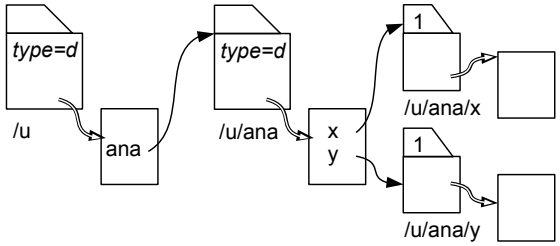
CC4302 Sistemas Operativos – Control 3 – Semestre Otoño 2018 – Prof.: Luis Mateu

Pregunta 1

I. (3 puntos) El reverso de este enunciado corresponde a la implementación de la estrategia del working set. Reimplemente la misma estrategia pero considerando una MMU que no implementa el bit R (*reference*). Ayuda: use astutamente el bit de validez (V) en su reemplazo, agregando otro bit (V2) que indique la validez efectiva de una página. No copie toda la implementación, coloque sólo las líneas que modificó más 2 líneas antes y 2 líneas después de la modificación.

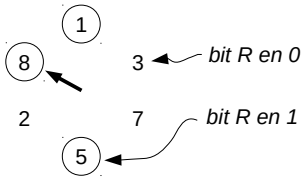
II. (1,5 puntos) La figura de la izquierda muestra varios archivos y directorios de la partición /u en un sistema Unix. A la derecha se muestran los comandos ejecutados por *ana*:




```
$ cd /u/ana
$ mkdir tmp
$ ln y tmp/w
$ cp tmp/w z
$ ln -s tmp/w v
```

Rehaga la figura de acuerdo a los cambios realizados por *ana*. No olvide indicar el contador de links para los archivos normales.

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: 5, 7, 4, 7, 2, 6.

Pregunta 2

a.- (1,5 puntos) Un computador posee un disco duro con un tiempo de acceso de 10 milisegundos y una velocidad de transferencia de 100 MB/seg. Estime cuanto tiempo tomaría mostrar todas las líneas que contengan el string "jperez" en (i) un archivo de texto desordenado de 10 MB, y (ii) 1000 archivos de texto desordenado de 10 KB cada uno, en el mismo directorio pero dispersos en el disco.

b.- (1,5 puntos) ¿Cuál de las 2 estrategias de reemplazo de páginas vistas en clases usaría Ud. en un sistema operativo mono-proceso para implementar paginamiento en demanda? Explique por qué descarta la otra estrategia.

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

```
static char inc_buf, ult= 0;
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 (ult=='\n') {
   ult= 0;
   return 0;
 }
 down(&lleno);
 copy_to_user(buf, &inc_buf, 1);
 copy_to_user(buf+1, &inc_buf, 1);
 ult= inc_buf;
 up(&vacio);
 return 2;
}
```

```
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) {
 for (size_t i= 0; i<count; i++) {
   down(&vacio);
   copy_from_user(&inc_buf, buf, 1);
   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***
$ **echo > /dev/inc** *(ver nota 1)*	
$ **echo abcd > /dev/inc** *(ver nota 2)*	
	$ **cat /dev/inc** *(ver nota 3)*
	$ **cat /dev/inc**
	$ **cat /dev/inc**
$ **echo ef > /dev/inc**	
$ **echo ghi > /dev/inc** ***<control-C>***	

Notas:

(1) El comando *echo* sin argumentos escribe solo un byte en su salida estándar: el newline (\n). En este caso su salida estándar se redirigió a */dev/inc*.

(2) Se escriben 5 bytes: a, b, c, d y el newline.

(3) El comando *cat* abre su único parámetro (*/dev/inc*) con *open* y lee repetidamente con *read* arrojando lo leído a la salida estándar (el terminal en este caso) hasta que *read* retorne 0. Luego invoca *close* y termina.

CC4302 Sistemas Operativos – Control 3 – Semestre Otoño 2018 – 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;
  }
}
```