SISTEMAS OPERATIVOS

PRIMERA PRUEBA
DE
EVALUACIÓN A DISTANCIA
(PED1)

Curso 2016-2017

Nombre: Pablo Francisco
Apellidos: Guerra de Diego
DNI: 48.895.423-E
Centro: UNED de Cádiz

SISTEMAS OPERATIVOS
Primera Prueba de Evaluación a Distancia (PED1)

1. Explique **razonadamente** si las siguientes afirmaciones son verdaderas o falsas:

I) (1 p) El bloqueo de interrupciones es una solución simple a la exclusión mutua con apoyo del hardware que no presenta serios inconvenientes.
II) (1 p) El tiempo de conmutación de un cambio de proceso es independiente del hardware.
III) (1 p) El grado de multiprogramación viene definido por el número de procesadores existentes en el computador.
IV) (1 p) El algoritmo de planificación basado en prioridades solo puede implementarse como un algoritmo de tipo no expropiativo.

I) Afirmación **falsa**, puesto que aunque es cierto que es una solución simple a la exclusión mutua, no es el hardware quien apoya esta técnica, sino el mismo proceso ejecutando unas instrucciones especiales. Además sí que presenta serios inconvenientes:

1º- El rendimiento del sistema se puede degradar bastante al no permitirle atender las interrupciones más prioritarias en el momento que llegan.
2º- Como ya hemos mencionado antes, se deja en manos del proceso la decisión de ceder el procesador y si éste nunca lo devuelve el sistema se quedará colgado.
3º- No sirve para el caso de sistemas multiprocesadores, ya que aunque estén deshabilitadas las interrupciones, otro proceso en otro procesador podría intentar acceder a la misma región crítica que el proceso que ha solicitado el bloqueo.

II) Afirmación **falsa**, ya que este tiempo depende principalmente de factores asociados al hardware del computador como por ejemplo, la velocidad de lectura/escritura de la memoria principal, el número de registros del procesador cuyo contenido hay que salvar o cargar. La velocidad del procesador y la existencia de instrucciones especiales en el repertorio del procesador para salvar o cargar todos los registros.

III) Afirmación **falsa**, un sistema puede ser multiprogramado incluso teniendo sólo un procesador. El grado de multiprogramación viene definido por la cantidad de procesos que se pueden alojar en la memoria principal, el encargado de regular dicho grado en el sistema es el planificador a largo plazo.

IV) Afirmación **falsa**, puede ser tanto expropiativo como no expropiativo, si es expropiativo se dice que está guiado por eventos, y en su implementación no expropiativa un proceso solo puede acceder al procesador cuando el proceso que se está ejecutando finaliza o se bloquea.

2. Se dispone del siguiente conjunto de trabajos para su planificación:

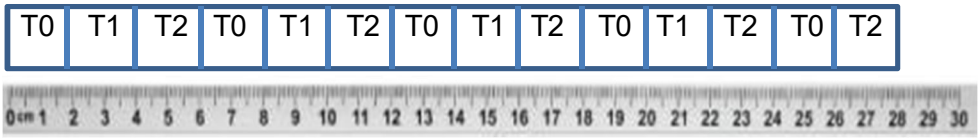
Trabajo	Instante de llegada (ut)	Duración de las ráfagas de CPU (ut)
T0	0	3, 2, 2, 2,1
T1	1	2, 3, 1, 2
T2	3	8, 1, 1

Si el sistema operativo utiliza un algoritmo de planificación de tipo round-robin con un quantum q=2 ut. Se pide:

a) (1 p) Dibujar el diagrama de uso de la CPU.
b) (1 p) Calcular los tiempos de espera y finalización de cada trabajo.

c) (1 p) Si se considera el tiempo de respuesta medio de un trabajo como el tiempo transcurrido en el sistema hasta completarse cada ráfaga del trabajo dividido por el número de ráfagas del trabajo, calcular los tiempos medios de respuesta para cada trabajo.

a)



b)

$T_{RT0} = T_{FT0} - T_{LLT0} = 26 - 0 = 26$ ut
$T_{ET0} = T_{RT0} - T_{ST0} = 26 - (3+2+2+2+1) = 26 - 10 =$ 16 ut
$T_{FT0} =$ 26 ut

$T_{RT1} = T_{FT1} - T_{LLT1} = 22 - 1 = 21$ ut
$T_{ET1} = T_{RT1} - T_{ST1} = 21 - (2+3+1+2) = 21 - 8 =$ 13 ut
$T_{FT1} =$ 22 ut

$T_{RT2} = T_{FT2} - T_{LLT2} = 28 - 3 = 25$ ut
$T_{ET2} = T_{RT2} - T_{ST2} = 25 - (8+1+1) = 25 - 10 =$ 15 ut
$T_{FT2} =$ 28 ut

c)

Tiempo de respuesta medio para la primera ráfaga de T0 que dura 3 ut:

$T_{RMTO} = 7/5 =$ 1,4 ut

Tiempo de respuesta medio para la segunda ráfaga de T0 que dura 2 ut:

$T_{RMTO} = 13/5 =$ 2,6 ut

Tiempo de respuesta medio para la tercera ráfaga de T0 que dura 2 ut:

$T_{RMTO} = 19/5 =$ 3,8 ut

Tiempo de respuesta medio para la cuarta ráfaga de T0 que dura 2 ut:

$T_{RMTO} = 25/5 =$ 5 ut

Tiempo de respuesta medio para la quinta ráfaga de T0 que dura 1 ut:

