UNIVERSIDAD TÉCNICA FEDERICO SANTA MÁRIA
DEPARTAMENTO DE INFORMÁTICA
CAMPUS SSJJ

Sistemas Operativos,
Pauta Certamen #2, 02/2020
Santiago, 05.12.2020

1. [25% ~ 5% c/u] Preguntas generales:

a) De todos los posibles cambios de contexto que se pueden generar en un SO mencione el que menos y el que más tiempo ocupan.
Respuesta:
- Menos tiempo: Producido por dos hebras del mismo proceso implementadas en espacio usuario.
- Más tiempo: Producido por dos hebras de distintos procesos implementados en espacio kernel.

b) Indique una situación en la que se ve afectado el criterio de progreso.
Respuesta: Cuando un proceso quiere entrar a la sección crítica pero no puede, a pesar de que nadie la está usando y nadie está esperando por entrar.

c) ¿Cuál es la (s) diferencia (s) entre un proceso de una hebra y uno de varias?
Respuesta: La diferencia se encuentra en la estructura. Un proceso de varias hebras debe definir una región específica para cada hebra, la cual contendrá el tid, PC, valores de registros y un stack propio.

d) Un SO de tiempo compartido es aquel que permite ejecutar varios procesos de distintos usuarios en un computador de forma concurrente, generando la sensación en los usuarios de que tienen todos los recursos asignados. ¿Cuál es la política de planificación de CPU más adecuada?
Respuesta: El más adecuado es round robin, ya que asigna a todos los procesos por igual un delta de tiempo de uso de CPU.

e) ¿Cuál es el peor escenario que puede enfrentar FCFS? ¿Por qué?
Respuesta: El peor escenario es cuando llegan todas las tareas que utilizan por más tiempo la CPU y luego las que la utilizan poco, generando el efecto convoy. Esto genera que el tiempo promedio de **respuesta** aumente considerablemente.

2. [25%] **Hebras**: Se tienen las matrices A y B de tamaños MxN y NxR respectivamente. Utilizando la librería **sthreads** construya un programa (con el mínimo número de instrucciones) que calcule la suma de todos los elementos de la matriz resultante luego de multiplcar AxB.

Nota: Se evaluará definición de variables, uso de llamadas al sistema de la API sthreads y pensamiento lógico utilizado para la resolución de este problema.

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{np} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

Respuesta: Una posible implementación:

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

int const filaA=M, columnaB=R, hebras=N;
static int A[filaA][hebras];
static int B[hebras][columnaB];
static int workers[hebras];
static int resultado=0;
static void suma (int u);

int main (int argc, char**argv) {
        int a;
        for (a=0; a<hebras; a++){
                sthread_create(&(workers[a]), &suma, a);
        }
        for (a=0; a<hebras; a++){
                resultado = resultado + sthread_join(workers[a]);
        }
        printf(“%d”, resultado);
        return 0;
}
void suma (int u) {
        int i, j, x=0;
        for (i=0; i<filaA;i++) {
                for (j=0; j<columnaB;j++) {
                        x = x + A[i][u]*B[u][j];
                }
        }
        sthread_exit(x);
 }
```

3. [25% ~ 15% y 10%] **Sincronización:**

a) Utilizando candados y variables de condición se pide implementar la clase xMom de tal forma que al compartir una de instancia de ella entre dos o más hebras siempre se muestre por pantalla la secuencia "MAMA..."

Respuesta: Una posible solución:

```
class XMom {
private:
        bool primeroM;
        Lock lock;
        CV cvM, cvA;
public:
        void writeM();
        void writeA();
        void XMom();
}
void XMom() {
        primeroM = True;
}
void  XMom::writeM(){
   int i=0;
   for(i=0;i<2;i++) {
      lock.aqcuire();
      while(!primeroM){
            cvM.wait(&lock);
      }
      printf("M");
      primeroM = False;
      cvA.signal();
      lock.release();
   }
   return;
}
void  XMom::writeA(){
   int i=0;
   for(i=0;i<2;i++) {
      lock.aqcuire();
      while(primeroM){
            cvA.wait(&lock);
      }
      printf("A");
      primeroM = True;
      cvM.signal();
      lock.release();

   }
   return;
}
```

b) Ahora se pide implementar lo mismo que el punto (a) pero utilizando semáforos. Para esto solo debe definir el/los semáforos y modificar los métodos correspondientes.

Respuesta: Una posible solución:

```
Semaphore sM = 1;
Semaphore sA = 0;

void  writeM(){
   int i=0;
   for(i=0;i<2;i++) {
      wait(sM);
      printf("M");
      signal(sA);
   }
   return;
}
```

```
void  writeA(){
   int i=0;
   for(i=0;i<2;i++) {
      wait(sA);
      printf("A");
      signal(sM);
   }
   return;
}
```

4. [25%~ 18%, 4% y 3%] Considere un SO que realiza la planificación de la CPU a través de una MLQ. Las tareas se clasifican en 3 tipos (A, B y C). Cada Tipo tiene 1 cola asignada (QA, QB y QC, priorizadas en este orden) y utilizan los algoritmos RR=3, prioridad apropiativa y RR=5 respectivamente. Todo empate en cualquier cola se resuelve por orden de llegada inicial.

Considere la siguiente carga de trabajo:

Tarea	Prioridad	Tipo	Llegada	CPU,ES,CPU
T0	-	C	0	4,5,3
T1	2	B	2	3,2,4
T2	-	A	4	2,7,3
T3	1	B	6	4,3,5
T4	-	A	8	1,4,5

Se pide:

- **a)** Construir la carta Gantt de planificación y calcular los tiempos de espera (TespR*), respuesta y ejecución promedio.
- **b)** Determinar, en caso de que ocurra, el % de tiempo en el cual la CPU quedó ociosa.
- **c)** Si ocurrió el punto (b) indicar un leve cambio para resolver el problema.

Respuesta: Gantt:

T0	T1	T2	T3	T4	T3	T1	T0	T2	T4	T4	T3	T1	T0	L	T0
0	2	4	6	8	9	11	12	13	16	19	21	26	30	31	36	39

Cuadro resumen de tiempos:

	T_ej	T_res	T_esp	E/S	CPU
T0	39	0	27	5	7
T1	28	0	19	2	7
T2	12	0	0	7	5
T3	20	0	8	3	9
T4	13	0	3	4	6
	22,4	0	11,4		

La CPU está ociosa (5/39)*100 = 12,82%

Dos posibles opciones: Reducir la primera ráfaga de T0 a 2 en vez de 4 o desplazar la llegada de T1 y los demás en 2 unidades de tiempo.