CC4302 Sistemas Operativos – Control 3 – Semestre Primavera 2024 – Profs.: Mateu, Torrealba, Arenas

Pregunta 1

```
void pedir(int id);
void devolver(int id);
void init( );
void filosofo(int id) {
  for (;;) {
    pedir(id);
    comer(id,(id+1)%5);
    devolver(id);
    pensar();
  }
}
```

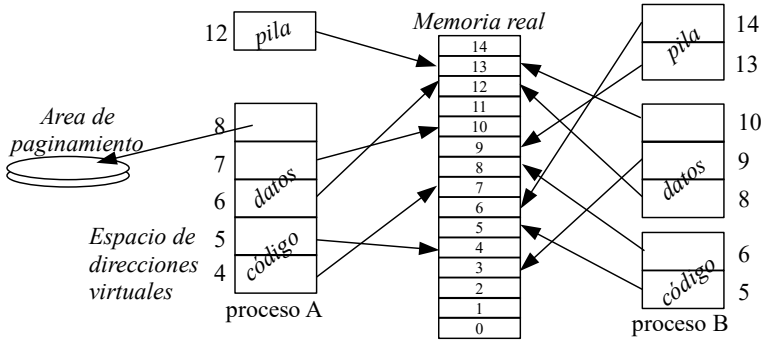
A la derecha se muestra una solución parcial de la cena de filósofos, en donde 5 filósofos numerados de 0 a 4 cenan juntos en una mesa redonda con 5 palitos, numerados de 0 a 4. Programe las funciones *pedir* y *devolver*. El parámetro *id* es el número del filósofo. Para evitar hambruna **la atención debe ser por orden de llegada**. Por ejemplo si el filósofo 1 está comiendo y 0 llama a *pedir*, 0 deberá esperar porque el palito 1 está ocupado. Si a continuación 3 llama a *pedir* también deberá esperar porque llegó después que 0. Finalmente, el filósofo 1 termina de comer e invoca *devolver*. Entonces 0 y 3 deberán comer en paralelo. Observe que la función *devolver* puede despertar 0, 1 o 2 filósofos.

Metodología obligatoria: La sincronización debe hacerse mediante spinlocks y el patrón request. Use una cola para los pedidos pendientes. Un pedido consiste en el número del filósofo que solicita comer y un spinlock en donde espera la autorización para comer.

Pregunta 2

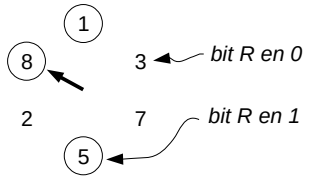
A) Modifique la implementación de la estrategia del reloj que aparece en el reverso de este enunciado de manera que no se grabe una página en disco si esa página ya había sido grabada previamente y no ha sido modificada. Aproveche que el procesador sí implementa el bit D (*dirty*). No copie toda la implementación, escriba sólo las líneas que modificó más 2 líneas antes y 2 líneas después de la modificación.

B) El diagrama de la siguiente columna muestra la asignación de páginas en un sistema Unix que ejecuta los procesos A y B.



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 A después de que este invoca *sbrk* pidiendo 5 KB adicionales. (b) A continuación el proceso B invocó *fork*. Construya la tabla de páginas para el proceso hijo justo después de que este modificó la página virtual 9. No construya la tabla del padre. En las tablas indique página virtual, página real y atributos de validez y escritura.

Pregunta 3



I) (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: 5, 7, 4, 8, 2, 6.

II) (4 puntos) El siguiente cuadro muestra a la izquierda el código que lee del dispositivo */dev/mem* visto en cátedra:

Implementación actual	***Ejemplo de uso***

`static ssize_t mem_read(`
`  struct file *filp, char *buf,`
`  size_t count, loff_t *f_pos) {`
`  down(&mutex);`
`  if (count > curr_size - *f_pos) {`
`  count= curr_size-*f_pos;`
`  }`
`  copy_to_user(buf,mem_buf+*f_pos,count);`
`  *f_pos+= count;`
`  up(&mutex);`
`  return count;`
`}`
	`$ echo tal > /dev/mem`
	`$ cat < /dev/mem`
	`l`
	`$ cat < /dev/mem`
	`a`
	`$ cat < /dev/mem`
	`t`
	`$`

Modifique la función *mem_read* de manera que cada vez que se abra el archivo en modo lectura, se obtenga un solo caracter y en orden inverso a como fueron escritos. El cuadro de arriba muestra a la derecha un ejemplo de uso. Por simplicidad ignore completamente la presencia de *newlines* (\n).

No necesitará modificar el resto de las funciones del driver. Observe que el comando *cat* primero abrirá el archivo. Luego usará *read* para leer muchos caracteres (*count* >1). Ud. debe entregarle a lo más un caracter. En seguida *cat* invocará *read* una 2^{da} vez para leer muchos caracteres nuevamente. Ud. deberá entregarle 0 caracteres para señalar el fin del archivo. Ud. puede darse cuenta que es la 2^{da} vez que *cat* invoca *read* porque **f_pos* será mayor que 0. Finalmente *cat* cerrará el archivo.

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 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, considerar el siguiente proceso
      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]))   // Sí fue referenciada
          setBitR(&qtab[cursor_page], 0);
        else // no, se reemplaza la pág. 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;
  }
}
```