$T_{RMTO} = 26/5 =$ 5,2 ut

Tiempo de respuesta medio para la primera ráfaga de T1 que dura 2 ut:

$T_{RMT1} = 4/4 =$ 1 ut

Tiempo de respuesta medio para la segunda ráfaga de T1 que dura 3 ut:

T_{RMT1} = 15/4 = 3,75 ut

Tiempo de respuesta medio para la tercera ráfaga de T1 que dura 1 ut:

T_{RMT1} = 16/4 = 4 ut

Tiempo de respuesta medio para la cuarta ráfaga de T1 que dura 2 ut:

T_{RMT1} = 22/4 = 5,5 ut

Tiempo de respuesta medio para la primera ráfaga de T2 que dura 8 ut:

T_{RMT2} = 24/3 = 8 ut

Tiempo de respuesta medio para la segunda ráfaga de T2 que dura 1 ut:

T_{RMT2} = 27/3 = 9 ut

Tiempo de respuesta medio para la tercera ráfaga de T2 que dura 1 ut:

T_{RMT2} = 28/3 = 9,33 ut

3. (3 p) En una fábrica disponen de tres robots R1, R2 y R3 que producen piezas de tipo T1, T2 y T3, respectivamente. Los tres robots comparten una bandeja con capacidad para colocar dos piezas de tipo T1 y dos piezas de tipo T2. Los robots R1 y R2, cuando terminan de fabricar una pieza la depositan en la bandeja compartida si hay sitio en ella para colocarlas, en caso contrario detienen su actividad. Por su parte el robot R3 para fabricar una pieza de tipo T3 necesita dos piezas de tipo T1 y dos piezas de tipo T2 las cuales recoge de la bandeja compartida solo cuando están sobre ella las cuatro piezas que necesita. Los robots R1 y R2 reanudan su actividad cuando el robot R3 termina de recoger las cuatro piezas de la bandeja. Escribir el pseudocódigo de un programa en lenguaje C que usando semáforos binarios coordine la actividad de los robots. Dicho programa debe tener cinco partes: declaración de variables y semáforos, código del robot R1, código del robot R2, código del robot R3 y código para inicializar los semáforos y lanzar la ejecución concurrente de los robots.

Nota: Antes de escribir el pseudocódigo se debe explicar adecuadamente el significado de cada uno de los semáforos binarios y variables que se van a utilizar en el mismo.

- Semáforos

sR1: se encarga de regular las ejecuciones y bloqueos del proceso_R1(). Cuando dicho proceso pasa por primera vez por wait_sem(sR1) pone el estado de este semáforo a 0 y continúa su ejecución; la segunda vez que lo alcanza entra en la cola de procesos bloqueados del semáforo, de esta forma se garantiza que sólo recorra el bucle completo 2 veces, haciendo de esta forma exactamente 2 piezas T1.

sR2: se encarga de regular las ejecuciones y bloqueos del proceso_R2(). Cuando dicho proceso pasa por primera vez por wait_sem(sR2) pone el estado de este semáforo a 0 y continúa su ejecución; la segunda vez que lo alcanza entra en la cola de procesos bloqueados del semáforo, de esta forma se garantiza que sólo recorra el bucle completo 2 veces, haciendo de esta forma exactamente 2 piezas T2.

sR3: se encarga de regular las ejecuciones y bloqueos del proceso_R3(). Cuando dicho proceso se inicia y pasa por primera vez por wait_sem(sR3), comprueba que el semáforo está a 0 y

entra en la cola de procesos bloqueados del semáforo esperando la señal para ponerse a 1 y salir de ahí y entrar en la cola de procesos preparados. El segundo wait_sem(sR3),controlado por una condición, se encarga en iteraciones posteriores de bloquear o no el proceso, dependiendo de si los robots R1 y R2 se han adelantado o no en su labor a R3.

- TRUE

Mientras este valor sea cierto los bucles de los procesos se seguirán ejecutando.

- piezas

Esta variable entera es la encargada de llevar la cuenta de piezas acabadas por R1 y R2.

- estado

Esta variable entera perteneciente al struct semáforo es la encargada de guardar el estado del semáforo binario, por lo tanto sólo puede albergar los valores 0 o 1.

PSEUDOCÓDIGO DEL PROGRAMA

```
struct semáforo{
        int estado;
        /* Otras variables para implementar el semáforo y su cola
           (dependen de cada sistema operativo)*/
}
#define TRUE 1
semáforo sR1;
semáforo sR2;
semáforo sR3;
int piezas = 0;

void proceso_R1() {
        while(TRUE) {
                produce_T1();
                deposita_T1();
                piezas++;
                if(piezas == 4){
                        signal_sem(sR3);
                }
                wait_sem(sR1);
        }
}
void proceso_R2() {
        while(TRUE) {
                produce_T2();
                deposita_T2();
                piezas++;
                if(piezas == 4){
                        signal_sem(sR3);
                }
                wait_sem(sR2);
        }
}
void proceso_R3() {
        while(TRUE) {
                wait_sem(sR3);
                if(piezas != 4){
                        wait_sem(sR3);
                }
                recoge_piezas();
                piezas = 0;
                signal_sem(sR1);
                signal_sem(sR2);
                produce_pieza();
        }
}
void main() {
        init_sem(sR1, 1);
        init_sem(sR2, 1);
        init_sem(sR3, 0);
        ejecución_concurrente(proceso_R1, proceso_R2, proceso_R3);
}
```