Documento de Seguimiento

1. Nivel alcanzado

Hemos alcanzado el nivel 1 satisfactoriamente y estamos intentando implementar sbrk(). Tenemos el código hecho pero no nos funciona. No hemos realizado prioridades ni gestión de fps porque no hemos finalizado el nivel 2.

2. Decisiones nuevas tomadas.

- Implementar el buffer circular lo primero de todo: Derivado del primer problema encontrado en el documento de diseño. Ahora el buffer se usa tanto cuando hay un proceso como en modo multiproceso.
- Nuevo juego de pruebas para put_screen(): Creamos el juego de pruebas específico para put_screen en el que si pulsas una tecla tiene que escribir una matriz de X en la pantalla.
- Implementamos el buffer circular en dos ficheros nuevos: buffer.c y buffer.h, no dentro de un fichero ya creado.
- sbrk no es void: Devuelve un puntero a la primera posición de memoria nueva alocatada.
- Nuevo juego de pruebas para sbrk(): Simple, que testee si sbrk incrementa el valor de retorno (es decir, la posición inicial de memoria alocatada)

3. Problemas encontrados respecto al documento de Diseño y su solución

- Implementación de get_key diferente(char *c): Inicialmente pensamos que get_key tenía que interactuar con la interrupción de teclado. Pero estábamos equivocados: Para implementar correctamente tanto get_key como el buffer circular, la interrupción de teclado debía escribir directamente en el buffer y get_key leer de él. Para ello creamos 2 funciones (read_buffer() y write_buffer()) y la estructura junto con sus punteros en buffer.h y buffer.c
- Modificación de la interrupción de teclado, no de la interrupción de reloj: Para enseñar por pantalla o actuar en función de la tecla pulsada se debe modificar la interrupción de teclado haciendo que escriba el valor pulsado en el buffer, haciendo uso de write_buffer(). Suponemos que fue un lapsus y por eso no nos referimos a la interrupción adecuada.
- modificar exit + implementación sbrk(): No teníamos forma de saber cuál había sido la última posición de memoria alocatada, así que tuvimos que añadir el puntero adicha posición de memoria: last_pos, necesario para alocatar y dealocatar de memoria forma dinámica, usado tanto en sbrk() como en exit().

4. Listado de Tareas Realizadas

Leyenda

-> Hecho
-> En proceso
-> No hecho

- Crear Buffer circular -> Realizado antes
- Implementación get_key(char *c): obtener una tecla que pulse el usuario
    a. Modificar sys_call_table.S
    b. Hacer Wrapper
    c. Programar Rutina de Servicio
    d. Modificar la interrupción de teclado -> Realizado antes (y no la de reloj)
    e. Hacer Juego de Pruebas get_key()
- Implementación put_screen(char *s): Llamada a sistema que pinta el escenario de 25x80 en la pantalla.
    a. Modificar sys_call_table.S
    b. Hacer Wrapper
    c. Programar Rutina de Servicio
    d. Hacer Juego de pruebas get_key() -> Añadido
- Crear la matriz(en user.c)
- Hacer Juego de Pruebas Nave - int put_screen(char *s)
- Implementación sbrk() : saber cual fue ultima pos dentro de mem y en caso de falta poner + pags.
    a. Modificar sys_call_table.S
    b. Hacer Wrapper
    c. Programar Rutina de Servicio
    d. Hacer juego de pruebas sbrk() -> Añadido
- exit() modificar
- Crear vector de matrices (user.c)
- Hacer Juego de Pruebas fruta + Diferentes escenarios
- Crear sistema de prioridades
- Modificar interr. hardware clock routine para que solo llame cada x tics
- Juego de pruebas: demasiados ticks!
- Sistema de prioridades cuando hay muchos procesos + uso del buffer
- Juego de pruebas: pocos ticks!
- Juego de pruebas final: Snake -> Nos falta el cuerpo de la serpiente y hacer distintos escenarios, así como mejorar el problema en la movilidad, que suponemos que se arreglará cuando implementemos gestión de fps.