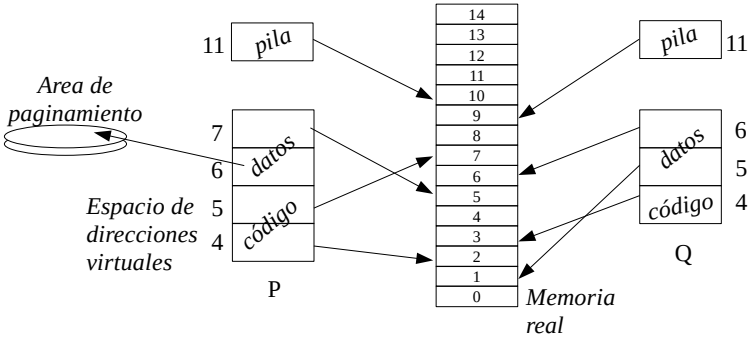
CC4302 Sistemas Operativos – Control 3 – Semestre Otoño 2025 – Profs.: Mateu, Torrealba, Arenas

Pregunta 1

I. 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. El diagrama de más abajo muestra la asignación de páginas en un sistema Unix que ejecuta los procesos P y Q. 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 P después de que este invoca sbrk pidiendo 10 KB adicionales. (b) Considere que el proceso Q invocó fork. Construya la tabla de páginas para el proceso hijo justo después de que este modificó la página virtual 11. No construya la tabla del padre. En las tablas indique página virtual, página real y atributos de validez y escritura.



Pregunta 2

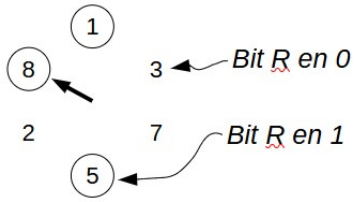
Implemente un semáforo usando como herramienta de sincronización solo spin-locks. El único tipo de *busy-waiting* permitido es el que hace *spinLock*. Los encabezados son los que se muestran en el cuadro de la derecha.

Ayuda: Complete la definición de la estructura *Sem*. Guarde en la cola *q* punteros a los spin-locks en donde esperan las llamadas pendientes de *sem_wait*.

```
typedef struct {
  Queue *q;
  ...
} Sem;
void sem_init(Sem *s,
       int iniFichas);
void sem_wait(Sem *s);
void sem_post(Sem *s);
```

Pregunta 3

A) (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 de abajo 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, 8, 2, 6.

B) (4 puntos) La columna de la izquierda del siguiente cuadro es la parte relevante de la implementación del driver para el dispositivo incógnito */dev/inc*:

```
int k= 0;
char *digs= "1234567890";
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,&digs[k%10],1);
  copy_to_user(buf+1,"\n",1);
  *f_pos += 2; k++;
  return 2;
}
```

```
$ cat < /dev/inc
…
$ cat < /dev/inc
…
$ cat < /dev/inc
…
```

La columna de la derecha 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.

Ayuda: Recuerde que si *read* retorna 0, el comando *cat* lo interpreta como fin de archivo y termina de inmediato.

CC4302 Sistemas Operativos – Control 3 – Semestre Otoño 2025 – Profs.: Mateu, Torrealba, Arenas

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;
} }
```