0x1000`. Each Page Table Entry (PDE) is 4 bytes large. So we'll find entry no. ... at physical address `...` . (**Pregunta d)** (**1 punto**) - **What is the physical address?**)

It reads `0x0532f047` where the most significant 20 bits is the number of the physical memory frame. Now we can calculate a Frame Base Address and considering the offset in this frame obtain the physical address for the Peb's virtual address `0x7ffdf000`. (**Pregunta e)** (**1 punto**) - **What is the Peb's physical address?**)

La práctica ha sido preparada por VK
en Linux Mint 16 con LibreOffice Writer.

Profesores del curso: (0781) V. Khlebnikov

Pando, 10 de junio de 2014