SISTEMAS OPERATIVOS

CURSO 2017-2018

SIMULADOR DE UN SISTEMA INFORMÁTICO MULTIPROGRAMADO CON INTERRUPCIONES DE RELOJ y un nuevo Planificador a Largo Plazo

Manual V3

Introducción

Al simulador de la versión V2, le modificamos el Planificador a Largo Plazo, de forma que los programas puedan llegar en cualquier momento (no necesariamente en el instante 0 como hasta ahora).

Se han producido en el simulador cambios en sus diferentes componentes, que pasamos a detallar. Donde no haya habido cambios o estos hayan sido poco relevantes, no se detallará la naturaleza de los mismos.

DISEÑO

El Sistema Informática

- Estructuras de datos añadidas/modificadas:
    - Cola de llegada de programas: `arrivalTimeQueue` gestionada como un montículo binario, usando el instante de llegada como criterio de ordenación. La gestión se hace de forma similar al resto de montículos utilizados en el sistema.

El sistema operativo

- Funcionalidad:
    - Planificador a Largo Plazo (PLP) o Long-Term Scheduler (LTS).
        - Se invoca en el inicio, y en cada interrupción de reloj.
    - Rutinas de tratamiento de interrupción.
        - Se modifica la rutina de tratamiento de las interrupciones de reloj, para invocar al PLP y obrar en consecuencia.

SIMULADOR DE UN SISTEMA INFORMÁTICO MULTIPROGRAMADO con Interrupciones de reloj y un nuevo Planificador a Largo Plazo

Tareas V3

Tareas iniciales

Saca un duplicado de tu directorio `V2` (una vez completados los ejercicios de la V2) denominándolo **`V3`**. El trabajo a realizar en los ejercicios siguientes se desarrollará sobre la copia indicada de los ficheros contenidos en el directorio `V3`, dentro de tu directorio personal. Haz un `make clean` para limpiar el código.

Copia los ficheros de **`/var/asignaturas/ssoo/2017-2018/V3-studentsCode`** a tu directorio V3.

Ejercicios

Vamos a eliminar la restricción de que los procesos tengan que llegar al sistema obligatoriamente en el instante de tiempo cero. A partir de esta versión, se puede especificar en línea de órdenes el instante de llegada (algo así como el instante en que se hace doble clic en un icono en un entorno Windows) de cada proceso al sistema:

```
./Simulator A ej1 4 ej2 5 ej3 7 eje4 0
```

Para conseguirlo, es necesario utilizar una cola de llegada de programas, implementada con un montículo; que es lo que se hace en el ejercicio 0:

0. La implementación y uso de la cola de llegada de programas.
   a. Añade las declaraciones siguientes en el fichero **`ComputerSystem.c`** para definirlas:

      ```
      int arrivalTimeQueue[PROGRAMSMAXNUMBER];
      int numberOfProgramsInArrivalTimeQueue=0;
      ```

   b. Se necesita código adicional de manejo de la cola de llegada de programas en el ComputerSystem que se proporciona en la nueva versión de los ficheros `ComputerSystemBase.c` y `ComputerSystemBase.h`
      También hay código en el Heap para manejar el montículo de la cola de llegada, que necesita las estructuras creadas en el apartado anterior Para poder usarlo, debes añadir en **`ComputerSystem.h`** la definición siguiente para que compile el código necesario de Heap:

      ```
      #define ARRIVALQUEUE
      ```

c. La función `ComputerSystem_FillInArrivalTimeQueue()` que se encuentra en el fichero **`ComputerSystemBase.c`**, se encarga de meter en la cola de llegada (montículo) los programas de la **`programList`**, ordenados por instante de llegada.
   Inserta una llamada a dicha función, **justo antes de la llamada inicial al planificador a largo plazo**, en la inicialización del sistema operativo.
d. Añade una llamada a la función `OperatingSystem_PrintStatus()` justo después de la llamada a la función anterior (del apartado 0-c), en la inicialización del sistema operativo.

1. Puesto que ya va a haber mucho movimiento de procesos, nos conviene saber a qué proceso pertenece la instrucción ejecutada en cada momento.

   Modifica el mensaje mostrado en pantalla desde la función `Processor_DecodeAndExecuteInstruction` para que muestre el identificador del proceso en ejecución. Ya no sirve el mensaje 3 (de messagesTCH.txt) así que define para hacerlo un mensaje nuevo de número 130, y en la misma sección de interés que el anterior. El aspecto final del mensaje debe ser como el que sigue:

   ```
   [35] j -3 0 (PID: 1, PC: 1, Accumulator: -2, PSW: 4 [------------N--])
   ```

   Para tener acceso al PID del proceso en ejecución, implementa la función siguiente que devuelve el PID del proceso en ejecución:

   ```
   int OperatingSystem_GetExecutingProcessID() {…}
   ```

   Nótese que en el caso de la instrucción OS ejecutada en el tratamiento de interrupciones, se muestra el PID del proceso interrumpido. Y en el caso de la instrucción YRET, se muestra el del proceso interrumpido o el del nuevo proceso despachado si ha habido cambios en el tratamiento de la interrupción.

