Nombre: DNI:

Cumplimentar el cuestionario en la tabla anexa. (**30 % nota global**).
+0.2 respuesta correcta, -0.05 por error, 0 por repuesta no contestada.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
b	c	b	d	c	b	a	b	b	d	a	b	c	a	b

1) Para que un programa concurrente sea correcto, deben cumplirse las siguientes propiedades:
   a) Seguridad e inanición.
   b) Viveza y seguridad.
   c) Interbloqueo e inanición.
   d) Exlusión mútua y viveza.

2) La exclusión mutua mediante inhibición de interrupciones:
   a) Garantiza la ausencia de inanición.
   b) Mejora el rendimiento de las aplicaciones
   c) No puede utilizarse en sistemas multiprocesador.
   d) Únicamente garantiza la exclusión mutua en operaciones de E/S.

3) El algoritmo de exclusión mutua de Dekker:
   a) Está orienado a entornos distribuidos.
   b) Está orientado a entornos centralizados
   c) Es válido para n procesos con apenas modificaciones.
   d) Presenta situaciones en las que puede no garantizar las propiedades de viveza.

4) El problema de interbloqueo:
   a) Solo tiene solución si se resuelve mediante el uso de semáforos.
   b) Solo tiene solución si se resuelve mediante el uso de monitores.
   c) Solo tiene solución si se resuelve mediante el uso de algoritmos de espera ocupada.
   d) Todas las anteriores son falsas.

5) En términos de eficiencia:
   a) Los algoritmos de espera ocupada son más eficientes que los semáforos.
   b) Los monitores son más eficientes que los semáforos.
   c) A priori, no puede determinarse qué técnica de sincronización es la más eficiente.
   d) La eficiencia de los semáforos depende exclusivamente de la CPU.

6) La operación wait(s):
   a) Bloquea el proceso que la ejecuta si s=1.
   b) Bloquea el proceso que la ejecuta si s=0;
   c) Decrementa el valor de s y entonces bloquea el proceso si s=0.
   d) Si s=0 decrementa el valor de s y bloquea el proceso.

7) La gestión de los procesos bloqueados en un semáforo:
   a) Debe ser siempre FIFO para evitar la inanición.
   b) Puede ser FIFO o LIFO
   c) El Sistema Operativo desbloqueará los procesos en función de la prioridad.
   d) Mediante el uso de semáforos, los procesos no pasan a estado bloqueado.

8) Un semáforo “s” inicializado al valor 2
   a) Permite que dos procesos estén simultáneamente en su sección crítica.
   b) Dos procesos podrán ejecutar wait(s) sin bloquearse.
   c) Los semáforos se inicializan siempre a valor 1.
   d) El primer proceso que alcance la sentencia wait podrá acceder a su sección crítica.

9) Los monitores requieren de la utilización y definición de dos tipos de procesos:
   a) Procesos bloqueados y procesos bloqueantes
   b) Procesos activos y procesos pasivos.
   c) Procesos padre y procesos hijo.
   d) Proceso monitor y proceso principal.

10) En los monitores los procesos bloqueados:
   a) Se bloquean en las colas asociadas a variables de condición.
   b) Se bloquean en las colas de acceso al propio monitor
   c) Solo pasan a estado bloqueado si se ejecuta *delay*(var)
   d) a y b son ciertas

11) En la semántica resume & exit, el proceso desbloqueado por *resume(v)* es:
   a) El primer proceso que estuviera bloqueado en la cola de la variable de condición *v*
   b) El primer proceso que estuviera esperando para acceder al monitor.
   c) El sistema elige aleatoriamente entre la alternativa a y b
   d) Ninguna de las anteriores es cierta.

12) En el direccionamiento asimétrico del paso de mensajes:
   a) El emisor no identifica al receptor pero el receptor identifica al emisor
   b) El emisor identifica al receptor, pero el receptor no identifica al emisor
   c) El emisor identifica al receptor y el receptor identifica al emisor.
   d) El emisor no identifica al receptor y el receptor no identifica al emisor

13) En la instrucción de espera selectiva select, el proceso que la ejecuta se bloquea si:
   a) No se cumple ninguna de las guardas, si las tuviera.
   b) No disponga de alternativa else
   c) No existe ningún mensaje en los buzones/canales que se manejan
   d) La instrucción select no genera bloqueo del proceso.

14) El paso de mensajes síncrono permite la comunicación:
   a) Uno a uno.
   b) Uno a muchos
   c) Muchos a muchos
   d) Muchos a uno.

15) En las llamadas a procedimiento remoto (RPC), la invocación al resguardo del cliente:
   a) Siempre genera el bloqueo del proceso que realiza la invocación.
   b) Debe garantizar que existe concordancia entre los parámetros.
   c) No requiere de conexión entre cliente y servidor.
   d) La invocación se realiza siempre a un módulo que se encuentra en otro sistema.

---

1) (**1 punto**) Explicar el proceso de ejecución en los mecanismos de RPC.

