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.

Falso. Es una solución simple con apoyo del hardware (exceptuando los sistemas multiprocesadores, en los que no funciona), pero sí presenta serios inconvenientes. Provoca que no se pueda atender ninguna otra interrupción, sin importar su prioridad, además un proceso egoísta puede tomar el control del procesador el tiempo que quiera.

II) (1 p) El tiempo de conmutación de un cambio de proceso es independiente del hardware.

Falso. Es dependiente prácticamente en su totalidad del hardware, puesto que pueden variar el número de registros del procesador, la velocidad del procesador y de la memoria principal, entre otros elementos del hardware, que influyen directamente en este tiempo.

III) (1 p) El grado de multiprogramación viene definido por el número de procesadores existentes en el computador.

Falso. El grado de multiprogramación viene definido por el número de programas cargados en memoria principal.

IV) (1 p) El algoritmo de planificación basado en prioridades solo puede implementarse como un algoritmo de tipo no expropiativo.

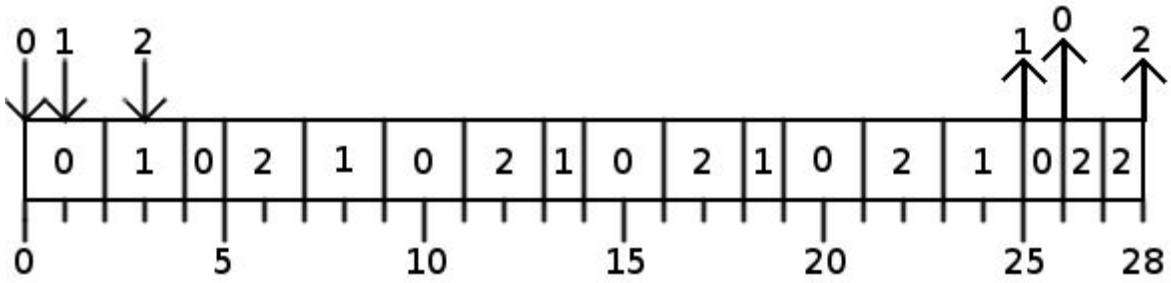
Falso. Puede implementarse tanto como algoritmo no expropiativo como expropiativo, en cuyo caso se dice que está guiado por eventos.

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.



(Para simplificar el diagrama he puesto 0,1 y 2 en lugar de T0, T1 y T2)

b) (1 p) Calcular los tiempos de espera y finalización de cada trabajo.

Calcularé el tiempo de espera de un trabajo como el tiempo transcurrido desde que llega hasta que termina, restándole el tiempo en el que está haciendo uso del procesador. Así:

T_E (T0) =16 ut **T_E (T1) = 16 ut** **T_E (T2) = 15 ut**
T_F (T0) =26 ut **T_F (T1) = 25 ut** **T_F (T2) = 28 ut**

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.

Tiempo de respuesta medio (T0) = 5.2
Tiempo de respuesta medio (T1) = 6.5
Tiempo de respuesta medio (T0) = 8.333…

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.

Al no indicar lo contrario, entiendo que los tres robots pueden acceder al mismo tiempo a la bandeja.

Puesto que en este caso concreto el espacio para cada tipo de pieza es independiente, y R3 coge las cuatro piezas a la vez, sumado a la necesidad de usar semáforos binarios, en mi solución tanto R1 como R2 harán una pieza, comprobará si está vacía su “zona” de la bandeja, depositarán dicha pieza, y al hacer la segunda pieza la depositarán sin comprobación alguna, puesto que son los únicos que actúan sobre su “zona” de la bandeja, junto con R3 que solo coge las piezas de dos en dos. Al depositar la segunda pieza será cuando avise a R3 de que ya están las dos piezas.

Por tanto usaré cuatro semáforos binarios, dos para cada tipo de pieza. Con el primero controlaré que la bandeja está vacía (vacia_T1 y vacia_T2), y lo inicializaré a 1. Este semáforo será al que esperen R1 y R2 para depositar la primera pieza, y con el que R3 les indicará que ya ha cogido las piezas. Con el segundo marcaré que la bandeja está llena para ese tipo de tipo de pieza (llena_T1 y llena_T2) y serán con los que R1 y R2 indiquen a R3 que ya están sus piezas, y al que espere R3 para recogerlas y fabricar la suya. Este se inicializará a 0.

```
/*Declaración de variables y semáforos*/
#define TRUE 1
semaforo binario vacia_T1, vacia_T2, llena_T1, llena_T2;
```

```
/*Código del robot R1*/
void r1()
{
	int t1;
	while (TRUE)
	{
		t1=fabricar_pieza_T1();
		wait_sem (vacia_T1);
		depositar_pieza_T1 (t1);
		t1=fabricar_pieza_T1();
		depositar_pieza_T1(t1);
		signal_sem (llena_T1);
	}
}

/*Código del robot R2*/
void r2()
{
	int t2;
	while (TRUE)
	{
		t2=fabricar_pieza_T2();
		wait_sem (vacia_T2);
		depositar_pieza_T2 (t2);
		t2=fabricar_pieza_T2();
		depositar_pieza_T2(t2);
		signal_sem (llena_T2);
	}
}

/*Código del robot R3*/
void r3()
{
	while (TRUE)
	{
		wait_sem (llena_T1);
		wait_sem (llena_T2);
		recoger_piezas();
		signal_sem (vacia_T1);
		signal_sem (vacia_T2);
		fabricar_pieza_T3 ();
	}
}
```

```
/*Código para inicializar los semáforos y lanzar la ejecución concurrente de los
robots*/

void main ()
{
	init_sem (vacia_T1,1);
	init_sem (vacia_T2,1);
	init_sem (llena_T1,0);
	init_sem (llena_T2,0);
	ejecucion_concurrente (r1, r2, r3);

}
```

Otra posible solución sería utilizar una variable de contador para cada tipo de pieza, de la siguiente forma:

```
#define N1 2

void r1_version2()
{
	int t1;
	int contadort1=0;
	while (TRUE)
	{
		t1=fabricar_pieza_T1();
		if (contador1==0)
		{
			wait_sem (vacia_T1);
		}
		depositar_pieza_T1 (t1);
		contador++;
		if (contador1==N1)
		{
			signal_sem (llena_T1);
			contador1=0;
		}
	}
}
```

Para el robot R2 sería análogo. Esta segunda solución es menos eficiente, pero más escalable.