Nombre:
DNI:

Segundo control de teoría

Todas las respuestas se tienen que justificar brevemente. **Una respuesta sin justificar se dará como no contestada.**

1. *(4 puntos) Sistema de ficheros y Gestión de memoria*

Se quiere añadir a ZeOS gestión de memoria virtual. Para ello necesitamos añadir el código de la rutina para gestionar el fallo de página (*page fault exception*) y un proceso que se encargue de ejecutar el algoritmo de reemplazo (*pageout daemon*).

Para implementar los accesos a disco de ambos códigos se quiere utilizar un único gestor que se encargará de gestionar el área de swap. Este gestor leerá las páginas solicitadas por la rutina de gestión del fallo de página y escribirá las páginas víctimas seleccionadas por el pageout daemon. Mientras queden frames libres en el sistema el gestor priorizará las peticiones de la rutina del fallo de página. Sólo si no quedan frames libres priorizará las peticiones del pageout daemon.

Las operaciones solicitadas por la rutina de gestión de fallo de página tienen que ser síncronas. Esta rutina también se encarga de avisar al pageout daemon de que es necesario empezar a limpiar frames.

Las operaciones solicitadas por el pageout daemon tienen que ser asíncronas. Para cada página víctima hará una solicitud de escritura. El resultado de cada operación indicará si ha acabado sin error y cuál es la posición en el área de swap que ocupa la página.

En el anexo del final del enunciado tenéis el esquema que sigue una llamada a sistema síncrona usando gestor. A continuación tenéis el pseudo-código que deberá ejecutar cada uno de estos componentes. Se han marcado en negrita las funciones que implican alguna sincronización entre los 3 bloques de código.

Nombre:
DNI:

```
pageout_daemon(){
    while(1) {
       esperar_orden_de_limpiar()

       enviadas = 0
       while (enviadas < num_paginas_a_swap) {
           pag[enviadas] = seleccionar_victima()
           pedir_escritura_en_swap(pag[enviadas]<<12)
           enviadas++
       }

       for (i = 0; i < enviadas; i ++)
              posición_en_swap[i] = get_info_from_swap(pag[i])

       for (i = 0; i < enviadas; i++){
              liberar_frames(pag[i])
              actualizar_lista_regiones(pag[i], posición_en_swap[i], SWAP)
              actualizar_tabla_paginas(pag[i], NOPRESENTE)
       }

    }
}
```

```
page_fault_exception(){
    pagina = pagina_que_provoca_la_excepcion()
    if (es_valida(pagina)) {
       posicion_en_swap = obtener_info_lista_regiones(pagina)
       frame = alloc_frame()
       if (quedan_pocos_frames) {
              pedir_limpieza_de_frames()
       }
       ok = pedir_lectura_de_swap(posicion_en_swap, pagina<<12)
       if (ok) {
              actualizar_lista_regiones(pagina, frame, MEMORIA)
              actualizar_tabla_paginas(pagina, frame)
              retornar_reiniciando_instrucción()
       }
    }
    gestionar_acceso_invalido();
}
```

```
 gestor_swap(){
    while(1) {
       esperar_trabajo();
       iorb = elegir_iorb();
       if (es_lectura(iorb)){
          ok = leer_swap(iorb);
          rellenar_y_encolar_io_fin(iorb->id_io, ok, -1)
          avisar_fin_operación(iorb);
       } else {
          posición_swap = buscar_posicion_libre_swap();
          ok = escribir_swap(posición_swap, iorb);
           rellenar_y_encolar_io_fin(iorb->id_io, ok, posición_swap)
          avisar_fin_operación(iorb);
       }
}
```

Nombre:
DNI:

Se pide que contestes razonadamente a las siguientes preguntas:

1. *(0,75 puntos)* Sobre el inicio de las operaciones del gestor...
   a. ¿Cuántos semáforos se necesitan para implementar la función del gestor **`esperar_trabajo`**? Ponles nombre para su posterior identificación.

   b. ¿En cuál o cuáles de las siguientes rutinas pondrías algún *sem_signal* sobre este/estos semáforo/s?

función	**Nombre del semáforo/s**
`esperar_orden_de_limpiar()`	
`pedir_escritura_en_swap(...)`	
`get_info_from_swap(...)`	
`pedir_limpieza_de_frames()`	
`pedir_lectura_de_swap(...)`	
`esperar_trabajo()`	
`avisar_fin_operación(...)`	

2. *(0,75 puntos)* Sobre la finalización de las operaciones del gestor…
   a. ¿Cuántos semáforos se necesitan para implementar la función `avisar_fin_operación()`?