2) (**1 punto**) Discutir la corrección o incorrección del siguiente algoritmo de exclusión mutua

```
process P0                          process P1
Repeat                              Repeat
        C0=quiereentrar;                    C1=quiereentrar;
        while turno ≠ 0 do                  while turno ≠ 1 do
                while C1=quiereentrar do;           while C0=quiereentrar do;
                turno=0;                            turno=1;
        Sección Critica0;                   Sección Critica1;
        C0=restoproceso;                    C1=restoproceso;
        Resto0;                             Resto1;
Forever                             Forever
```

---

3) (**1,5 puntos)** Supongamos que un proceso productor y *nCons* procesos consumidores comparten un buffer limitado de *n* elementos. El productor deposita elementos en el buffer mientras haya sitio y los consumidores los extraen. Cada mensaje depositado por el productor debe ser recibido por los *nCons* consumidores. Más aún, cada consumidor recibe los mensajes en el orden en que fueron depositados; sin embargo, diferentes consumidores pueden recibir los mensajes en tiempos diferentes. Resolver el problema usando semáforos.

4) (**2,5 puntos**) Una cuenta bancaria es compartida entre distintas personas. Cada titular puede operar sobre la cuenta mediante las siguientes operaciones:

- ingresar dinero,
- reintegrar dinero de la cuenta.
- consultar el saldo,

Las condiciones de contratación exigen que el saldo de la cuenta nunca puede ser negativo. Construir un sistema de comunicación asíncrona para gestionar el mantenimiento de la cuenta, contemplando que las operaciones de extracción de dinero deben quedar pendientes hasta que haya saldo disponible en la cuenta en los siguientes supuestos:

   a) Las extracciones se atenderán en el orden de llegada.
   b) Las extracciones se atenderán atendiendo al criterio de menor cuantía.

5) (**1 punto**) Un fichero de texto almacenado en una cinta infinita se ha codificado de forma que cada palabra se codifica mediante dígitos decimales, siendo el 0 el número usado para indicar la finalización de la palabra. Supongamos que tenemos un proceso lector, un proceso traductor, y un tercer proceso impresor. El proceso lector lee secuencias de números del fichero hasta encontrar un 0. El proceso traductor es el encargado de convertir las secuencias de números en cadenas de caracteres atendiendo a la codificación especial utilizada, la cual dependerá del número de caracteres (traducción tipo A si es par y tipo B si es impar). Finalmente el proceso impresor muestra el texto traducido por pantalla. Resolver el problema mediante el uso de monitores

2. Modelaremos tres procesos (coche, pasajero y controlador) y el monitor.

// Ver solución en libro de Palma, página 137

```
Program    Productor_Consumidor ;

type tipobuffer=record of
                    elem:tipoe1emento ;
                    ncon :integer;
          end;
var
          huecos ,mutex : semaphore;          // Semáforos para exclusión mutua
          vacios : array [1 .. ncons] of semaphore;  // Asegura que cada dato
                                                     // e lee por los nCons consumidores
          //Para que los consumidores se bloqueen a la espera de que se produzca
          buffer: array [l .. n] of tipobuffer;
          cola: array [l . . nCons] of integer; //Indica por donde va leyendo cada consumidor
          i:integer;

process productor;
var
          elem:tipoelemento ;
          i,frente :integer ;
begin
          repeat
                    producir (elem);
                    wait (huecos) ; //Esperar a que haya hueco
                    wait (mutex); // sección crítica
                    //Insertamos el elemento en el buffer
                    buffer[frente].elem=elem;;
                    buffer[frentel .ncon:=k;
                    frente: = (frente+1) mod n;
                    //Despertar a los consumidores que pudiera haber bloqueados en ese hueco
                    for i: =1 to nCons do signal (vacios [i]) ;
                    signal (mutex);
          forever
end ;

process type consumidor (id : integer) ;
var
          elem: tipoelemento;
begin
          repeat
                    wait (vacíos [id]);
                    wait (mutex) ;
                    //Leer el elemento del buffer
                    elem =buffer(cola[id]).elem ;
                    buffer[cola(id]].ncon=buffer[cola[id]].ncon-1;
                    // Si es el último en leer, desbloquea al productor
                    if buffer[cola[id]].ncon=0 then signal (huecos);
                    // Actualizar el siguiente element a consumir
                    cola[id]= (cola[id]+1) mod n;
                    signal (mutex) ;
                    consumir(elem);
          forever
end;

var consumidores: array [1 . . nCons] of consumidor;

begin
          initial (mutex) ;
          initial (huecos,n);
          for i=1 to nCons do initial (vacios[i],0);
          Cobegin
                    productor;
                    for i=1 to nCons do  consumidores[i];
          coend ;
end.
```

// se puede sustituir el array de semáforos por un array de contadores, para conocer cuántos consumidores han consumido un dato en concreto, forzando a que el productor no escriba en una posición concreta del buffer si su contador asociado es distinto de 0.

