Ingeniería en Informática

Sistemas Operativos

Trabajo Investigación

Implementación de planificador de procesos configurable

Equipo Docente	Nicanor Casas
	Graciela De Luca
	Waldo Valiente
	Gerardo Puyo
	Sergio Martín
	Federico Díaz
	Sebastian Deuteris

GRUPO: 5
Días de Cursada: *Miércoles / Viernes*

Alumnos		
Apellido	**Nombre**	**DNI**
Gonzalez	Demian	30655542
Lanzillotta	Hernan	27480269
de Vega	Gonzalo	31650431

Acreditación:

Instancia	Fecha	Calificación
PRE-ENTREGA	/ /2012	
ENTREGA	/ /2012	
FINAL	/ /2012	

Objetivo del TP

El objetivo de nuestro trabajo práctico es permitir la selección en tiempo de ejecución del planificador de procesos que utilizará SODIUM. Para lograr esto se implementará un comando de usuario (***planif***) que permitirá especificar el planificador a utilizar de entre los disponibles en el sistema, con el que también se le definirán algunos parámetros que apliquen al mismo, los cuáles afectarán la ejecución del algoritmo de planificación correspondiente. Con este comando, al seleccionar el planificador existente actualmente de tipo Round Robin se podrá especificar el quantum de tiempo asignado a los procesos.

Además de esto, se creará un nuevo planificador basado en prioridades. El mismo tendrá un comportamiento de tipo Round Robin, con la distinción de que el quantum de tiempo asignado a cada proceso será variable dependiendo de su prioridad. El comando para seleccionar este planificador permitirá establecer el quantum asignado a cada prioridad.

Resumen de modificaciones

Desde el inicio de la investigación hasta la fecha, analizando el código fuente de Sistema Operativo SODIUM, hemos modificado los siguientes archivos

- Makefile: del directorio /kernel
- definiciones.h: del directorio /kernel
- syscall.c: del directorio /kernel
- system.c: del directorio /kernel
- gdt.c: del directorio /kernel
- libsodium.h: del directorio /include/usr/lib
- Makefile: del directorio /usr/bin
- definiciones.h: del directorio /include/kernel
- sched.h: del directorio /include/kernel
- system.h: del directorio /include/kernel

Y hemos agregado el archivo planif.c en el directorio /usr/bin.
En adición a la tarea expuesta vamos a desarrollar los archivos prioridades.c y fifo.c en el directorio /kernel/planif.

Resumen de agregados

- build\Makefile
  Agregado planif.bin

- include\kernel\definiciones.h
  Se agrega variable iQuantumRR para contener el quantum variable de RR

- include\kernel\sched.h
  Se agrega #define PRIORIDAD 3 para planificador por prioridades
  Se agrega variable iTipoDePrioridadSeleccionada para utilizar en la selección del planificador
  Se agrega función vFnPlanificadorPORPRIORIDADES para contener el algoritmo de este planificador

- include\kernel\syscall.h
  Se agrega función lFnSysSetPrioridad para establecer la prioridad de un proceso

- include\kernel\system.h
  Se agrega función lFnSysSetPrioridad para establecer la prioridad de un proceso

- include\usr\lib\libsodium.h
  Se agrega #define PRIORIDADES 3 para planificador por prioridades

- kernel\planif\prioridades.c
  Se agrega el archivo con la función vFnPlanificadorPORPRIORIDADES

- kernel\Makefile
  Se agrega prioridades.o a la lista de OBJETOS_PLANIF

- kernel\syscall.c
  Se agrega function lFnSysSetPrioridad para establecer la prioridad de un proceso

- solo\Makefile
  Se agrega planif.bin a la lista de comandos LIST_PARAMS

- bin\Makefile
  Se agrega planif.bin a la lista de OBJETOS_BIN

- usr\bin\planif.c
  Se agrega el archivo para realizar la selección del planificador por medio de este comando de usuario

Documentación

Se agregan las siguientes funciones:

void **lFnSysSetPrioridad** (int iIndicePCB)
Establece la prioridad que le corresponde al proceso.

iIndicePCB (int): Indice del proceso dentro de la tabla de PCBs

Definición en la línea 1261 del archivo syscall.c.
Referenciado por iFnCrearPCB(), vFnHandlerTimer() y vFnPlanificador().

Se utiliza para que a un proceso (correspondiente al subíndice dentro de la tabla de PCBs) se le asigne una prioridad automáticamente. En el caso nuestro esta prioridad se decide en base al tamaño del mismo.
Esta función es invocada en la creación de un PCB.

void **vFnPlanificadorPORPRIORIDADES** ( )

Planificador Round - Robin con prioridades.

Definición en la línea 27 del archivo prioridades.c.

Hace referencia a CANTMAXPROCS, guiTipoMemoria, _stuPCB_::iEstado, iFlags, iIndicePCB, iIndicePCBAnterior, MODOPAGINADO, PROC_DETENIDO, PROC_EJECUTANDO, PROC_ELIMINADO, PROC_ESPERANDO, PROC_LISTO, PROC_NO_DEFINIDO, PROC_ZOMBIE, pstuPCB, ROJO, TSS_TAMANIO_STACK_SS0, uiFnSwapInBlock4k(), _stuPCB_::uiIndiceGDT_TSS, uliQuantum, ulProcActual, vFnImprimir() y vFnImprimirContextSwitch().

