PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
3ra práctica (tipo a)
(Segundo semestre de 2021)

Horario 0781: prof. V. Khlebnikov

Duración: 1 h. 50 min.
Nota: **La presentación, la ortografía y la gramática influirán en la calificación.**
Puntaje total: 20 puntos

Pregunta 1 (6 puntos – 30 min.) Su respuesta debe estar en la carpeta **INF239_0781_P3_P1_Buzón** de la **Práctica 3** en PAIDEIA **antes de las 09:40**. Por cada 3 minutos de retardo son -2 puntos.
El nombre de su archivo debe ser <*su_código_de_8_dígitos*>`_31.txt`. Por ejemplo, `20171903_31.txt`.

El siguiente extracto de código es de `alloc.c`, parte del *process manager* (*PM task*) de Minix 2 y Minix 3, y es invocado dentro del PM cuando se libera memoria, por ejemplo, cuando algún proceso termina y el sistema ya puede recuperar el espacio de memoria anteriormente ocupado. Como se desprende de la descripción mínima de la función, la administración se hace con lista de "agujeros" o *hole list*, cuyos elementos tienen un campo `h_base` (la dirección inicial), otro campo `h_len` (el tamaño) y un campo `h_next` (el puntero al siguiente elemento de *hole list.* El primer elemento de *hole list* es apuntado por un puntero global `hole_head`. Otro puntero `free_slots` apunta a elementos nuevos libres para ser usados de ser necesarios.

La función recibe la dirección inicial (medida en *clicks*) y el tamaño (también en *clicks*) del bloque que se está liberando.

```
1   /*===========================================================================*
2    *                               free_mem                                    *
3    *===========================================================================*/

4   PUBLIC void free_mem(base, clicks)
5   phys_clicks base;      /* base address of block to free */
6   phys_clicks clicks;    /* number of clicks to free */
7   {
8       /* Return a block of free memory to the hole list.  The parameters tell where
9        * the block starts in physical memory and how big it is.  The block is added
10       * to the hole list.  If it is contiguous with an existing hole on either end,
11       * it is merged with the hole or holes.
12       */
13      register struct hole *hp, *new_ptr, *prev_ptr;
14
15      if (clicks == 0) return;
16      if ( (new_ptr = free_slots) == NIL_HOLE)
17          panic(__FILE__,"hole table full", NO_NUM);
18      new_ptr->h_base = base;
19      new_ptr->h_len = clicks;
20      free_slots = new_ptr->h_next;
21      hp = hole_head;
22
23      /* If this block's address is numerically less than the lowest hole currently
24       * available, or if no holes are currently available, put this hole on the
25       * front of the hole list.
26       */
27      if (hp == NIL_HOLE || base <= hp->h_base) {
28          /* Block to be freed goes on front of the hole list. */
29          new_ptr->h_next = hp;
30          hole_head = new_ptr;
31          merge(new_ptr);
32          return;
33      }
34
35      /* Block to be returned does not go on front of hole list. */
36      prev_ptr = NIL_HOLE;
37      while (hp != NIL_HOLE && base > hp->h_base) {
38          prev_ptr = hp;
39          hp = hp->h_next;
40      }
41
```

```
42      /* We found where it goes.  Insert block after 'prev_ptr'. */
43      new_ptr->h_next = prev_ptr->h_next;
44      prev_ptr->h_next = new_ptr;
45      merge(prev_ptr);        /* sequence is 'prev_ptr', 'new_ptr', 'hp' */
46  }
```

a) (1 punto) Linea 20: ¿por qué `free_slots` **toma el valor de** `new_ptr->h_next` **en el momento cuando los campos** `new_ptr->h_base` **y** `new_ptr->h_len` **recien se definen?**

b) (1 punto) Linea 36: ¿cuál es la importancia de esta asignación? Explique.

c) (4 puntos) Usando la notación

```
punt_A -> [B,C] -> [D,E] -> ... -> [Y,Z] -> NIL
                   ^
                   |
                   punt_B
```

presente todas las listas usadas para su ejemplo con los valores numéricos decimales de los campos en vez de las letras B, C, …, Z. Presente dos casos de funcionamiento de la función `merge()` **en la lista con varios nodos cuando la lista se modifica con la fusión de 2 y de 3 nodos. Indique a dónde apuntan los punteros usados antes de la fusión y la lista resultante.**

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 20.2 “Uma”

Profesor del curso: V. Khlebnikov

Lima, 12 de noviembre de 2021

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
3ra práctica (tipo a)
(Segundo semestre de 2021)

Horario 0781: prof. V. Khlebnikov

Duración: 1 h. 50 min.
Nota: **La presentación, la ortografía y la gramática influirán en la calificación.**
Puntaje total: 20 puntos

Pregunta 2 (6 puntos – 30 min.) Su respuesta debe estar en la carpeta **INF239_0781_P3_P2_Buzón** de la **Práctica 3** en PAIDEIA **antes de las 10:20**. Por cada 3 minutos de retardo son -2 puntos.
El nombre de su archivo debe ser <*su_código_de_8_dígitos*>`_32.txt`. Por ejemplo, `20171903_32.txt`.

