SIMULADOR DE UN SISTEMA INFORMÁTICO MULTIPROGRAMADO

TAREAS - V1

Tareas iniciales

Copia el directorio `/var/asignaturas/ssoo/2017-2018/V1` de `ritchie` en tu directorio personal. El trabajo a realizar en los ejercicios siguientes se desarrollará sobre la copia indicada de los ficheros contenidos en el directorio `V1`, dentro de tu directorio personal.

Notas:

- **No se deben modificar los ficheros** `OperatingSystemBase.*` `ComputerSystemBase.*` **ni** `messagesTCH.txt`
- **Suprímanse todas las tildes de los mensajes generados por el simulador.**

Ejercicios

0. Incorpora las modificaciones realizadas en el ejercicio de la V0 en el que añadías la instrucción MEMADD (tiene que ser reimplementada por la aparición de la MMU).

1. Implementa una función `ComputerSystem_PrintProgramList()` que muestre en pantalla **los programas de usuario** contenidos en el vector `programsList`; ten en cuenta que el contenido de cada posición del vector es un puntero a un `struct` de C, y que la primera posición del vector está reservada para el programa que implementa el sipID que no es un programa de usuario.
   Para mostrar la información en la pantalla se tendrá que usar la función `ComputerSystem_DebugMessage()`, utilizando los números de mensajes 101 y 102 (los mensajes añadidos por el alumno deben ir en el fichero de mensajes "`messagesSTD.txt`", y usando la constante `INIT` como valor para el segundo argumento de la misma (sección de interés). El mensaje mostrado deberá que tener el aspecto siguiente (`<tab>` es un tabulador y el nombre e instante de llegada están en color cyan):

   ```
   User program list:
   <tab>Program [ejemplo1] with arrival time [0]
   <tab> …
   ```

2. Muestra la información del ejercicio 1, cuando se tenga la información necesaria para hacerlo en `ComputerSystem_PowerOn()`

3. Ejecuta el simulador con dos argumentos que sean el mismo programa de la siguiente forma:

   ```
   ./Simulator HD programVerySimple programVerySimple
   ```

   ¿Se ejecuta 2 veces? ¿Por qué? Crea un programa `newProgramVerySimple` que se ejecute dos veces si se ejecuta el simulador así:

   ```
   ./Simulator HD newProgramVerySimple newProgramVerySimple
   ```

4. Ejecuta el simulador con un número de programas en línea de comandos que supere la capacidad de la tabla de procesos. ¿Qué programa queda en ejecución? ¿Por qué? Realiza las modificaciones siguientes que solucionan el problema:

    a. Modifica la función `OperatingSystem_CreateProcess()`, para que devuelva a la función `OperatingSystem_LongTermScheduler()` el valor `NOFREEENTRY` cuando la primera función fracasa al intentar conseguir una entrada libre en la tabla de procesos.

    b. Modifica la función `OperatingSystem_LongTermScheduler()`, para que distinga el caso de creación de proceso con éxito y el error en caso de que lo hubiese, indicándolo mediante la función `ComputerSystem_DebugMessage()`, utilizando el número de mensaje 103, y la constante `ERROR` como valor para el segundo argumento de la misma (sección de interés). El mensaje debe tener el aspecto siguiente:

ERROR: There are not free entries in the process table for the program [nombre_programa]

5. Ejecuta el simulador con el nombre de un programa inexistente en línea de comandos. Estudia lo que sucede. Repite la ejecución, pero, en este segundo caso, indica un nombre de programa que no incluya un valor entero para el tamaño del proceso. Estudia el resultado y realiza las modificaciones siguientes:

    a. Modifica la función `OperatingSystem_LongTermScheduler()`, para que distinga el caso de creación de proceso con éxito y el error en caso de que lo hubiese, indicándolo en el segundo caso mediante la función `ComputerSystem_DebugMessage()`, utilizando el número de mensaje 104, y la constante `ERROR` como valor para el segundo argumento de la misma (sección de interés). El mensaje debe tener el aspecto siguiente:

    ERROR: Program [nombre_programa] is not valid [--- mensaje causa del error ---]

    El mensaje cambiará según sea al error (ver apartados b y c).

