

CC4302 Sistemas Operativos – Examen – Semestre Otoño 2026

Profs.: Mateu, Torrealba

Pregunta 1

El siguiente código es una implementación incorrecta e ineficiente de un sistema de subastas multi-threaded.

```
typedef enum {PEND, RECHAZ, ADJUD} Resol;
typedef struct {
    int n;           // número de unidades subastadas
    PriQueue *q;     // cola de prioridades
} Subasta;

Subasta *nuevaSubasta(int n) {
    Subasta *s= malloc(sizeof(Subasta));
    s->n= n;
    s->q= makePriQueue();
    return s;
}

int ofrecer(Subasta *s, double oferta) {
    Resol r= PEND; // Agregar dirección de r a la cola
    priPut(s->q, &r, oferta); // oferta=prioridad
    if (priLength(s->q)>s->n) {
        // Se extrae la mejor oferta, que resulta ser la menor
        Resol *pr= priGet(s->q);
        *pr= RECHAZ; // se rechaza la menor oferta
    }
    while (r == PEND)
        ; // esperar hasta el rechazo o la adjudicación
    return r == ADJUD; // 1 si producto fue adjudicado
}

double adjudicar(Subasta *s, int *punid) {
    double recaud= 0.0; // dinero recaudado
    *punid= 0; // número de unidades adjudicadas
    while (!emptyPriQueue(s->q)) {
        double oferta= priBest(s->q);
        recaud += oferta;
        Resol *pr= priGet(s->q);
        *pr= ADJUD; // se adjudica a un thread oferente
        (*punid)++; // podrían ser menos que n
    }
    return recaud; // retorna el dinero recaudado
}
```

Este código por sí solo es suficiente para que Ud. comprenda qué es lo que debe hacer el sistema de subastas. Estúdielo detenidamente y descubrirá lo siguiente. La implementación funciona bien si las operaciones no se invocan simultáneamente. El propietario crea su subasta invocando *nuevaSubasta*, señalando en *n* el número de unidades idénticas del producto que subastará. Múltiples threads llaman a *ofrecer*, indicando su *oferta* por una unidad del producto. La subasta termina cuando el propietario invoca *adjudicar*.

Reprograme este código utilizando el *patrón request* para lograr una versión correcta y eficiente del sistema de subastas multi-threaded. Debe evitar cambios de contexto inútiles.

Ayuda: En la solución incorrecta, los elementos que

se agregan a la cola de prioridades son direcciones del entero *r*. En su solución necesitará agregar direcciones de estructuras.

Pregunta 2

Programa el mismo sistema de subastas de la pregunta 1 como herramienta de sincronización nativa de *nThreads*, es decir usando operaciones como *START_CRITICAL*, *setReady*, *suspend*, *schedule*, etc. Puede definir nuevos estados y agregar nuevos campos al descriptor de los nano threads. Debe implementar el tipo *nSubasta* y las siguientes funciones:

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

No puede implementar esta API en términos de otras herramientas pre-existentes en *nThreads* como semáforos, mutex o condiciones.

Ayuda: Se recomienda agregar la dirección del descriptor de los nano threads a la cola de prioridades. Puede serle útil el campo *ptr* de tipo *void ** declarado en el descriptor de los nano threads para almacenar la dirección de *r*.

Pregunta 3

A)Cuál es la función del bit D (*dirty bit*) en una entrada de la tabla de páginas? ¿Qué optimización permite realizar en el sistema operativo?

B) ¿Cuál es la diferencia entre *swapping* y paginación por demanda? ¿Cuál es la principal desventaja de utilizar únicamente *swapping*?

C) Se tiene un archivo de 98 KB en una partición Unix con bloques de 4 KB ($98/4=24.5$). Haga un diagrama mostrando inodo, bloques de datos y de indirección.

D) 5 procesos se encuentra en estado de espera porque hicieron peticiones para leer sectores del disco en el siguiente orden: 100, 900, 200, 400, 300. El último sector leído fue el 700 y el penúltimo el 800. Indique en qué orden se harán las lecturas de estos 5 procesos cuando la estrategia de scheduling de disco es: (i) *first come first served*, (ii) *shortest seek first*, (iii) método del ascensor (o *look*).