2. En **`OperatingSystemBase.c`** hay una nueva función `OperatingSystem_IsThereANewProgram()`. Dicha función devuelve 1 si hay al menos un programa de usuario que ha llegado antes del tic de reloj actual y todavía no ha sido tratado por el PLP. Si no hay ninguno, devuelve 0; y si ya no quedan programas de usuario devuelve -1.

   Es decir, si ejecutásemos el simulador así:

   ```
   ./Simulator A ej1 9 ej2 8 ej3 14 ej4 0
   ```

   E invocásemos a la función `OperatingSystem_IsThereANewProgram()` en el instante 12, devolvería como resultado 1 porque el programa `ej1` (y el `ej2`) ha llegado antes del instante 12, el programa `ej3` todavía no ha llegado y el programa `ej4` ya ha sido tratado anteriormente si `INTERVALBETWEENINTERRUPTS` está definido con valor 5.

   Usando esta función:

   Modifica tu planificador a largo plazo para que se intenten crear los procesos correspondientes **mientras llamadas sucesivas** a `OperatingSystem_IsThereANewProgram` devuelvan 1, indicando que hay programas que han llegado al sistema hasta el momento actual. Puedes mirar cómo se utiliza la función `Heap_poll()` en otras colas.

   Si el ejercicio está correctamente resuelto, el simulador sólo creará y ejecutará los programas que tengan instante de llegada igual a cero.

3. En la rutina de tratamiento de interrupciones:

    a. Añade una llamada a tu planificador a largo plazo dentro de la rutina de tratamiento de interrupciones de reloj, **colocándola después de despertar los procesos que tengan que despertarse de una llamada al sistema SLEEP**.

    De esta manera, el planificador a largo plazo se ejecutará:

        i. En la inicialización del sistema operativo
        ii. Con cada ocurrencia de una interrupción de reloj

    b. Si se ha creado algún proceso, se deberá hacer una llamada a la función: `OperatingSystem_PrintStatus()`

    c. Si el **PLP** ha creado algún proceso o el tratamiento de la `sleepingProcessesQueue` ha desbloqueado alguno, será necesario comprobar si el proceso en ejecución sigue siendo el más prioritario de todos los que pueden competir por el procesador. En caso de no serlo, será necesario sustituir al proceso en ejecución por el proceso más prioritario, mostrando además el mensaje con el aspecto siguiente (sección `SHORTTERMSCHEDULE`):

    `[27] Process [1] is thrown out of the processor by process [2]`

    En este caso (se cambia el proceso en ejecución), además se deberá hacer una llamada a la función: `OperatingSystem_PrintStatus()`

    Nótese que es lo mismo que se hacía en la V2 cuando se desbloqueaba un proceso más prioritario que el que está en ejecución, pero aplicado también a los posibles procesos nuevos creados por el PLP.

4. Para evitar posibles comportamientos incorrectos, revisa:

    a. La condición que detiene la simulación tras ejecutarse el planificador a largo plazo durante la iniciación del sistema operativo. Ten en cuenta que quizá no se tenga que crear ningún proceso en el instante 0, pero que sí haya algún programa con instante de llegada posterior.

    b. La condición que detiene la simulación tras ejecutarse el planificador a largo plazo en una interrupción de reloj.

    c. La condición que detiene la simulación cada vez que un proceso finaliza su ejecución.

Escuela Ingeniería Informática de Oviedo. Sistemas Operativos: Simulador V3 Final

MainMemory

MAR

MBR

mainMemory

0 … 59 60 … 119 120 … 179 180 … 239 240 … (*)

(*) = MAXMEMORYSIZE - 1

Clock

tics

MMU

Base

Limit

MAR

ComputerSystem

ProgramList

*PROGRAMDATA | *Name | Arrival | Type

0 → *Name, Arrival, Type

1

2 null

(*) null

(*) = PROGRAMMAXNUMBER-1

Arrival
(index in programList)

0 1 2 … (*)

(*) = PROGRAMMAXNUMBER-1

NumArrival

Messages Subsystem
…

Processor

accum

PC

IR

MAR

MBR

PSW

A

Int

VInt.

0	null
1	null
2	SysCall Entry
3	null
4	null
5	null
6	Exception Entry
7	null
8	null
9	ClockInt Entry
…	…
(*)	null

(*) = INTERRUPTTYPES -1

OperatingSystem

executingProcessID

sipID

NonTerminated

numberOfClockInterrupts

ProcessTable

PID	busy	initialAddr	size	state	priority	Copy PC	queueID	whenToWakeUp	…
0									
1									
2									
…									
(*)									

ReadyToRun (PID)

	0	1	2	…	(*)
USER					
DAEM					

NumReadyToRun

USER

DAEM

Sleeping (PID)

0 1 2 … (*)

NumSleeping

(*) PROCESSTABLEMAXSIZE - 1