   b. ¿En cuál o cuáles de las siguientes rutinas colocarías algún *sem_wait* sobre uno de estos semáforos?

función	**Nombre del semáforo/s**
`esperar_orden_de_limpiar()`	
`pedir_escritura_en_swap(...)`	
`get_info_from_swap(...)`	
`pedir_limpieza_de_frames()`	
`pedir_lectura_de_swap(...)`	
`esperar_trabajo()`	
`avisar_fin_operación(...)`	

Nombre:
DNI:

3. *(0,75 puntos)* Sobre la sincronización entre el pageout_daemon y el fallo de página
   a. ¿Cuántos semáforos se necesitan para implementar la función el sincronización entre el pageout-daemon y la rutina del fallo de página? Ponles nombre para su posterior identificación.

   b. Indica en cuál o cuáles de las siguientes funciones deberías usar uno de esos semáforos para implementar esa sincronización y de qué manera (*sem_wait* o *sem_signal*)

función	**Nombre del semáforo/s y operación**
`esperar_orden_de_limpiar()`	
`pedir_escritura_en_swap(...)`	
`get_info_from_swap(...)`	
`pedir_limpieza_de_frames()`	
`pedir_lectura_de_swap(...)`	
`esperar_trabajo()`	
`avisar_fin_operación(...)`	

4. *(0.5 puntos)* ¿Cuántas colas de iorb definirías para tener un acceso eficiente?

5. *(0.5 puntos)* ¿Qué campos pondrías en el iorb?

6. *(0.25 puntos)* ¿Cuántas colas de io_fin se necesitan?

Nombre:
DNI:

7. *(0.5 puntos)* Si decidimos que la escritura del pageout daemon sea síncrona, ¿afectaría de alguna manera al código del gestor?

2. *(3 puntos) Preguntas cortas*

a) ¿Cual o cuales de las llamadas a sistema vistas en clase relacionadas con sockets modifican la tabla de canales?

b) ¿Cuál es la unidad mínima de trabajo dentro del sistema de ficheros?

c) A la hora de gestionar el espacio ocupado de un dispositivo ¿hay algún mecanismo de asignación que pueda producir fragmentación externa? Pon un ejemplo.

d) ¿Qué estructura de Linux permite a varios procesos compartir el acceso a un fichero?

Nombre:
DNI:

e) El señor Baka Baka crea el siguiente trozo de código para "crear 2 procesos que incrementen una variable A":

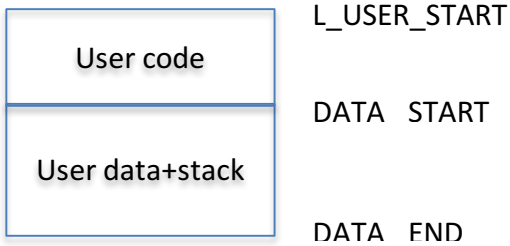
```
int A = 0;
sem_init(1,1);
fork();
sem_wait(1);
A++;
sem_signal(1);
```

El resultado que esperaba después del último sem_signal es que el valor de A en uno de los procesos fuera 1 y en el otro fuera 2, pero siempre le sale 1 en los dos casos. Suponiendo que las llamadas a sistema NO fallan nunca ¿Puedes explicarle qué está haciendo mal?

f) ¿Qué información guarda un inodo?

3. *(3 puntos) Gestión de memoria*

La siguiente figura representa el espacio lógico de direcciones de un proceso en ZeOS, cuando no se dispone de memoria dinámica:



Nombre:
DNI:

Contesta razonadamente a las siguientes preguntas.

a) *(1,5 puntos)* Si añadimos memoria dinámica
   i. ¿Afectará de alguna manera al esquema del espacio lógico de direcciones? Si es así representa en la siguiente figura tu propuesta para la nueva configuración

   ii. ¿Afectará de alguna manera a la gestión del espacio físico del proceso?

   iii. ¿Afectará de alguna manera a la implementación de la llamada a sistema exit?

b) *(1,5 puntos)* Si añadimos memoria virtual
   i. ¿Afectará de alguna manera al esquema del espacio lógico de direcciones? Si es así representa en la siguiente figura tu propuesta para la nueva configuración

Nombre:
DNI:

ii. ¿Afectará de alguna manera a la gestión del espacio físico del proceso?

iii. ¿Afectará de alguna manera a la implementación de la llamada a sistema exit?

ANEXO

Esquema de una llamada a sistema síncrona usando un gestor: