Pregunta 1

El siguiente código es una implementación incorrecta e ineficiente de un sistema de subastas para múltiples threads:

```
typedef enum {ADJUD, RECHAZ, SINRES} Resol;
typedef struct {
  PriQueue c; /* cola de prioridades */
  int n;      /* numero de unidades */
} *Subasta;
typedef struct {
  double oferta; /* dinero ofrecido */
  Resol resol;   /* estado de la oferta*/
} Req;
Subasta nuevaSubasta(int n) {
  Subasta s= malloc(sizeof(*s));
  s->n= n;
  s->c= MakePriQueue();
  return s;
}
Resol ofrecer(Subasta s, double oferta) {
  Req r= {oferta, SINRES};
  PriPut(s->c, &r, oferta);
  if (PriSize(s->c)>s->n) {
    Req *pr= PriGet(s->c);
    pr->resol= RECHAZ; /* se rechaza */
  }
  while (r.resol == SINRES)
    ;
  return r.resol;
}
double adjudicar(Subasta s, int *punid) {
  double recaud= 0.0;
  *punid= 0;
  while (!EmptyPriQueue(s->c)) {
    Req *pr= PriGet(s->c);
    pr->resol= ADJUD; /* se adjudica */
    (*punid)++; recaud += pr->oferta;
  }
  s->n -= *punid;
  return recaud;
}
```

Este código por sí solo debe ser suficiente para que Ud. comprenda qué es lo que debe hacer el sistema de subastas. Estúdielo detenidamente y descubrirá lo siguiente. La implementación funciona correctamente si las operaciones no se invocan simultáneamente. Un propietario crea su subasta invocando *nuevaSubasta*. En *n* indica el número de unidades idénticas del producto que subastará. Múltiples threads invocan concurrentemente *ofrecer* para hacer una oferta por una unidad de la subasta *s*. El dinero ofrecido se indica en *oferta*. La función *ofrecer* espera hasta que se resuelva si se adjudicó o no la unidad. Esto ocurre cuando se llega a los *n* threads que ya hicieron una oferta mayor (por simplicidad suponga que todas las ofertas son distintas) en cuyo caso se retorna *RECHAZ*; o cuando el propietario llama a *adjudicar* y en ese caso *ofrecer* retorna *ADJUD*. La función *adjudicar* retorna el total recaudado y entrega el número de unidades que se subastaron en **punid* (menor que *n* si no hay suficientes oferentes).

Observe que cuando llega un oferente se usa *PriPut* para encolar un objeto *r* de tipo *Req* en la cola *c* de la subasta. Su prioridad es la oferta. Si se llega a *n+1* oferentes, se extrae con *PriGet* la oferta menor y se rechaza.

A) (2 puntos) Corrija esta solución usando los monitores de nSystem, de acuerdo a los conocimientos que Ud. adquirió en el curso.

B) (3 puntos) Implemente la siguiente API de forma nativa en nSystem:

```
nSubasta nNuevaSubasta(int n);
Resol nOfrecer(nSubasta s, double oferta);
double nAdjudicar(nSubasta s, int *punid);
```

Debe reimplementar la misma funcionalidad de la parte A usando las funciones de bajo nivel de nSystem (START_CRITICAL, Resume, PutTask, etc.). Sea eficiente: evite cambios de contexto innecesarios.

C) (1 punto) Explique cuantos spin-locks se deben usar en el núcleo del sistema operativo para cada uno de los siguientes 4 tipos de núcleo: núcleo clásico para monocore, núcleo clásico para multicores, núcleo moderno para monocore y núcleo moderno para multicores.

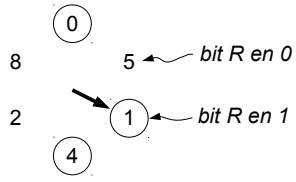
Pregunta 2

I. (3 puntos) Considere una máquina con 8 cores físicos sin un núcleo de sistema operativo, y por lo tanto no hay scheduler de procesos. Los 8 cores ejecutan código en paralelo y comparten la memoria. Los spin-locks son la única herramienta disponible para sincronizar los distintos cores. Programe la misma API de la parte A de la pregunta 1 para esta máquina. Observe que no hay threads, los oferentes son los distintos cores. Dado que no hay una cola de procesos ready, para esperar no le queda otra que hacerlo mediante un spin-lock. Ud. sí dispone de *malloc*, el tipo *PriQueue* y sus operaciones.

II. (1 punto) Se acaba de leer el bloque 500 de un disco tradicional. Los procesos P1, P2, P3, P4 y P5 se encuentran en espera con peticiones para leer los bloques 1000, 100, 600, 400, 900 respectivamente. Las peticiones se hicieron en ese orden. ¿En qué orden atendería las peticiones y por qué? ¿Y si el disco fuese un Solid State Drive (SSD)?

III. (1 punto) Suponga que en el sistema de archivos de Linux se usó *memcopy* en vez de *copy_from_user* para implementar la operación de escritura (*write*). Explique (a) cómo aprovecharía esto para leer datos del núcleo de Linux sin ser el administrador (root), y (b) por qué esta brecha no es suficiente para poder alterar el código del núcleo.

IV. (1 punto) Considere un sistema Unix que implementa la estrategia del reloj. El sistema posee 6 páginas reales disponibles y corre un solo proceso. La siguiente 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.