SO2

Nombre:
DNI:

Examen final de laboratorio

Justifica todas tus respuestas. Una respuesta sin justificación se considerará errónea.

Al descomprimir el fichero **final.tar.gz** creará un directorio *final* con el código necesario para el ejercicio 1 de este examen. Recuerda empaquetar de nuevo todo el código con tus cambios y entregarlo via racó: *tar zcvf mifinal.tar.gz final*

1. (2 puntos) Mecanismos de Entrada al Sistema

Añade la siguiente llamada a sistema a la implementación de ZeOS:

int positive_sum(int a, int b, int c, int d)

Esta rutina devolverá la suma de los 4 enteros pasados como parámetros. Si alguno de los parámetros es un negativo devolverá un -1 y colocará en una variable global errno el valor 34.

Esta llamada a sistema debe usar la interrupción 0x90.

NOTA: Aunque este ejercicio no requiera la entrada a sistema para cumplir la funcionalidad, es requisito indispensable que lo haga.

2. (1 punto) Gestión de procesos

a) ***(0.75 puntos)*** Dada la página P que no está asignada dentro del espacio lógico de un proceso, completa el siguiente código para realizar la herencia de datos de usuario usando única y exclusivamente esta página extra. Puedes suponer que la variable *frames[NUM_PAG_DATA]* contiene los frames asignados para la zona de datos del nuevo proceso.

```
int data_start = NUM_PAG_KERNEL + NUM_PAG_CODE;
for( i = 0; i < NUM_PAG_DATA; ++i  ) {




        copy_data(




}
```

Nombre:
DNI:

b) ***(0.25 puntos)*** ¿Qué inconveniente tiene este sistema frente al código visto en clase en el que se usan tantas páginas extra como ocupa la zona de datos?

3. (1 punto) Gestión E/S

Supón que en ZeOS tienes un proceso que ha intentando hacer una lectura de teclado: *read(0, &buff, size)*. El sistema no tiene suficientes carácteres disponibles y por lo tanto bloquea el proceso. A continuación, se recibe una interrupción de teclado, y el carácter recibido era el que faltaba para completar la petición del proceso anterior. En esta situación, y desde el código de la misma interrupción de teclado, ¿sería correcto usar el siguiente código para copiar los carácteres disponibles en el sistema al buffer de usuario de ese proceso?

```
copy_to_user(system_buf, buff, size);
```

Donde *buff* y *size* son los parámetros que nos ha pasado el usuario y *system_buf* es un buffer de sistema que contiene todos los bytes que ha pedido el usuario.

4. (1 punto) Semáforos

Dado una rutina *main* que crea 3 procesos (A, B y C) y cada uno ejecuta la rutina *fA*, *fB* y *fC* respectivamente. Indica el **mínimo** código necesario para garantizar que el orden de ejecución de los procesos sea C, B y A.

a) Código a añadir al principio de la rutina *main*

b) Código a añadir al principio de la rutina *fA*

c) Código a añadir al principio de la rutina *fB*

d) Código a añadir al principio de la rutina *fC*

5. (5 puntos) Memoria

En un sistema ZeOS, en el que **sólo hay procesos** (sin threads) y sin memoria dinámica, hay que añadir un mecanismo para crear regiones de memoria compartida entre procesos. Estas regiones se identificarán por un número natural y los procesos podrán añadirlas a su espacio de direcciones. Cualquier cambio hecho por un proceso en una de estas regiones será visible por el resto de procesos que la compartan. Al crear un nuevo proceso estas regiones se heredaran.

Para ello hay que implementar las siguientes llamadas a sistema:

- *int shmget (int key, int size):* Crea una nueva región de memoria compartida identificada con el número *key* de tamaño *size* bytes. Esta llamada a sistema devuelve un identificador para que el proceso pueda referenciar la región localmente o -1 (y errno con un valor ENOMEM) si no hay memoria suficiente para crear la región. Si la región ya existía no crea ninguna región nueva, sino que devuelve un identificador para la región existente.
  NOTA: Puedes suponer que habrá un máximo de MAX_REGIONS definidas en el sistema y que cada región puede tener un tamaño máximo de MAX bytes.

- *void* shmat(int id, void* addr):* Añade la región de memoria identificada por el parámetro *id* al espacio de direcciones del proceso actual con permisos de lectura y escritura. Esta llamada a sistema devuelve la dirección inicial donde se ha ubicado

esta región, intentará asignarla en la dirección pasada por el usuario, pero si esta dirección no está alineada a página o si hay algún solape con alguna región anterior, se asignará la dirección más cercana e inferior a *addr* alineada a página. Si no se puede colocar la región devolverà la dirección 0 y errno cogerá el valor EINVAL.
NOTA: Puedes suponer que como máximo habrá un máximo de MAX_REGIONS regiones por proceso.

- *int shmdt(void *addr):* Borra la región compartida cuya dirección inicial sea *addr* del espacio de direcciones del proceso actual.

- *int shmrm(int key):* Marca la región compartida identificada por key para ser borrada. Si la región no está usada por ningún proceso, se borrará directamente, si no el último proceso en borrar la región de su espacio de direcciones se encargará de hacerlo.

Responde a las siguientes preguntas:

a) ¿Es necesario crear alguna estructura global? Caso afirmativo indica los campos que debe tener y justifica cada uno.

b) ¿Es necesario modificar el PCB? Caso afirmativo indica los campos que modificas y/o añades y justifica cada uno.

Nombre:
DNI:

c) ¿Cual es la granularidad mínima de la llamada **shmget**?

d) Al **crear** una nueva región de memoria ¿qué rutina usas para conseguir memoria?

e) Supón que el usuario hace un *shmget(KEY, 6000)* ¿Es necesario que las direcciones de memoria devueltas por la rutina anterior sean todas consecutivas?

f) Al hacer un *shmat*, ¿como puedes detectar un solape? Dibuja los 4 casos posibles.

Nombre:
DNI:

g) Indica el pseudocódigo para la rutina de sistema *shmdt*

h) ¿Seria necesario modificar la llamada a sistema *fork*?

i) ¿Seria necesario modificar la llamada a sistema *exit*?

j) ¿Seria necesario modificar el *task_switch*?