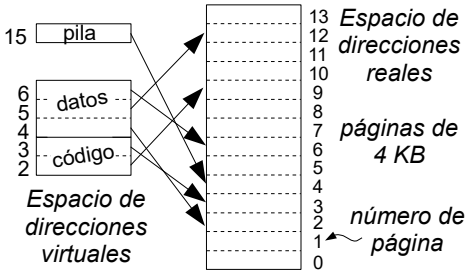
# CC4302 Sistemas Operativos – Control 3 – Semestre Primavera 2023 – Prof.: Luis Mateu

## Pregunta 1

***I.*** Considere un procesador con una MMU que *no implementa* el bit D (*dirty*). Modifique la implementación de la estrategia del reloj 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. No copie toda la implementación, escriba sólo las líneas que modificó más 2 líneas antes y 2 líneas después de la modificación.

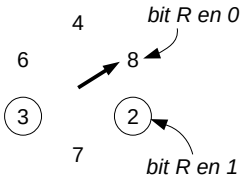
*Ayuda*: Use el bit W para emular el bit D y suponga que existe un bit de software W2 que es el que realmente indica si una página se puede escribir o no. Note que el hardware también gatilla un page-fault cuando se escribe en una página cuyo bit V es 1, pero su bit W es 0.

***II.*** 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 4 y el proceso padre escribe en la página virtual 6. Incluya en las tablas: número de página virtual, número de página real y atributos de validez y escritura (V y W).

***III.*** 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: 3, 8, 9, 3, 5, 2.

## Pregunta 2

***a.-*** (4 puntos) Un puente puede soportar PESOMAX como peso máximo. Los vehículos deben solicitar permiso para ingresar al puente invocando la función *entrar(p)* y notificar su salida llamando a la función *salir(p)*, en donde *p* es el peso del vehículo. Este peso nunca superará a PESOMAX y al salir siempre se indicará el mismo peso que anunció ese vehículo al entrar. La siguiente implementación usa mutex y condiciones para resolver el problema, pero no garantiza ingreso por orden de llegada.

```
pthread_mutex_t m= PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c= PTHREAD_COND_INITIALIZER;
double pesoActual= 0;
```

```
void entrar(double peso) {
  pthread_mutex_lock(&m);
  while (pesoActual+peso >
         PESOMAX)
    pthread_cond_wait(&c,&m);
  pesoActual += peso;
  pthread_mutex_unlock(&m);
}
```

```
void salir(double peso) {
  pthread_mutex_lock(&m);
  pesoActual -= peso;
  pthread_cond_broadcast(&c);
  pthread_mutex_unlock(&m);
}
```

Resuelva el mismo problema usando spinlocks para la sincronización. Además debe garantizar que los vehículos ingresen al puente por orden de llegada. Use una cola para los vehículos en espera. No olvide que *peek* permite obtener el primer elemento de la cola sin extraerlo.

***b.-*** (2 puntos) La siguiente es la parte relevante de la implementación del driver para el dispositivo incógnito */dev/inc*:

```
static struct semaphore s;
static int k= 0;
static ssize_t inc_read(
    struct file *filp, char *buf,
    size_t count, loff_t *f_pos) {
  if (*f_pos!=0)
    return 0;
  copy_to_user(buf,"1234567890",k);
  *f_pos += count;
  up(&s);
  return k;
}
```

```
int inc_init(void) {
  …
  sema_init(&s, 0);
}
static ssize_t inc_write(
    struct file *filp, char *buf,
    size_t count, loff_t *f_pos)
{
  k= count;
  down(&s);
  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 "abcd" > /dev/inc` | |
| | `$ cat < /dev/inc` |

Note que el comando *echo* también escribe un caracter \n (fin de línea).

## La estrategia del reloj para un núcleo clásico monocore

```
// 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 p 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 realPage= getAvailableRealPage();
  if (realPage>=0)      // ¿Quedan páginas reales disponibles?
    return realPage;    // Sí, retornamos esa página
  // no, hay que hacer un reemplazo
  for (;;) {
    if (cursor_process==NULL) { // ¿Quedan páginas en proceso actual?
      // no
      if (!hasNext(it))       // ¿Quedan procesos por recorrer?
        resetIterator(it);    // partiremos con el primer proceso nuevamente
      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?
        if (bitR(qtab[cursor_page]))   // no fue referenciada
          setBitR(&qtab[cursor_page], 0);
        else // 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;
  }
}
```