UNIVERSIDAD TÉCNICA FEDERICO SANTA MÁRIA
DEPARTAMENTO DE INFORMÁTICA
CAMPUS SSJJ

**Sistemas Operativos,**
**Pauta Certamen #2, 02/2019 – F2**
**Santiago, 16.03.2020**

**1.** [25% ~ 5% c/u] Conteste brevemente las siguientes preguntas:

**a)** ¿Cuáles son los dos grandes problemas del modelo Muchos a una? (implementación Hebras)
**Respuesta**: El proceso se bloquea completo y no existe paralelismo.

**b)** Mencione y describa brevemente los tipos de paralelismos.
**Respuesta**:
- Datos: Se ejecutan las mismas operaciones sobre distintos sets de datos.
- Operaciones: se separa la ejecución de operaciones.

**c)** ¿Qué es la conservación de trabajo?
**Respuesta**: La conservación de trabajo se produce cuando un algoritmo de planificación de uso de CPU no la deja nunca ociosa.

**d)** ¿En todos los algoritmos de planificación de uso de CPU el overhead producido por los cambios de contexto se puede despreciar? Fundamente.
**Respuesta**: No, en RR con q pequeño no es despreciable.

**e)** Indique las propiedades que debe cumplir toda solución a un problema de sincronización.
**Respuesta**: Seguridad y Vivacidad.

**2.** [25%] Una de las famosas fórmulas para calcular combinatorias corresponde a:

$$C_m^n = \frac{m!}{n!\,(m-n)!}$$

En donde m corresponde a los elementos disponibles y n a los elementos que tomamos por grupos. Utilizando la API simple thread y pseudocódigo construya un programa que permita calcular el valor C para cierto valor de m y n. Resuelva la concurrencia utilizando hebras! Algunas de las funciones útiles:

- void sthread_create(thread, func, arg)
- int sthread_join(thread)
- void sthread_exit(ret)

Respuesta: Una posible solución

```
#include <stdio.h>
#include “sthread.h”

static sthread_t thread;
static void factorial (int x);
int main (int argc, char**argv) {
```

```
        sthread_create(&thread, &factorial, m);
        int a = sthread_join(thread);

        sthread_create(&thread, &factorial, n);
        int b = sthread_join(thread);

        sthread_create(&thread, &factorial, m-n);
        int c = sthread_join(thread);

        printf("%f", (a/(b*c)));
        return 0;
}
void factorial (int x) {
        int result;
        if (x==0)
                result = 1;
        else
                sthread n1;
                sthread_create(&n1, &factorial, x-1);
                result=x*sthread_join(n1);

        sthread_exit(result);
 }
```

**3.** [25% ] El Banco SOntander gestiona los giros y abonos a las cuentas corrientes de sus clientes a través de hebras. Las operaciones, que se realizan de forma concurrente por las hebras, pueden abonar hasta un tope máximo de $200.000 en una cuenta y no pueden dejar una cuenta con menos de $10.000. Considere saldo inicial de $20.000

a) Si las operaciones retornan el monto del nuevo saldo si es exitoso y -1 si fallan, implemente la clase XThreads con el método correspondiente.
**Respuesta**: Una posible solución:

```
class XThreads {
private:
        int saldo;
        Lock lock;
public:
        int depositar(int dep);
        int girar(int giro);
        void XThreads();
}
```

```
int XThreads::depositar(dep){
    if (saldo+giro>200.000)
                return -1;
    else
        lock.aqcuire();
        saldo = saldo + dep;
        lock.release();
     return saldo;
```

```
void XThreads() {
        int saldo = 20.000
}
```

```
}
int XThreads::girar(giro){
   if (saldo-giro<10.000)
                return -1;
    else
        lock.aqcuire();
        saldo = saldo - giro;
        lock.release();
     return saldo;
}
```

b) Si en vez de generar errores las hebras se bloquean implemente nuevamente la clase XThreads.

**Respuesta**: Una posible solución:

```
class XThreads {
private:
        int saldo;
        Lock lock;
        CV cv1, cv2;
public:
        int depositar(int dep);
        int girar(int giro);
        void XThreads();
}

void XThreads() {
        int saldo = 20.000
}
```

```
int XThreads::depositar(dep){
    lock.aqcuire();
    while(saldo+dep>200.000){
        cv1.wait(&lock);
    }
    saldo = saldo + dep;
    cv2.signal();
    lock.release();
     return saldo;
}
int XThreads::girar(giro){
   lock.aqcuire();
    while(saldo-giro<10.000){
        cv2.wait(&lock);
    }
    saldo = saldo - giro;
    cv1.signal();
    lock.release();
     return saldo;
}
```

c) En el alcance de este problema, ¿existe diferencia entre usar broadcast o signal?

**Respuesta**: Si. Una hebra despertada podría no lograr realizar una operación por lo que se dormiría nuevamente. Utilizando broadcast() podríamos despertar a todas y lograr que la primera que pueda realizar una operación lo haga.

**4.** [25%] Considere un sistema operativo que virtualiza otros dos sistemas operativos y que gestiona la asignación de CPU a través de una cola de varios niveles utilizando 3 colas (Q1, Q2 y Q3, priorizadas en ese orden para SO Host, SO Guest1 y SO Guest2). Los algoritmos de itineración corresponden a RR=3, RR=5 y FCFS respectivamente. Considere la siguiente carga de trabajo:

| Process | CPU, I/O, CPU | Arrival Time |
|---|---|---|
| P0 | 5,6,7 | 0 |
| P1 | 4,2,3 | 3 |
| P2 | 2,3,4 | 4 |
| P3 | 5,2,7 | 7 |
| P4 | 3,2,4 | 14 |

Se pide construir la carta Gantt de planificación y calcular los tiempos de espera, respuesta y ejecución promedio. Nota: P0 y P2 son Guest1, P1 y P4 Guest2 y P3 Host.

**Respuesta**: Este tipo de implementación es apropiativa entre colas y las tareas se mantienen en las mismas.

Gantt:

| P0 | P2 | P3 | P3 | P2 | P3 | P0 | P2 | P0 | P1 | P4 | P1 | P4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 5 | 7 | 10 | 12 | 14 | 21 | 26 | 28 | 30 | 34 | 37 | 40 | 44 |

Cuadro resumen de tiempos:

| | T_ej | T_res | T_esp | E/S | CPU |
|---|---|---|---|---|---|
| P0 | 26 | 0 | 8 | 6 | 12 |
| P1 | 37 | 27 | 1 | 2 | 7 |
| P2 | 24 | 1 | 14 | 3 | 6 |
| P3 | 14 | 0 | 0 | 2 | 12 |
| P4 | 30 | 20 | 1 | 2 | 7 |
| | 26,2 | 9,6 | 4,8 | | |