    b. Modifica la función `OperatingSystem_CreateProcess()`, para que devuelva a la función `OperatingSystem_LongTermScheduler()` el valor `PROGRAMDOESNOTEXIST` cuando la primera función fracasa si el programa no existe. El mensaje de causa del error que se mostraría en `OperatingSystem_LongTermScheduler()` sería:

    ERROR: Program [nombre_programa] is not valid [--- it does not exist ---]

    c. Haz lo mismo para el valor `PROGRAMNOTVALID`, devuelto cuando no se encuentran los valores enteros válidos correspondientes al tamaño y la prioridad, dentro del programa ejecutable. El mensaje de causa del error sería para ambos casos:"

    ERROR: Program [nombre_programa] is not valid [--- invalid priority or size ---]

6. Ejecuta el simulador con el siguiente programa, que especifica un tamaño de proceso que supera el máximo posible por proceso.

```
65
2
ADD 100 6
WRITE 7
```

```
NOP
TRAP 3
```

Estudia el resultado y realiza las modificaciones siguientes:

a. Modifica la función `OperatingSystem_CreateProcess()`, para que devuelva a la función `OperatingSystem_LongTermScheduler()` el valor `TOOBIGPROCESS` cuando la primera función fracasa al intentar crear un programa de un tamaño superior al espacio reservado en memoria principal. Se debe detectar el error al intentar obtener memoria para el programa.

b. Modifica la función `OperatingSystem_LongTermScheduler()`, para que distinga el caso de creación de proceso con éxito y el error en caso de que lo hubiese, indicándolo mediante la función `ComputerSystem_DebugMessage()`, utilizando el número de mensaje 105, y la constante `ERROR` como valor para el segundo argumento de la misma (sección de interés). El mensaje debe tener el aspecto siguiente:

ERROR: Program [nombre_programa] is too big

7. Ejecuta el simulador con un programa que tenga más instrucciones que su tamaño. Estudie el resultado y el tratamiento que hace al respecto la función `OperatingSystem_LoadProgram();` y realice la siguiente modificación:

a. Modifica la función `OperatingSystem_CreateProcess()`, para que devuelva a la función `OperatingSystem_LongTermScheduler()` el valor `TOOBIGPROCESS` cuando la primera función detecta que un programa tiene más instrucciones que el tamaño especificado.

b. Nótese que el mensaje de error sería el mismo que el del apartado 6-b, y no hace falta hacer modificaciones al respecto.

8. Pon a 3 la constante `INITIALPID` en `OperatingSystem.h` y comprueba cómo afecta a la asignación de PIDs a los procesos. En lo sucesivo, el simulador debería funcionar independientemente del PID asociado a cada proceso. Ten en cuenta en las modificaciones a partir de este momento, que no debe depender la ejecución del simulador de la asignación de PIDs a los procesos.

9. Estudia la estructura de datos que contiene la lista de procesos listos para su ejecución.

a. Implementa una función denominada `OperatingSystem_PrintReadyToRunQueue()` que muestre en pantalla el contenido de la cola de procesos LISTOS. Para mostrar la información en la pantalla se tendrá que usar la función `ComputerSystem_DebugMessage()`, utilizando los números de mensajes 106 y sucesivos, y la constante `SHORTTERMSCHEDULE` como valor para el segundo argumento de la misma (sección de interés). El mensaje mostrado deberá tener el aspecto siguiente:

```
Ready-to-run processes queue:
<tab>[1,0], [3,2], [0,100]
```

Donde los números en verde se refieren a identificadores de procesos (PID's) incluidos en cola y los números en color negro, serán sus prioridades.

b. Añade una invocación a la función recién creada al final de la función `OperatingSystem_MoveToTheREADYState()`.