Considere los siguientes datos que presenten el uso de la memoria (con páginas de 4 KB) durante la ejecución del mandato `cat`:

```
$ cat /proc/self/maps
address           perms  offset dev   inode      pathname
08048000-08051000 r-xp 00000000 08:07 1441834    /bin/cat
08051000-08052000 r--p 00008000 08:07 1441834    /bin/cat
08052000-08053000 rw-p 00009000 08:07 1441834    /bin/cat
086fe000-0871f000 rw-p 00000000 00:00 0          [heap]
b7329000-b7448000 r--p 002a3000 08:07 3804765    /usr/lib/locale/locale-archive
b7448000-b7648000 r--p 00000000 08:07 3804765    /usr/lib/locale/locale-archive
b7648000-b7649000 rw-p 00000000 00:00 0
b7649000-b77a3000 r-xp 00000000 08:07 5374093    /lib/i386-linux-gnu/libc-2.13.so
b77a3000-b77a4000 ---p 0015a000 08:07 5374093    /lib/i386-linux-gnu/libc-2.13.so
b77a4000-b77a6000 r--p 0015a000 08:07 5374093    /lib/i386-linux-gnu/libc-2.13.so
b77a6000-b77a7000 rw-p 0015c000 08:07 5374093    /lib/i386-linux-gnu/libc-2.13.so
b77a7000-b77aa000 rw-p 00000000 00:00 0
b77bc000-b77bd000 r--p 004ed000 08:07 3804765    /usr/lib/locale/locale-archive
b77bd000-b77bf000 rw-p 00000000 00:00 0
b77bf000-b77c0000 r-xp 00000000 00:00 0          [vdso]
b77c0000-b77dc000 r-xp 00000000 08:07 5376119    /lib/i386-linux-gnu/ld-2.13.so
b77dc000-b77dd000 r--p 0001b000 08:07 5376119    /lib/i386-linux-gnu/ld-2.13.so
b77dd000-b77de000 rw-p 0001c000 08:07 5376119    /lib/i386-linux-gnu/ld-2.13.so
bff18000-bff39000 rw-p 00000000 00:00 0          [stack]
```

a) (2 puntos) ¿Cuál es el tamaño de memoria que ocupa el programa, en páginas y en KB?

b) (1 punto) ¿Qué porcentaje (con resolución hasta las décimas) de la tabla de páginas de un nivel ocupan las entradas correspondiente al proceso? Para el cálculo considere los siguientes valores: 1K = 2^{10} = 1024, 1M = 2^{20} = 1048576, 1G = 2^{30} = 1073741824.

c) (3 puntos) Si la tabla de páginas es de 2 niveles, con el tamaño de tablas igual en cada nivel (4 KB), ¿qué entradas de estas tablas se usarían para el segmento de texto del programa *cat* (sin incluir *heap*) (2 puntos)? ¿Y para el segmento contiguo más grande (1 punto)? Para esta última pregunta, tome en cuenta la cantidad de entradas en cada tabla del nivel inferior, y por eso un segmento puede ocupar varias tablas del nivel inferior.

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 20.2 "Uma"

Profesor del curso: V. Khlebnikov

Lima, 12 de noviembre de 2021

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
3ra práctica (tipo a)
(Segundo semestre de 2021)

Horario 0781: prof. V. Khlebnikov

Duración: 1 h. 50 min.
Nota: **La presentación, la ortografía y la gramática influirán en la calificación.**
Puntaje total: 20 puntos

Pregunta 3 (8 puntos – 30 min.) Su respuesta debe estar en la carpeta **INF239_0781_P3_P3_Buzón** de la **Práctica 3** en PAIDEIA **antes de las 11:00**. Por cada 3 minutos de retardo son -2 puntos.
El nombre de su archivo debe ser <*su_código_de_8_dígitos*>`_33.txt`. Por ejemplo, `20171903_33.txt`.

a) (2 puntos) 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?

b) (2 puntos) It has been observed that the number of instructions executed between page faults is directly proportional to the number of page frames allocated to a program. If the available memory is doubled, the mean interval between page faults is also doubled. Suppose that a normal instruction takes 1 microsecond, but if a page fault occurs, it takes 2001 microseconds (i.e., 2 milliseconds to handle the fault). If a program takes 60 seconds to run, during which time it gets 15000 page faults, how long would it take to run if twice as much memory were available?

c) (2 puntos) Una pequeña computadora en una tarjeta inteligente tiene 4 marcos de página. En el primer tic de reloj los bits R son 0111 (página 0 es 0, el resto son 1). En los siguientes tic de reloj, los valores son 1010, 0011, 1100, 1000, 0001, 0010, 1101, 1010, 1011. Si se usa el algoritmo de envejecimiento (*aging*) con un contador de 8 bits, ¿cuáles serán los valores de cuatro contadores después del último tic?

d) (2 puntos) Write the binary translation of the logical address 0001010010111010 under the following hypothetical memory management scheme, and explain your answer: a paging system with a 256-address page size, using a page table in which the frame number happens to be four times smaller than the page number. How many bits are in a physical address?

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 20.2 “Uma”

Profesor del curso: V. Khlebnikov

Lima, 12 de noviembre de 2021