CC4302 Sistemas Operativos – Control 3 – Semestre Otoño 2026 – Prof.: L. Mateu & L. Torrealba

Pregunta 1

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 una condición 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);
}
```

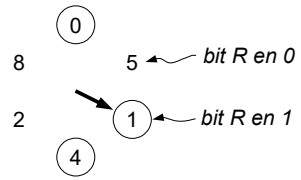
Resuelva el mismo problema usando spinlocks para la sincronización. Además debe garantizar que los vehículos ingresen al puente por orden de llegada. Use una cola para los vehículos en espera. No olvide que *peek* permite obtener el primer elemento de la cola sin extraerlo.

Pregunta 2

I. Modifique la implementación de la estrategia del *working set* que aparece en el reverso de este enunciado de manera que funcione correctamente con una MMU que *no implementa* el bit R (*reference*). No copie toda la implementación, escriba sólo las líneas que modificó más 2 líneas antes y 2 líneas después de la modificación.

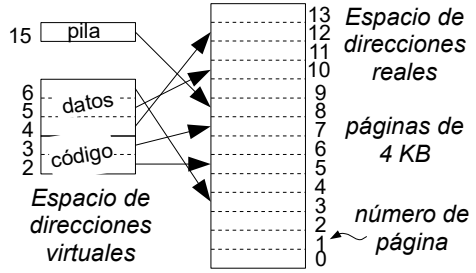
Ayuda: use astutamente el bit de validez (V) en su reemplazo, agregando otro bit VE que indique la validez efectiva de una página, con funciones *setBitVE* para establecer el valor de VE y *bitVE* para leer su valor.

II. 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

III. 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 5. Incluya en las tablas: número de página virtual, número de página real y atributos de validez y escritura (V y W).

Pregunta 3

Programe las funciones del recuadro para el módulo *voto*. Cada llamada a *voto_write* recibe un mensaje que puede ser el carácter 'A' o 'B', representando un voto por una de dos opciones.

```
int voto_init(void);
ssize_t voto_write(struct file *filp,
  char *buf, size_t count, loff_t *f_pos);
ssize_t voto_read(struct file *filp,
  char *buf, size_t count, loff_t *f_pos);
```

La función *voto_read* debe esperar hasta que se hayan **recibido 3 votos**. Luego debe entregar exactamente uno de los siguientes mensajes de **7 bytes**, según cuál opción tenga más votos al momento de la lectura: (i) Gana A, o (ii) Gana B. Una vez retornado el resultado, la votación debe reiniciarse y comenzar una nueva ronda de votación. La tabla de la derecha es un 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*
$ **cat < /dev/voto**		
	$ **echo A > /dev/voto** $	
		$ **echo B > /dev/Voto** $
	$ **echo A > /dev/voto** $	
Gana A $		

Para la sincronización debe usar los semáforos del núcleo de Linux. Por simplicidad no considere dataraces, manejo de errores, condiciones de borde ni señales con Control-C. El comando *read* siempre viene antes de los 3 *echo*. Lo importante es que funcione con ejemplos como el de este enunciado.

CC4302 Sistemas Operativos – Control 3 – Semestre Otoño 2026 – Prof.: L. Mateu & L. Torrealba

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;
} }
```