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.

FALSA porque, aunque es cierto que es la solución más simple, sí que presenta inconvenientes serios: el sistema podría reducir su rendimiento al no poder atender a interrupciones más prioritarias cuando lleguen; y también presenta otro inconveniente a tener bastante en cuenta, y es que al bloquear la interrupción, el proceso que se ejecuta en ese momento tiene el poder absoluto para ceder el uso del procesador, y si no lo hace el sistema se quedará colgado.

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

FALSA, ya que -en general- para cambiar de proceso hay que realizar operaciones como: guardar el contexto del proceso en ejecución o terminado, ejecutar el algoritmo de planificación del procesador para seleccionar el proceso a ejecutar, y cargar el contexto del proceso B a ejecutar. Estas operaciones implican lectura y escritura de la memoria principal y los registros del procesador, por lo que el tiempo de conmutación dependerá de las velocidades de las operaciones descritas y ºdel número de registros del procesador, así como de la velocidad del procesador en ejecutar el algoritmo de planificación.

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

FALSA, el grado de multiprogramación es el número de programas cargados en la memoria principal, y además puede ser variable o constante.

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

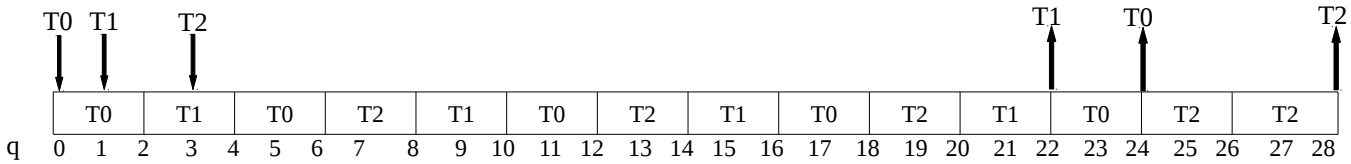
FALSA, puesto que puede implementarse como algoritmo tanto expropiativo como no expropiativo.
- **implementación expropiativa**: si actualmente se ejecuta un proceso A y llega un proceso B con una prioridad mayor que la del proceso A, se interrumpe el proceso A e inmediatamente se planifica el proceso B.
- **implementación no expropiativa**: se se ejecuta un proceso A y llega un proceso B con mayor prioridad que el A, se planifica el proceso B cuando el proceso A 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 la 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.

Proceso	T_E	T_F
T0	14	24
T1	15	22
T2	15	28

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.

$$T0 = \frac{24}{5} ut = 4{,}8\, ut$$

$$T1 = \frac{22}{4} ut = 5{,}5\, ut$$

$$T2 = \frac{28}{3} ut = 9{,}3\, ut$$

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.

Como variables he utilizado pzT1, pzT2 y pzT1T2, que son las que almacenan el número de piezas que lleva fabricadas (y por tanto están en la bandeja compartida) los robots R1, R2 y R3 respectivamente.

En el pseudocódigo aparecen tres semáforos denominados S1, S2 y S3 cada uno de los cuales sincroniza su proceso de fabricación con los otros dos procesos para satisfacer las condiciones que da el problema.

```
/* zona de código de declaración de variables y semáforos */

#define TRUE1 1      /* definición de la constante TRUE1 como 1 */
#define TRUE2 1      /* definición de la constante TRUE2 como 1 */
#define TRUE3 1      /* definición de la constante TRUE3 como 1 */
int pzT1;  /* declaración de la variable pzT1 del tipo integer o entero, y que almacena el número de piezas que el robot R1 tiene
          fabricadas actualmente en la bandeja */
int pzT2;  /* declaración de la variable pzT2 del tipo integer o entero, y que almacena el número de piezas que el robot R2 tiene
          fabricadas actualmente en la bandeja */
int pzT1T2;  /* declaración de la variable pzT1T2 del tipo integer o entero, y que almacena el número total de piezas que los robots
          R1 y R2 tienen fabricadas actualmente en la bandeja */
semaforo_binario S1, S2, S3;  /* semáforos que sincronizan los procesos de los robots R1, R2, y R3 respectivamente */

/*zona de código del robot R1 */
void procesoR1()
{
          while(TRUE1)
          {
                    pzT1=pzT1+1;    /* aumenta en una unidad el número de piezas fabricadas */
                    pzT1<2;          /* sección no crítica, número de piezas T1 menor de 2 */
                    signal_sem(S1);  /* si el valor del semáforo S1=1, sigue fabricando */
                    pzT1=2;          /* sección crítica, no puede seguir fabricando */
                    wait_sem(S1);    /* si el valor del semáforo S1=0, lo pone a la cola */
          }
}

/*zona de código del robot R2 */
void procesoR2()
{
          while(TRUE2)
          {
                    pzT2=pzT2+1;    /* aumenta en una unidad el número de piezas fabricadas */
                    pzT2<2;          /* sección no crítica, número de piezas T2 menor de 2 */
                    signal_sem(S2);  /* si el valor del semáforo S2=1, sigue fabricando */
                    pzT2=2;          /* sección crítica, no puede seguir fabricando */
                    wait_sem(S2);    /* si el valor del semáforo S2=0, lo pone a la cola */
          }
}
```

```
/*zona de código del robot R3 */
void procesoR3()
{
	while(TRUE3)
	{
		pzT1T2=pzT1T2+1;  /* aumenta en 1 el número total de piezas de la bandeja */
		pzT1T2!=4;	/* sección crítica, no puede fabricar piezas T2 */
		wait_sem(S3);	/* si el valor del semáforo S3=1, fabrica piezas T3*/
		pzT1T2==4;	/* sección no crítica, puede fabricar piezas T2 */
		signal_sem(S3);	/* si el valor de S3=1 sigue fabricando */
	}
}

/* código de inicialización de semáforos y ejecución  */

void main()
{
	init_sem(S1,1);	/* inicializa el semáforo S1 a 1 */
	init_sem(S2,1);	/* inicializa el semáforo S2 a 1 */
	init_sem(S3,0);	 /* inicializa el semáforo S3 a 0 */
	ejecución_concurrente(proceso_R1,proceso_R2,proceso_R3);	/* ejecución de los procesos */
}
```