NIA y Grupo: .....
Nombre y apellidos: .....

ACLARACIONES PRINCIPALES

1. **Aunque las pruebas evaluables pueden calificarse de forma acumulativa (puntuar por apartados), recordar que un error grave puede suponer un cero en su valoración.**

   Se revisará principalmente que todos los requisitos pedidos se cumplen (incluidas las condiciones de entrega), que la solución es completa, correcta, entregada en plazo y está descrita adecuadamente. Aspectos específicos a ser evaluados serían descritos en la presentación de la prueba.

2. **El objetivo general de todas las pruebas es evaluar los conocimientos, habilidades, etc. de la asignatura (y sus pre-requisitos) que el alumno/a debe de demostrar haber adquirido.**
3. **En caso de** detectarse una **copia** entre (copiados y copiadores), los **estudiantes involucrados serán suspendidos.** Dada la gravedad del caso, **además, supondrá la apertura de un procedimiento administrativo.**
4. **Se presenta un ejemplo (que no modelo) de prueba de evaluación de los primeros temas** que incluye dos ejercicios. La dificultad de los ejercicios, el contenido y el formato o puntuación podría variar totalmente.
5. **Para entrenar se aconseja usar todos y cada uno de los ejercicios disponibles en los cuadernos de ejercicios de cada tema. Para validar el entrenamiento se aconseja usar este ejemplo de examen parcial, una vez que se haya entrenado bien.** Evite entrenar con el conjunto de validación (ejercicios de examen) si quiere garantizar una adecuada preparación.
6. Los ejercicios se han de realizar correctamente y en el mínimo tiempo posible para garantizar que se puede contestar este ejemplo de examen parcial a tiempo. **Realizar todos los ejercicios del cuaderno de ejercicios en papel y bolígrafo ofrece la posibilidad de adquirir soltura necesaria para lograr el objetivo.**

NIA y Grupo: .....
Nombre y apellidos: .....

NOTAS:

- Para la realización del presente examen se dispondrá de **60 minutos**
- **No** se pueden utilizar libros **ni** apuntes, ni usar calculadora, móvil (o similar)
- Será necesario presentar el DNI o carnet universitario para realizar la entrega del examen
- Hay que contestar cada ejercicio en el espacio dado. Es obligatorio la entrega del examen.
- Hay que indicar el nombre, NIA y grupo al que pertenece en TODAS las hojas que entregue.

Ejercicio 1 (3.0 puntos)

Disponemos de un sistema con CPU monoprocesador y sistema operativo con un *kernel* monolítico expulsivo (como el usado en clase). En nuestra empresa se nos pide cambiar este sistema operativo para añadir una nueva llamada al sistema denominada Crear_proceso2 que permita al crear un proceso, indicar la rodaja de tiempo que tendrá durante toda su ejecución. Esta nueva llamada se unirá a la existente Crear_proceso que usará la rodaja por defecto.

Sin cambiar el planificador, se pide:

a) Indicar las estructuras de datos y funciones que se han de modificar para poder añadir una nueva llamada al sistema comentada anteriormente.
b) Indicar en pseudo-código las funciones del apartado anterior.

SOLUCIÓN

a) Las estructuras de datos a modificar son: BCP, añadiendo el campo rodajaCompleta
Las funciones a modificar o añadir son: la llamada Crear_proceso2 y la interrupción hw/sw de reloj.

b) El pseudocódigo de las funciones pedidas es:

Crear_Proceso2 (int rodaja)

- REG0 <- código de llamada
- REG1 <- rodaja
- Trap
- return

Si **contestación adecuadamente** entonces **0.75**

kernel_Crear_Proceso2 ()

- Realizar lo mismo que Crear_proceso() pero con "BCP_procesoCreado->rodajaCompleta = REG1" en lugar de "BCP_procesoCreado->TICKS_POR_RODAJA"

Si **contestación adecuadamente** entonces **0.75**

NIA y Grupo: .....
Nombre y apellidos: .....

Manejador_interrupción_hw_reloj ():

- Ticks++;
- InsertarInterrupciónSoftware(Manejador_interrupción_sw_reloj);
- ActivarInterrupciónSoftware();

Manejador_interrupción_sw_reloj ():

- procesoActual->rodaja -- ;
- if (0 == procesoActual->rodaja)
    - procesoAnterior = procesoActual
    - procesoAnterior->estado = LISTO
    - procesoAnterior->rodaja = **procesoAnterior->rodajaCompleta**
    - procesoActual = planificador()
    - procesoActual->estado = EJECUTANDO
    - cambio_contexto(procesoAnterior->context, procesoActual->context)

Si **contestación adecuadamente** entonces **1.5**

NIA y Grupo: .....
Nombre y apellidos: .....

Ejercicio 2 (3.0 puntos)

Una compañía nos contrata para hacer el driver de un disco duro. El disco duro permite escribir o leer bloques desde/hacia una zona de memoria (DISCO_DMA) un bloque de disco y al terminar de transferir todo el bloque genera una interrupción hardware para avisar.

Es la tercera versión del driver y nos piden implementar que se encolen las peticiones de forma ordenada de manera que cuando termine de servirse a un proceso una petición se seleccione la siguiente petición en la cola respecto a la servida en ese momento.

