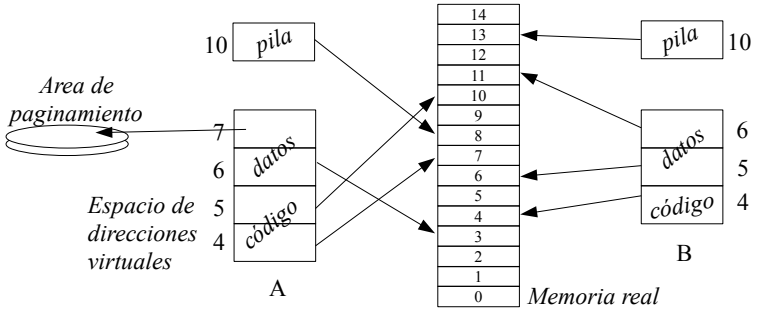
CC4302 Sistemas Operativos – Control 3 – Semestre Otoño 2017 – Prof.: Luis Mateu

Pregunta 1

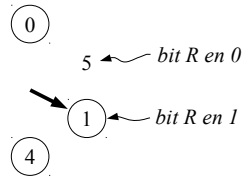
I. (3 puntos) Suponga que la MMU *no implementa* el bit D (*dirty*). Modifique la implementación de la estrategia del working set que aparece en el reverso de este enunciado de manera que no se grabe una página en disco si esa página ya había sido grabada previamente.

Ayuda: Use el bit W para emular el bit D y suponga que existe un bit de software W2 que es el que realmente indica si una página se puede escribir o no. Note que el hardware también gatilla un *page-fault* cuando se escribe en una página cuyo bit V es 1, pero su bit W es 0. Las zonas del código que Ud. debe modificar están marcadas con (!). No copie toda la implementación, coloque sólo las líneas que modificó más 2 líneas antes y 2 líneas después de la modificación.

II. (1,5 puntos) El diagrama de más abajo muestra la asignación de páginas en un sistema Unix que ejecuta los procesos A y B. Las páginas son de 4 KB. El núcleo utiliza la estrategia *copy-on-write* para implementar *fork*. (a) Construya la tabla de páginas del proceso A después de que este invoca *sbrk* pidiendo 6 KB adicionales. (b) Considere que el proceso B invocó *fork*. Construya la tabla de páginas para el proceso hijo justo después de que este modificó la página 5. No construya la tabla del padre. En las tablas indique página virtual, página real y atributos de validez y escritura.



III. (1,5 puntos) Considere un sistema Unix que implementa la estrategia del reloj. El sistema posee 6 páginas reales disponibles y corre un solo proceso. La figura indica el estado inicial de la memoria, mostrando las páginas residentes en memoria, la posición del cursor y el valor del bit R.



Dibuje los estados por los que pasa la memoria para la siguiente traza de accesos a páginas virtuales: 8 4 6 1 0 2

Pregunta 2

a.- (1,5 puntos) Considere un sistema operativo que implementa la estrategia del working set. Para el área de paginamiento se usa un solo disco que tiene una tasa de transferencia de 50 MB/segundo y un tiempo de acceso de 10 milisegundos.

Estime (i) el máximo número de *page-faults* por segundo que se puede atender cuando hay penuria de memoria; y (ii) cuanto tiempo tomaría llevar a disco (swap) un proceso cuyas páginas residentes en memoria totalizan 150 MB.

b.- (1,5 puntos) Se tiene un sistema operativo que ofrece solo procesos livianos. Explique si tiene o no sentido usar (i) la estrategia del reloj, o (ii) la estrategia del working set.

c.- (3 puntos) La siguiente es la parte relevante de la implementación del driver del dispositivo incógnito */dev/inc*:

```
static char inc_buf[80];
static ssize_t inc_count;
static struct semaphore lleno;
static struct semaphore vacio;

static ssize_t inc_read(
    struct file *filp, char *buf,
    size_t count, loff_t *f_pos) {
  if (*f_pos>0)
    return 0; // fin de archivo
  if (count>inc_count)
    count= inc_count;
  down(&lleno);
  copy_to_user(buf, inc_buf,
               count);
  *f_pos += count;
  up(&vacio);
  return count;
}
```

```
int inc_init(void) {
  …
  sema_init(&lleno, 0);
  sema_init(&vacio, 1);
  …
}

static ssize_t inc_write(
    struct file *filp, char *buf,
    size_t count, loff_t *f_pos) {
  if (count>80)
    count=80;
  down(&vacio);
  copy_from_user(inc_buf,
                 buf, count);
  inc_count= count;
  up(&lleno);
  return count;
}
```

