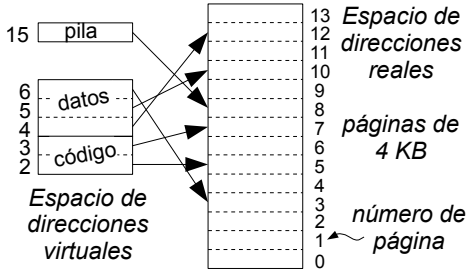
CC4302 Sistemas Operativos – Control 3 – Semestre Otoño 2019 – Prof.: Luis Mateu

Pregunta 1

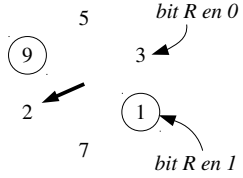
I. (2 puntos) El reverso de este enunciado corresponde a la implementación de la estrategia del working set. Reimplemente la misma estrategia considerando una MMU que sí implementa el bit D (*dirty*) de manera que no se grabe una página en disco si esa página ya había sido grabada previamente. No copie toda la implementación, coloque sólo las líneas que modificó más 2 líneas antes y 2 líneas después de la modificación.

II. (2 puntos) La figura de la izquierda muestra la asignación de páginas virtuales de un proceso a páginas reales de un sistema Unix que implementa *fork* usando la técnica *copy on write*.



Suponga que el proceso llama a *fork*. Indique el contenido de las tablas de páginas del proceso padre y del proceso hijo justo después que el hijo escribe en la página virtual 15 y el proceso padre escribe en la página virtual 4. Incluya en las tablas: número de página virtual, número de página real y atributos de validez y escritura (V y W).

III. (2 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: 9, 2, 6, 1, 2, 8.

Pregunta 2

a.- (1,5 puntos) Suponga que un proceso recorre 100 veces un arreglo de 4 MB secuencialmente en un procesador Intel x86 (con páginas de 4 KB). El arreglo se encuentra completamente residente en la memoria real. Estime el número de desaciertos en la TLB (*translation look-aside buffer*).

b.- (1,5 puntos) Considere un proceso cuyo código se ubica en el rango de direcciones [100 KB, 200 KB[, sus datos en [200 KB, 300 KB[y su pila en [3 GB - 12 KB, 3 GB[. Estime cuanta memoria se requiere en KB para almacenar sus tablas de páginas en un procesador intel x86 de 32 bits (páginas de 4 KB y tablas de 2 niveles).

c.- (3 puntos) La siguiente es la parte relevante de la implementación del driver para una memoria de 8 KB:

```
#define MAX 8192
static struct semaphore mutex;
static char mem_buf[MAX];
static int size;

static ssize_t mem_read(
    struct file *filp, char *buf,
    size_t count, loff_t *f_pos) {
  down(&mutex);
  if (count>size-*f_pos)
     count= size-*f_pos;
  copy_to_user(buf,
           mem_buf+*f_pos, count);
  *f_pos += count;
  up(&mutex);
  return count;
}
```

```
int mem_init(void) {
  …
  sema_init(&mutex, 1);
}

static ssize_t mem_write(
    struct file *filp, char *buf,
    size_t count, loff_t *f_pos) {
  down(&mutex);
  loff_t last= *f_pos+count;
  if (last>MAX) count -= last-MAX;
  copy_from_user(mem_buf+*f_pos,
                 buf, count);
  *f_pos += count;
  size= *f_pos;
  up(&mutex);
  return count;
}
```

Modifique este driver de manera que se comporte exactamente como el siguiente ejemplo de uso. El prompt $ indica el momento exacto en que termina el comando. Lo que ingresó el usuario está en **negritas**.

shell 1	*shell 2*	*shell 3*	*shell 4*
$ **echo hola > /dev/mem** *(espera)*			
	$ **echo que > /dev/mem**		
		$ **echo tal > /dev/mem**	
$			$ **cat /dev/mem** hola $
	$		$ **cat /dev/mem** que $
		$	$ **cat /dev/mem** tal $
			$ **cat /dev/mem** *(espera)*
$ **echo como estas > /dev/mem** $			como estas $

Restricción: Debe usar semáforos para la sincronización.

CC4302 Sistemas Operativos – Control 3 – Semestre Otoño 2019 – 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;
} }
```