3. Modelaremos los procesos usuario y cajero

Opción A:las peticiones se atienden atendiendo al criterio FIFO

```
Process titular (int id) {
var
        real c;
        token: Boolean;
{
Repeat
        acción= generarAcción(); //Ingreso, extracción o petición saldo
        Switch acción :
                Ingreso: {
                        c=random();
                        Send (ingresar,c);
                }
                peticionSaldo: {
                        send(quieroSaldo[id],c);
                        receive(tomaSaldo[id],c);
                        printf (“El saldo es %d\n”,c);
                }
                extracción:{
                        c=random();
                        send(quieroSacar[id],c);
                        receive(puedoSacar[id],token);
                        // bloqueo hasta recepción de permiso
                        Send (extraer, c);
                }
Forever
}
```

```
Process cajero
Var
        Saldo: real;
        c:real;
        saldoBloqueado: boolean; //cantidad bloqueada para retirar
begin
        saldo=0;
        repeat
                select  // Siempre se aceptan órdenes de ingreso
                        receive (ingresar,c);
                        saldo+=c;
                or      // Siempre se aceptan órdenes de petición de saldo
                        for i=1 to N replicate
                        receive (quieroSaldo[id],c);
                        send(tomaSaldo[id],c);
                //Gestionar peticiones de extracción
                or      when !saldoBloqueado // en orden FIFO
                        for i=1 to N replicate
                        receive (quieroSacar[id],c)
                        saldoBloqueado=TRUE; //Nadie más puede sacar
                        send(puedoSacar[id],TRUE); //Informo al titular id que puede sacar
                or when saldoBloqueado //Solo podrá extraer uno
                        Receive (extraer,c);
                        Saldo=saldo-c;
                        saldoBloqueado=FALSE; // Para atender la siguiente petición
                endSelect;
        forever
end;
```

3. Modelaremos los procesos usuario y cajero

Opción B: as peticiones se atienden al criterio de menor cuantía

```
Process titular (int id) {
var
        real c;
{
Repeat
        acción= generarAcción(); //Ingreso, extracción o petición saldo
        Switch acción :
                Ingreso: {
                        … // Igual que anterior solución
                }
                peticionSaldo: {
                        … // Igual que anterior solución
                }
                extracción:{
                        c=random();
                        send(quieroSacar[id],c);
                        receive(puedoSacar[id],c);
                }
Forever
}
Process cajero;
Typedef registro struct {
        id: integer; //Proceso que quiere sacar
        c:real; // cantidad a reintegrar
Var
        Saldo,c: real;
        listPend: Tlist of registro
        // Lista ordenada en la que se borra por el principio y las
        // inserciones se harán de manera ordenada atendiendo al valor del campo c
begin
        saldo=0;
        listPend=new listPend(); // Creamos la lista
        repeat
                // se atienden las peticiones de reintegro que puedan atenderse
                if  !listPend.empty() {
                        While listPend.First().c<=saldo do {
                                Saldo=saldo-listPend.First().c; //Actualizar saldo
                                send(puedoSacar[listPend.First().id,c);
                                listPend.deleteFirst(); //Actualizar lista
                        }
                }
                select  // Siempre se aceptan órdenes de ingreso
                        ..// Igual que anterior solución
                or      // Siempre se aceptan órdenes de petición de saldo
                        ..// Igual que anterior solución
                or      // Peticiones de extracción
                        for i=1 to N replicate receive (quieroSacar[id],c);
                or terminate;
                endSelect;
        forever
        delete listPend;
end;
```

4.

```
monitor gestor;
Var
        F: FILE of Text;
        cadenaAImprimir, cadenaATraducir: string;
        esperaTraductor, esperaLector, esperaImpresor: condition;
export
        leerFichero (), traducir (), imprimir();
function leerFichero():integer {
        if cadenaATraducir="" then delay(esperarTraductor)
        repeat
                read (f,c);
                concatenar(cadenaATraducir,convertToString(c));
        until c=0;
        resume (esperaTraductor);
        return(cadenaATraducir.length());
}
```

```
void traducir() {
        if cadenaATraducir="" then delay (esperaTraductor);
        if cadenaATraducir.length() %2 =0 then
                cadenaAImprimir=traduccionTipoA(cadenaATraducir)
        else     cadenaAImprimir=traduccionTipoB(cadenaATraducir)
        cadenaATraducir="";
        resume(esperaLector);
        resume(esperaImpresor);
}
```

```
function imprimir():string {
var s:string
        if cadenaAImprimir="" then delay (esperaImpresor);
        s=cadenaAImprimir;
        resume (esperaTraductor);
        return (s);
}
```

```
begin
        cadenaAImprimir="";
        cadenaATraducir="";
        abrir(f);
end;
```

```
process lector;
var
        n:integer;
begin
        repeat
                n=gestor.leerFichero();
        forever
end;
```

```
process Traductor {
        repeat
                gestor.traducir();
        forever
}
```

```
Process impresor {
        Repeat
                Printf("el mensaje oculto es ",gestor.imprimir);
        Forever
}
```