UNIVERSIDAD TÉCNICA FEDERICO SANTA MÁRIA
DEPARTAMENTO DE INFORMÁTICA
CAMPUS SSJJ

Sistemas Operativos,
Pauta Certamen #2, 01/2020
Santiago, 10.07.2020

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

a) ¿Cuáles son las técnicas con las cuáles el SO puede mantener todas las CPU's ocupadas? Menciónelas y explíquelas brevemente.
Respuesta: La idea es balancear la carga de tareas:
- Push: Constantemente buscar procesadores sobrecargados de tareas y migrar tareas a procesadores libres.
- Pull: Procesos disponibles pueden solicitar tareas a los ocupados.

b) ¿Qué es una condición de carrera?
Respuesta: Se produce cuando el comportamiento del programa depende de la forma en que se intercalan las operaciones que se deben ejecutar.

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 y explique las propiedades que debe cumplir toda solución a un problema de sincronización.
Respuesta: Se debe cumplir con:
- Exclusión mutua: Solo 1 tarea puede entrar a la sección crítica.
- Progreso: Las solicitudes de acceso deben avanzar.
- Espera acotada: Si se solicita entrar, la espera debe ser acotada.

2. [25%~ 15% y 10%] Problemas relacionados con deadloks:

a) Un SO tiene 3 recursos del tipo R1 y 3 recursos del tipo R2. El proceso P1 necesita 3 recursos del tipo R1 y 2 recursos del Tipo R2 y el proceso P2 necesita 2 recursos del tipo R1 y 3 recursos del tipo R2. Determinar (o dejar representada) una matriz de asignación, distinta de cero, que permita que el sistema quede en un estado seguro.

Respuesta: Considere la matriz Max = 3 2 / 2 3 y Allocation = u v / w x. Con esto la matriz Need = 3-u 2-v / 2-w 3-x. Bajo estas condiciones le vector disponible W nos queda =< 3-(u+w), 3-(v+x). Al correr el algortimo de seguridad y despejar las variables podemos notar que las únicas opciones para Allocation son z z / 00, z z / 0 1, 0 0 / z z, 1 0 / z z. En donde z es cualquier valor consistente con el número de recursos.

b) Un sistema tiene 4 procesos y 5 recursos para ser asignados. Considere la siguiente instantánea:

Task	Asignado	Máximo
A	10211	11212
B	20110	22210
C	11010	21310
D	11110	11221

Los recursos disponibles son: 0 0 **X** 1 1. Determinar el mínimo valor de X para que el estado sea seguro.

Respuesta: Utilizando el banquero se puede apreciar que bataría con X = 1.

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();
}

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

```
int XThreads::depositar(dep){
    if (saldo+giro>200.000)
                return -1;
    else
        lock.aqcuire();
        saldo = saldo + dep;
        lock.release();
     return saldo;
}
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 SO que realiza la planificación de la CPU a través de una versión modificada de la MFQ. Tiene 3 colas (Q1, Q2 y Q3, priorizadas en este orden) y utiliza los algoritmos SJF Apropiativo, RR=5 y FCFS respectivamente. En el caso de la cola que tiene SJF si una tarea es interrumpida (por apropiación) se queda en la misma cola. Solo avanza si logra finalizar el ciclo de CPU que se le asignó en la cola y no termina. El resto funciona igual. Considere la siguiente carga de trabajo:

Task	Llegada	CPU,ES,CPU
T0	0	5,2,7
T1	2	4,2,3
T2	3	2,6,4
T3	5	5,1,7
T4	10	3,8,4

Se pide construir la carta Gantt de planificación, calcular los tiempos de espera, respuesta y ejecución promedio y determinar el % de tiempo en el cual la CPU quedó ociosa.

Respuesta: Gantt:

T0	T2	T1	T4	T3	T0	T1	T2	T3	T4	T0	T3
0	5	7	11	14	19	24	27	31	36	40	42	44

Cuadro resumen de tiempos:

	T_ej	T_res	T_esp	E/S	CPU
T0	42	0	28	2	12
T1	25	5	11	2	7
T2	28	2	14	6	6
T3	39	9	17	1	12
T4	30	1	14	8	7
	32.8	3,4	16.8		