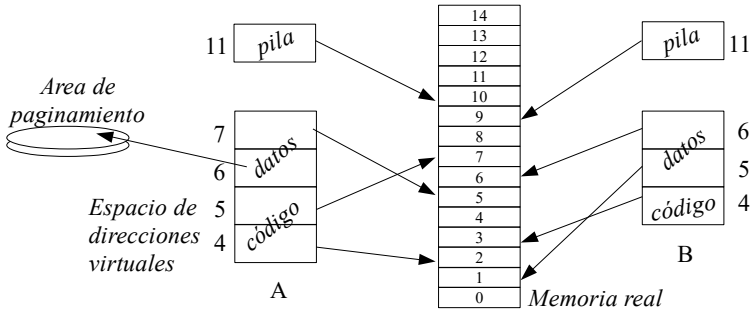
CC4302 Sistemas Operativos – Tarea 5 – Semestre Primavera 2020 – Prof.: Luis Mateu

Pregunta 1

A. Modifique la implementación de la estrategia del working set que aparece en la próxima página considerando un procesador con una MMU que *no implementa* el bit R (*reference*). Para ello revise la clase del jueves 3 de diciembre, en donde se implementa la estrategia del reloj y luego se modifica considerando el caso en que la MMU no implementa el bit R (se usa el bit de validez como bit de referencia). No copie toda la implementación, coloque sólo las líneas que modificó más 2 líneas antes y 2 líneas después de la modificación.

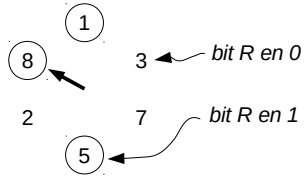
B. El diagrama de más abajo muestra la asignación de páginas en un sistema Unix que ejecuta los procesos A y B. Las páginas son de 4 KB. El núcleo utiliza la estrategia *copy-on-write* para implementar *fork*. (a) Construya la tabla de páginas del proceso A después de que este invoca *sbrk* pidiendo 10 KB adicionales. (b) Considere que el proceso B invocó *fork*. Construya la tabla de páginas para el proceso hijo justo después de que este modificó la página 11. No construya la tabla del padre. En las tablas indique página virtual, página real y atributos de validez y escritura.



Pregunta 2

I. Se argumenta que el paginamiento es lento porque por cada acceso a la memoria se debe hacer un acceso adicional en la tabla de páginas para traducir la dirección virtual a una dirección real. ¿Qué respondería Ud.? Explique.

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

Pregunta 3

A. Considere el siguiente diagrama de lecturas y escrituras (r, w) en memoria de un proceso Unix en un sistema que usa la estrategia del *working set*. Las filas corresponden a las páginas (0, 1, 2, …) y las columnas a los intervalos de cálculo del working set (A, B, C, …).

página

	A	B	C	D	E	F	G
6	r w	r	r	w r	r w w	r	w
5	r w r	r w		w r r		r w w	r
4	r						
3	r w w			r w			r
2	w r r			r			r
1	r r r	r r	r r		r r r	r	r
0	r r	r		r r r r	r r r r	r r r	r r

Tiempo

(i) Indique el valor del atributo *Referenced* para todas las páginas al inicio del intervalo E y al final de ese intervalo. (ii) Indique para los períodos C a F qué accesos pueden producir *page-faults*. Utilice coordenadas del estilo *(G, 4, 1er. acceso)*. (iii) Suponga que las páginas 2 y 3 son reemplazadas en C. En F es necesario reemplazarlas nuevamente. Explique si se deberán escribir nuevamente en disco considerando un sistema de paginamiento optimizado.

B. El sistema operativo de los smartphones no implementa paginamiento en demanda. Explique entonces cuál es la *principal* utilidad de los espacios de direcciones virtuales en un smartphone.

Entrega: Esta tarea no se resuelve en el computador. Responda en el formato de su preferencia: archivo ascii, pdf. con papel y lápiz (suba una foto legible a U-cursos). Si son varios archivos súbalos en formato .zip.

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;
}}
```