SOA2 (13/04/2023)

Primer control de laboratorio

Crea un fichero que se llame “respuestas.txt” donde escribirás las respuestas para los apartados de los ejercicios del control. Indica para cada respuesta, el **número de ejercicio y el número de apartado** (por ejemplo, 1.a, 1.b, ...).

Importante: para cada uno de los ejercicios tienes que partir del código suministrado de Zeos.

1. *(3 puntos)* Comprensión de Zeos

a) (1 punto) ¿En qué página lógica se encuentra el vector de procesos?¿Como la has encontrado?¿Y en qué página física?
b) (1 punto) ¿En qué momento del código proporcionado (fichero y línea de código) se inicializa la pila de sistema a usar por el proceso inicial?
c) (1 punto) ¿Como modificarías el Makefile para linkar un objeto (examen.o) en el ejecutable de usuario?

2. *(3 puntos) Show me the processes!*

Queremos modificar nuestro ZeOS para permitir añadir un nombre a los procesos que creamos, por lo tanto, modifica la llamada a sistema fork para añadir un nuevo parámetro:

```
int fork(char *name)
```

Esta llamada funciona igual que la llamada *fork* vista en laboratorio pero además se guarda el nombre del proceso copiando el contenido de ´name´ (hasta que encuentre un \0 o haya copiado 13 carácteres que es la longitud máxima de los nombres de nuestros procesos).

Además añade una nueva llamada:

```
void ps(void)
```

Que muestre por pantalla la lista de procesos en ejecución, mostrando su PID y su nombre. Esta llamada no debe poder invocarse por el mecanismo tradicional sino que se tiene que ejecutar via la interrupción 200 y sólo debe ejecutar esta funcionalidad.

Se pide:
a) Modifica el código del wrapper de la llamada *fork.*
b) Añade el wrapper de la llamada *ps*.
c) Implementa el código del handler de *ps*.
d) Añade el código necesario para inicializar estas llamadas.
e) Añade el código de la rutina *sys_fork*.
f) Añade el código de la rutina *sys_ps*.

3. (2 puntos) Task switch

El reputado investigador BakaBaka propone el siguiente código en ensamblador para hacer el cambio de contexto y ahorrarnos el wrapper visto en teoría:

```
1:  ; void task_switch(struct task_struct *new)
2:  task_switch:
3:    pushl %ebp
4:    movl %esp, %ebp
5:    pushl %edi
6:    pushl %esi
7:    pushl %ebx
8:    pushl XXX ( %esp ) ; Apilar´new´
9:    call save_EBP ; Función que guarda el contenido del registro EBP
10:                   ; en el campo KERNEL_ESP del PCB del proceso actual
11:   call change_next_stack ; Función que actualiza la pila de sistema a
12:                          ; usar la próxima vez que se entre a sistema
13:   call change_memory; Función que actualiza el espacio @para que use
14:                      ; el del proceso pasado como parámetro
15:   call get_next_EBP ; Función que devuelve el valor guardado en el
16:                     ; campo KERNEL_ESP del PCB pasado como parámetro
17:   movl %eax, %esp
18:   popl %ebx
19:   popl %esi
20:   popl %edi
21:   popl %ebp
22:   ret
```

Responde justificadamente a estas preguntas:

a) ¿Qué valor hay que colocar en XXX para apilar el parámetro de la función *task_switch*?
b) ¿Qué detalle hay que cambiar para que funcione este código? Indica el código necesario para solventarlo.
c) Indica el código necesario para implementar la función *get_next_EBP*.
d) Indica como debe cambiar el contexto del proceso idle en su inicialización.

4. (2 puntos) Copy user data

Queremos implementar la siguiente función:

```
int copy_user_data(char* src, char* dst,
                   struct task_struct* PCB)
```

Que copia el contenido de memoria entre las direcciones ´src´ y ´dst´ del proceso actual, hacia las mismas direcciones del proceso ´PCB´, reservando la memoria física requerida. Si hay algún error devuelve un número negativo.

a) Implementa la función copy_user_data
b) Modifica la rutina sys_fork para que use esta función.

5. Entrega

Sube al Racó los ficheros “respuestas.txt” junto con el código que hayas creado en cada ejercicio. Para entregar el código utiliza:

```
> tar zcfv examen.tar.gz zeos
```