PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

3ra práctica (tipo a)
(Segundo semestre de 2017)

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 (5 puntos – 25 min.) Considere el siguiente programa en C++:

```
$ cat -n 2017-2_pr3_v1.cpp | expand
     1  #include <bits/stdc++.h>
     2  using namespace std;
     3
     4  void algorithm(int blockSize[], int m, int processSize[], int n)
     5  {
     6      int allocation[n];
     7
     8      memset(allocation, -1, sizeof(allocation));
     9
    10      cout << "\nBlock No.\tBlock Size\n";
    11      for (int i = 0; i < m; i++)
    12          cout << "   " << i+1 << "\t\t" << blockSize[i] << endl;
    13
    14      for (int j=0; j<n; j++) {
    15          int i = -1;
    16          for (int k=0; k<m; k++)
    17              if (blockSize[k] >= processSize[j])
    18                  if (i == -1 || blockSize[i] > blockSize[k])
    19                      i = k;
    20          if (i != -1) {
    21              allocation[j] = i;
    22              blockSize[i] -= processSize[j];
    23          }
    24      }
    25      cout << "\nProcess No.\tProcess Size\tBlock no.\t...\n";
    26      for (int i = 0; i < n; i++) {
    27          cout << "   " << i+1 << "\t\t" << processSize[i] << "\t\t";
    28          if (allocation[i] != -1) {
    29              cout << allocation[i] + 1;
    30              cout << "\t\t" << blockSize[allocation[i]];
    31          }
    32          else
    33              cout << "Not Allocated";
    34          cout << endl;
    35      }
    36
    37      cout << "\nBlock No.\t... Block Size\n";
    38      for (int i = 0; i < m; i++)
    39          cout << "   " << i+1 << "\t\t" << blockSize[i] << endl;
    40  }
    41
    43  int main() {
    44      int blockSize1[] = {100, 500, 200, 300, 600};
    45      int processSize1[] = {321, 488, 143, 452};
    46      int m1 = sizeof(blockSize1)/sizeof(blockSize1[0]);
    47      int n1 = sizeof(processSize1)/sizeof(processSize1[0]);
    48
    49      algorithm(blockSize1, m1, processSize1, n1);
    50
    51      int blockSize2[] = {150, 350};
    52      int processSize2[] = {300, 25, 125, 50};
    53      int m2 = sizeof(blockSize2)/sizeof(blockSize2[0]);
    54      int n2 = sizeof(processSize2)/sizeof(processSize2[0]);
    55
    56      algorithm(blockSize2, m2, processSize2, n2);
    57
    58      return 0;
    59  }
```

```
$ g++ 2017-2_pr3_v1.cpp -o 2017-2_pr3_v1
$ ./2017-2_pr3_v1 | expand

Block No.       Block Size
   1            100
   2            500
   3            200
   4            300
   5            600

Process No.     Process Size    Block no.       ...
   1            321             ...             ...
   2            488             ...             ...
   3            143             ...             ...
   4            452             ...             ...

Block No.       ... Block Size
   1            ...
   2            ...
   3            ...
   4            ...
   5            ...

Block No.       Block Size
   1            150
   2            350

Process No.     Process Size    Block no.       ...
   1            300             ...             ...
   2            25              ...             ...
   3            125             ...             ...
   4            50              ...             ...

Block No.       ... Block Size
   1            ...
   2            ...
```

a) (3 puntos – 15 min.) Complete los resultados de ejecución del programa.

b) (2 puntos – 10 min.) Solamente para el 2do conjunto de datos (2 bloques para 4 procesos), entre los algoritmos *First-Fit*, *Next-Fit*, *Best-Fit*, *Worst-Fit*, ¿hay algunos que producen un mejor resultado? Si hay, ¿cuáles algoritmos y cuáles serán sus resultados?

Pregunta 2 (4 puntos – 20 min.) En la memoria de 4 MB se usa el *buddy system* con los bloques de orden-0 de 64 KB.

a) (2 puntos – 10 min.) ¿Cuáles son las direcciones hexadecimales de todos los bloques del orden 3?

b) (2 puntos – 10 min.) A la solicitud de 150 KB fue asignado el bloque que es un *buddy* al bloque en la dirección `0x2c0000`. ¿Cuál es su dirección?