Si dos procesos realizan cada uno una petición al mismo bloque se consideran peticiones separadas. No obstante para varias peticiones consecutivas de lectura sobre el mismo bloque, los datos de la primera petición se envían al resto. Se pide minimizar el número de copias de memoria entre el sistema operativo el procesos solicitante.

Se pide:

a) Indicar la interfaz completa que tendrá el driver
b) Indicar las estructuras de datos que se necesiten
c) Implementar en pseudocódigo la funcionalidad, la interrupción hardware, interrupción software y la función de petición de bloque.

NOTA: considera para simplificar comprobaciones que el tamaño a leer será múltiplo del tamaño de bloque, y que existe una función de inserción ordenada en una lista.

NIA y Grupo: .....
Nombre y apellidos: .....

Si **contestación adecuadamente** entonces **0.75**

SOLUCIÓN

a) La interfaz se divide en, al menos, tres bloques:

- Gestionar la interrupción hardware/software
  - **Manejador_interrupcion_HW_disco ()**
  - **Manejador_interrupción_SW_disco ()**
  - **Función_Petición_Bloque ()**

- Gestionar las llamadas al sistema del driver:
  - **Desc = Open (nombre_dispositivo, flags)**
    - Permite iniciar una sesión de trabajo
  - **Res = Close (Desc)**
    - Libera la sesión de trabajo
  - **Res = Read (Desc, buffer, size)**
    - Permite leer los bytes indicados en size dentro del buffer
  - **Res = Write (Desc, buffer,size)**
    - Permite escribir los bytes indicados en size dentro del buffer.

- Gestionar la carga y descarga del driver en tiempo de ejecución:
  - **CargarDriver_disco()**
    - Carga el driver en el espacio de memoria del kernel
    - Inserta la estructura del driver (variables y operaciones) en la tabla de drivers.
  - **DescargarDriver_disco ()**
    - Elimina la estructura del driver de la tabla de drivers.
    - Borra el código del driver del espacio de memoria del kernel.

Si **contestación adecuadamente** entonces **0.5**

b) Las estructuras de datos requeridas son:

- Operaciones del driver: Incluye punteros a las funciones de interfaz
- **Disco_DMA:** zona de memoria donde el disco copia el bloque de datos pedido
- **PeticiónActual:** que es un puntero a la petición en curso.
- **listaPeticiones:** podrá ser NULL si no hay peticiones pendientes.
  Cada petición incluye
  - El **identificador del bloque** pedido
  - La **operación** sobre el bloque pedido,
  - La **dirección del buffer** en la que almacenar los datos del dispositivo y
  - El puntero al **proceso bloqueado** en la petición.

NIA y Grupo: .....
Nombre y apellidos: .....

c) Los eventos involucrados son:
- Interrupción hardware/software
- Función de petición de bloque
- Llamada al sistema read

Si **contestación adecuadamente** entonces **1**

Manejador_Interrupción_HW_disco()

- **Si (peticiónActual->operación==READ)**
  - **Copiar de Disco_DMA a peticionActual->Buffer**
- Insertar_Interrupción_Software(Manejador_Interrupción_SW_disco)
- Generar_Interrupcion_Software();

Manejador_Interrupción_SW_disco()

- **peticionActual->bloqueado->estado = LISTO**
- **Insertar (ListaListos, peticiónActual->bloqueado)**
- **peticiónAnterior = peticiónActual**
- **peticiónActual= obtenerElementoSiguiente(peticionAnterior)**
- **quitarElementodeLista (listaPeticiones, peticionAnterior)**
- **while (peticionActual !=NULL) &&**
  (peticionActual->operación == READ) &&
  (peticionAnterior-> operacion == peticionActual->operacion)&&
  (peticionAnterior->bloque_id == peticionActual->bloque_id)
  - **copiar de Disco_DMA a petcionActual -> buffer**
  - **peticionActual->bloqueado->estado=LISTO**
  - **Insertar (listaListos, peticionActual->bloqueado)**
  - **peticionAnterior=peticionActual**
  - **peticionActual=obtenerElementoSiguiente (peticionAnterior)**
  - **quitarElementodeLista (listaPeticiones, peticionAnterior)**
- **Si (peticionActual != NULL)**
  - **SI (peticionActual->operación == WRITE)**
    - **Copiar de peticionActual -> buffer a DISCO_DMA**
    - **EscrituraDisco (peticiónActual->bloque_id, DISCO_DMA)**
  - **Si (peticionActual->operación ==READ)**
    - **lecturaDisco(peticionActual->bloque_id, DISCO_DMA)**

NIA y Grupo: .....
Nombre y apellidos: .....

Si **contestación adecuadamente** entonces **0.75**

Funcion_Peticion_Bloque (bloque_id,operación, buffer)

- petAux = crearPeticion(procesoActual, bloque_id, operación, buffer)
- **insertarOrdenadamente** (listaPeticiones, petAux)

- Si peticiónActual == NULL
  - peticionActual = petAux
  - **Si (operación == WRITE)**
    - **Copiar de buffer a DISCO_DMA**
    - **escrituraDisco(peticionActual->bloque_id,DISCO_DMA)**
  - Si (operación ==READ)
    - lecturaDisco(peticionActual->bloque_id_DISCO_DMA)

- procesoActual->estado=BLOQUEADO
- procesoAnterior=procesoActual
- procesoActual = Planificador ()
- procesoActual->estado = EJECUTANDO
- CambioContexto (procesoAnterior, procesoActual)