10. Se desea ver en pantalla todos los cambios de estado que sufren los procesos. Para ello:

    a. Pega en el fichero `OperatingSystem.c` la definición de la siguiente estructura de datos:

    ```
    char * statesNames [5]={"NEW","READY","EXECUTING","BLOCKED","EXIT"};
    ```

    b. Localiza en el código del SO todos los puntos en los que los procesos sufren un cambio de estado y, en cada uno de ellos, inserta una llamada a la función `ComputerSystem_DebugMessage()` que muestre un mensaje con el aspecto siguiente (número de mensajes 110 y 111 respectivamente, y sección `SYSPROC`):

    ```
    Process [2] moving from the [READY] state to the [EXECUTING] state
    ```

    Si el proceso es nuevo, el mensaje será:

    ```
    New process [2] moving to the [NEW] state
    ```

11. Modificaciones en la política de planificación a corto plazo.

    a. Pasará a ser de colas multinivel, con dos colas: la de mayor prioridad, para los procesos de usuario y, la de menor prioridad, para los demonios del sistema. Para ello, las estructuras de datos relacionadas con la gestión de la cola de LISTOS pasan a estar definidas así:

    ```
    #define NUMBEROFQUEUES 2            // in OperatingSystem.h
    #define USERPROCESSQUEUE 0          // in OperatingSystem.h
    #define DAEMONSQUEUE 1       // in OperatingSystem.h
    // En OperatingSystem.c
    int readyToRunQueue [NUMBEROFQUEUES][PROCESSTABLEMAXSIZE];
    int numberOfReadyToRunProcesses[NUMBEROFQUEUES]={0,0};
    ```

    Con el fin de que cada proceso conozca la cola a la que pertenece, y tenga fácilmente accesible la información, será necesario ampliar la información almacenada en el PCB con un nuevo campo:

    ```
        int queueID;
    ```

    b. Modifica la función `OperatingSystem_PrintReadyToRunQueue()`, para que muestre las colas de READY-to-RUN, con el aspecto siguiente (números de mensajes nuevos 112 y sucesivos y sección `SHORTTERMSCHEDULE`):

    ```
    Ready-to-run processes queue:
    <tab>USER: [1,0], [3,2]
    <tab>DAEMONS: [0,100]
    ```

    c. Modifica lo necesario en el sistema (empezando desde la creación de los procesos) para que se utilice correctamente la doble cola.

12. Añade una nueva llamada al sistema `SYSCALL_YIELD` (defínela con el valor entero 4 en el enumerado correspondiente). Cuando el proceso en ejecución invoque esta llamada al sistema, cederá voluntariamente el control del procesador al proceso READY con prioridad idéntica a la suya, y que figure como proceso más prioritario en la cola LISTOS que le corresponde al proceso que invoca la llamada a sistema. Si no existiese tal proceso, la cesión del procesador no tendría lugar. Además, se realizará una llamada a la función

`ComputerSystem_DebugMessage()`, mensaje 115 y sección de depuración `SHORTTERMSCHEDULE`, que muestre un mensaje como el que sigue:

```
Process [1] transfers control of the processor to process [3]
```

13. Estudia la función `OperatingSystem_SaveContext()`

    a. ¿Por qué hace falta salvar el valor actual del registro PC del procesador y de la PSW?

    b. ¿Sería necesario salvar algún valor más?

    c. A la vista de lo contestado en los apartados a) y b) anteriores, ¿sería necesario realizar alguna modificación en la función `OperatingSystem_RestoreContext()`? ¿Por qué?

    d. ¿Afectarían los cambios anteriores a la implementación de alguna otra función o a la definición de alguna estructura de datos?

14. Modifica la función `OperatingSystem_Initialize()` para que la simulación finalice cuando el Planificador a Largo Plazo sea incapaz de crear proceso de usuario alguno (revise la función `OperatingSystem_TerminateProcess()` para recordar cómo se puede provocar el fin de la simulación).

15. Se desea modificar el comportamiento de algunas instrucciones para que pasen a ser instrucciones privilegiadas y que solo puedan ser ejecutadas en caso de que el modo de ejecución del procesador sea protegido. Para ello:

    a. Modifica las implementaciones de las instrucciones `HALT`, `OS` e `YRET` para que se ejecuten únicamente cuando el procesador esté ejecutándose en modo protegido. En caso de que no sea así, el procesador deberá elevar una excepción (es decir, una interrupción de tipo excepción).

    b. Verificar que solamente el sistema operativo y los daemons se ejecuten en modo protegido.