Pregunta 3 (1 punto – 5 min.) In Fig. 3-3 the base and limit registers contain the same value, 16,384. Is this just an accident, or are they always the same? If this is just an accident, why are they the same in this example?

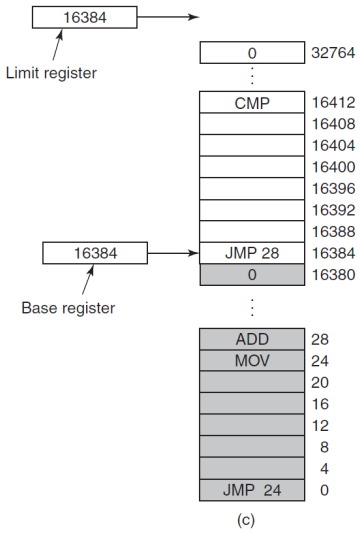

Figure 3-3. Base and limit registers can be used to give each process a separate address space.

Pregunta 4 **(1 punto – 5 min.)** For each of the following virtual addresses, compute the virtual page number and offset for a 4-KB page and for an 8-KB page: `0x4e20`, `0x8000`, `0xea60`.

Pregunta 5 **(2 puntos – 10 min.)** Using the page table of Fig. 3-9, give the physical address corresponding to each of the following virtual addresses:

(a) `0x44a8`
(b) `0x5bff`
(c) `0x9341`
(d) `0xb70d`

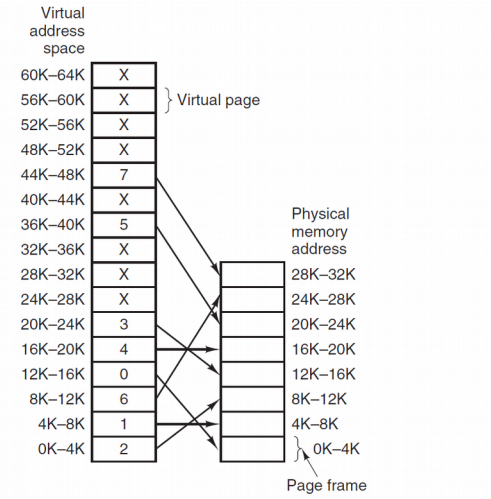

Figure 3-9. The relation between virtual addresses and physical memory addresses is given by the **page table**. Every page begins on a multiple of 4096 and ends 4095 addresses higher, so 4K–8K really means 4096–8191 and 8K to 12K means 8192–12287.

Pregunta 6 **(2 puntos – 10 min.)** A computer with a 36-bit address uses a two-level page table. Virtual addresses are split into a 7-bit top-level page table field, an 15-bit second-level page table field, and an offset. How large are the pages and how many are there in the address space?

Pregunta 7 **(5 puntos – 25 min.)** Below is an execution trace of a program fragment for a computer with 512-byte pages. The program is located at address `0x5fc`, and its stack pointer at `0x2200` (the stack grows toward 0 before every push operation).

a) (4 puntos – 20 min.) Give the page reference string generated by this program. Each instruction occupies 4 bytes (1 word) including immediate (instruction inside) constants. Both instruction and data references count in the reference string.

Load word `0x1a00` into register 0
Push register 0 onto the stack
Call a procedure at `0x1600`, stacking the return address
Subtract the immediate constant `0x10` from the stack pointer
Compare the actual parameter (in the stack) to the immediate constant 4
Jump if equal to `0x1620`

b) (1 punto – 5 min.) Si ninguna página de este programa está cargada en la memoria, ¿cuántos fallos de página sucederían para 3 marcos con el algoritmo óptimo?

La práctica ha sido preparada por VK
en Linux Mint 18.2 Sonya con LibreOffice Writer

Profesor del curso: (0781) V. Khlebnikov

Pando, 3 de noviembre de 2017