Referenciado por vFnPlanificador().

Esta función contiene el algoritmo del nuevo planificador Round Robin con prioridades. Es invocada en cuando se cumple el quantum del proceso actual.

Casos de prueba

- **Caso de prueba**: Cambiar en tiempo de ejecución el planificador que utiliza el S.O.

  - **Objetivo**: Se pretende cambiar de planificador activo en tiempo de ejecución por medio de un comando de usuario.

  - **Proceso**: Se ejecuta el comando "planif", pasándole como argumento alguno de los valores posibles. Las posibles combinaciones son las siguientes:
    - *planif 1 <quantum>* (Establece el planificador Round Robin con el quantum especificado)
    - *planif 2* (Establece el planificador FIFO)
    - *planif 3 <quantum1> <quantum2> <quantum3> <quantum4>* (Establece el planificador Round Robin con 4 colas de prioridades, asignándole a cada cola el quantum deseado)

  - **Salida**: Se muestra una leyenda y se cambia el planificador actual.

- **Caso de prueba**: Determinar cuál es el planificador en ejecución, junto con su configuración específica.

  - **Objetivo**: Se pretende ver, por medio de un comando, cuál es el planificador que se encuentra actualmente en ejecución.

  - **Proceso**: Se ejecuta el comando "*planif --actual*".

  - **Salida**: Se verá en pantalla una leyenda indicando cuál es el planificador actual, y, en los casos de los planificadores que tienen quantums asociados, los mismos también se muestran.

- **Caso de prueba**: Determinar la prioridad asociada a cada proceso.

  - **Objetivo**: Se pretende ver, en cualquier momento, la prioridad que tiene un proceso dentro del planificador de RR con colas de prioridades.

  - **Prerrequisito**: El planificador actual debe ser el RR con prioridades

  - **Proceso**: Se ejecuta el comando "*ps*"

- **Salida**: Se muestra la lista de procesos que tiene actualmente el sistema, junto a su prioridad asignada.

- **Caso de prueba**: Generar varios procesos con un comando de usuario y verificar el orden de ejecución, dentro de los planificador existentes y del nuevo planificador.

  - **Objetivo**: Se pretende generar varios procesos en el sistema para verificar la correcta creación de los mismos y cómo son administrados por el planificador en curso. Se pretende verificar también el orden en que son ejecutados los mismos, dependiendo del planificador seleccionado.

  - **Prerrequisito**: Establecer el planificador que se desea evaluar por medio del comando *planif*. Este ejercicio se puede ver afectado por una condición de carrera entre los distintos procesos que estarán involucrados, por lo que es conveniente configurar los planificadores con quantums altos (por ej. 200) para disminuir los efectos de esta condición y enfocarse en la prueba en sí.

  - **Proceso**: Se ejecuta el comando "*usuario 1*"

  - **Salida**: Se observa en pantalla la salida preparada por el grupo en la ejecución del comando, que muestra el orden en que fueron dándose los eventos. Aquí se puede distinguir la diferencia entre los planificadores. Si el planificador actual es RR, se verá que los procesos hijos son ejecutados en el mismo orden en que fueron creados. Esto se observa específicamente en la leyenda "*soy el hijo con id X*". Si, en cambio, el planificador es el de prioridades, se verá que los hijos no son ejecutados en el mismo orden. Esto se debe a que el planificador los seleccionará teniendo en cuenta la prioridad que se le asignó a cada uno al momento de ser creado.

- **Caso de prueba**: Generar un lote de varios procesos que queden huérfanos y ver su prioridad asignada.

  - **Objetivo**: Se pretende generar varios procesos en el sistema para verificar la correcta creación. Se espera que al ejecutar el comando ps se vean adicionados cinco procesos más a los que genera por defecto el Sodium al iniciarse, junto con su prioridad asignada.

  - **Prerrequisito**: Se debe establecer el planificador RR con colas de prioridades.

  - **Proceso**: Se ejecuta los siguientes comandos:
    - *usuario 2*
    - *ps*

  - **Salida**: Al ejecutar el primer comando se muestra un registro de lo que fue pasando dentro del proceso. El segundo comando es el importante para este caso, ya que muestra los procesos actuales, entre los cuales se encontrarán, en estado zombie, los procesos creados con el comando anterior junto con su prioridad asignada.

Conclusiones

Durante el desarrollo del nuevo planificador tuvimos como inconveniente que sobrepasamos el límite de memoria al cargar SODIUM con un nuevo comando, por lo que tuvimos que remover

algunos de test que se estaban cargando.
Otra complicación que hubo es que al programar en SODIUM uno se encuentra con menos herramientas en comparación con las que está acostumbrado al desarrollar en el mismo lenguaje pero sobre un sistema operativo más robusto (por ej., hay muchas menos funciones de c disponibles).
El trabajo práctico nos permitió aprender un poco más del funcionamiento de un planificador y fue interesante porque nos permitió además verlo en acción. Como esto es algo que siempre tenemos resuelto a la hora de utilizar cualquier sistema operativo, uno normalmente no le presta atención.