La siguiente tabla es un ejemplo de uso incompleto. Complete la tabla con el mensaje que despliega cada comando e indique con el prompt $ el momento exacto en que termina el comando (si es que termina).

shell 1	***shell 2***	***shell 3***	***shell 4***
$ **echo abc > /dev/inc**			
	$ **echo def > /dev/inc**		
		$ **cat /dev/inc**	
			$ **cat /dev/inc**
$ **cat /dev/inc**			
			$ **echo ghi > /dev/inc**
		$ **cat /dev/inc** *<control-C>*	

CC4302 Sistemas Operativos – Control 3 – Semestre Otoño 2017 – Prof.: Luis Mateu

Implementación de la estrategia del working set para un núcleo clásico monocore:

```
// Se invoca para recalcular el working set
void computeWS(Process *p) {
  int *ptab= p->pageTable;
  for (int i= p->firstPage; i<p->lastPage; i++) {
    if (bitV(ptab[i]) {          // ¿Es válida?
      if (bitR(ptab[i])) {       // ¿Fue referenciada?
        setBitWS(&ptab[i], 1); // Sí, se coloca en el working set
        setBitR(&ptab[i], 0);
      }
      else {
        setBitWS(&ptab[i], 0); // No, se saca del working set
      }
    }
  }
}
// Se invoca cuando ocurre un pagefault,
// es decir bit V==0 o el acceso fue una escritura y bit W==0
void pagefault(int page) {
  Process *p= current_process; // propietario de la página
  int *ptab= p->pageTable;
  if (bitS(ptab[page]))        // ¿Está la página en disco? (!)
    pageIn(p, page, findRealPage()); // sí, leerla de disco
  else
    segfault(page);            // no
}
// Graba en disco la página page del proceso q
int pageOut(Process *q, int page) {
  int *qtab= q->pageTable;
  int realPage= getRealPage(qtab[page]);
  savePage(q, page);  // retoma otro proceso (!)
  setBitV(&qtab[page], 0);
  setBitS(&qtab[page], 1);
  return realPage;    // Retorna la página real en donde se ubicaba
}
// Recupera de disco la página page del proceso q colocándola en realPage
void pageIn(Process *p, int page, int realPage) {
  int *ptab= p->pageTable;
  setRealPage(&ptab[page], realPage);
  setBitV(&ptab[page], 1);
  loadPage(p, page);  // retoma otro proceso (!)
  setBitS(&ptab[page], 0);
  purgeTlb();         // invalida la TLB
  purgeL1();          // invalida cache L1
}
```

```
Iterator *it; // = processIterator();
Process *cursor_process= NULL;
int cursor_page;

int findRealPage() {
  // recorre las páginas residentes en memoria de todos los procesos buscando una
  // página que no pertenezca a ningún working set y que no haya sido referenciada
  int needsSwapping= 0; // Se necesita para no caer en ciclo infinito
  for (;;) {
    int realPage= getAvailableRealPage();
    if (realPage>=0)       // ¿Quedan páginas reales disponibles?
      return realPage;     // Sí, retornamos esa página
    // no, hay que hacer un reemplazo
    if (cursor_process==NULL) {// ¿Quedan páginas en proceso actual?
      // no
      if (!hasNext(it)) {    // ¿Quedan procesos por recorrer?
        // no
        if (needsSwapping) { // ¿Segunda vez que pasamos por aquí?
          // sí, revisamos todos los procesos sin encontrar reemplazo posible
          doSwapping();      // hacemos swapping
          needsSwapping= 0;
          continue;          // recomenzamos el ciclo for(;;)
        }
        resetIterator(it);   // partiremos con el primer proceso nuevamente
        // Si pasamos otra vez por acá haremos swapping
        needsSwapping= 1;
      }
      cursor_process= nextProcess(it); // pasamos al próximo
      cursor_page= cursor_process->firstPage; // primera pág.
    }
    // Estamos visitando la página cursor_page
    // del proceso cursor_process
    int *qtab= cursor_process->pageTable;
    // mientras queden páginas por revisar en cursor_process
    while (cursor_page<=cursor_process->lastPage) {
      if ( bitV(qtab[cursor_page]) &&   // ¿Es válida?
           !bitWS(qtab[cursor_page]) && // ¿no está en WS?
           !bitR(qtab[cursor_page]) ) { // no fue referenciada
        // sí, se reemplaza la página cursor_page de cursor_process
        return pageOut(cursor_process, cursor_page++);
      }
      cursor_page++;
    }
    // Se acabaron las páginas de cursor_process,
    // hay que buscar en el próximo proceso
    cursor_process= NULL;
  }
}
```