MINIKERNEL

Ampliación de Sistemas Operativos

Doble Grado en Ingeniería de Computadores y Diseño y Desarrollo de Videojuegos.

Universidad Rey Juan Carlos

Juan Pedro Guirado Sánchez
Ángel Luis Serrano González

Índice

Opciones de diseño utilizadas

Dormir

Lo único a comentar acerca de la función dormir es la necesidad de restar 1 *tick* a cada uno de los procesos bloqueados, cosa que al principio no implementamos de forma correcta y nos provocó una serie de fallos en el resto del código. Una vez su tiempo de dormir ha llegado a (0) los procesos son despertados volviéndose a introducir en la lista de listos con el cuidado manejo de las interrupciones para evitar acciones inesperadas.

Mutex

Las principales decisiones de diseño que hemos tenido que tomar se encuentran en el apartado de la creación, destrucción, apertura y cierre de *mutex*. Nos resultaba especialmente difícil mantener un seguimiento correcto del estado de los *mutex* durante la ejecución del programa.

Por lo tanto, a la hora de crear el tipo *mutex*, era necesaria la inclusión de una serie de variables con este fin. Para ello, creamos las siguientes variables en el tipo mutex:

- ***int is_bloqued***: Una variable booleana que nos permitía conocer en todo momento si dicho mutex había sido bloqueado por un proceso.
- ***int num_proceso_bloqueados***: Un contador, que nos permitía conocer cuántos procesos se encontraban bloqueados a la espera de que dicho mutex se liberara.
- ***lista_BCPs lista_espera***: La variable anterior y esta se encuentran estrechamente relacionadas puesto que es una lista donde permanecían los procesos que se encontraban bloqueados a la espera de dicho mutex.
- ***int id_proc***: Esta variable contiene el id del proceso que tenía en ese momento bloqueado al mutex.
- ***int isCreated***: Esta variable nos permitía conocer si dicho mutex había sido creado, y por lo tanto inicializado o no.
- ***int proc_abiertos***: que nos permitía conocer el número de procesos tenían abierto dicho mutex.

Nos resultó especialmente útil la inclusión de la variable *isCreated* puesto que los *mutex* dentro de la tabla de *mutex* podían estar o no inicializados y esto nos daba muchísimos problemas al intentar crear un *mutex* que ya había sido creado o al intentar abrir un *mutex* que no había sido creado correctamente.

También creemos, que una importante decisión fue la de mantener siempre una coherencia en el movimiento de procesos entre listas, especialmente en el manejo de la *lista_listos*, que como recordamos, contenía los procesos listos para la ejecución. Con coherencia, nos referimos a que siempre que se fueran a modificar dichas listas, el nivel de interrupción debía ser el máximo, puesto que, en caso contrario, cualquier interrupción podría dejarnos "colgando" un proceso sin meter en la lista.

En este sentido, creamos la función *cambio_proceso()*, la cual recibe como parámetro una lista, que será la lista donde se va a meter el proceso actual , eliminándolo de la lista de listos y llamando al planificador para cambiar el proceso actual.

Round Robin

Durante la implementación del algoritmo de planificación *Round Robin* nos percatamos de que había que distinguir entre dos casos principales, dependiendo de si sólo existía un proceso en ejecución y ninguno más en la lista de listos o si por el contrario sí que existían más procesos que estaban esperando para ejecutar. En el primer caso habría que renovar la rodaja del único proceso en ejecución hasta que este terminase o bien otro proceso entrara en la lista de listos esperando para ejecutar. Si por el contrario existiesen otros procesos esperando para ejecutar, al finalizar la rodaja del que se encuentra en ejecución habría que realizar un cambio de proceso y contexto. Esta distinción se encuentra implementada en la función *int_sw().* El cambio de proceso se encuentra implementado con la función auxiliar *cambio_proceso(),* de la cual ya hemos hablado anteriormente.

Manejo básico de la entrada por teclado

El principal problema de diseño al que nos enfrentamos en esta implementación es el de usar un array circular. Esto requería de una coordinación entre dos funciones auxiliares *leer_buffer* y *escribir_buffer*. Para lograr esta coordinación era necesario crear una estructura (*buffer_terminal_t)* con una serie de variables cuya función será la del correcto manejo de un array de caracteres. Por un lado, necesitamos un par de variables int que contengan los índices de donde se ha de leer u escribir. En segundo lugar, necesitamos otra variable int para llevar la cuenta de caracteres escritos; para que en caso de que sea igual al tamaño máximo del array, se empiece a escribir en la posición (0), al ser este de tipo circular. Por último, es necesario crear una lista de *BCPs* bloqueados, para almacenarlos en caso de que hayan intentado leer del buffer, pero este se encuentre aun vacío.

De esta manera cada vez que se llama a la interrupción de terminal, esta recoge el carácter y se lo pasará a la función *escribir_buffer*. Esta función se encargará de escribirlo en el array de caracteres de nuestra estructura y de actualizar correctamente las variables que controlan el índice donde escribir y el número de caracteres escritos. A su vez también se ocupará de desbloquear el proceso bloqueado en el caso de que el buffer ya no este vacío. Por otro lado la función *leer_buffer(),* llamada desde la función *sis_leer_caracter(),* se encargará de leer los caracteres escritos y de actualizar la posición a leer y el número de caracteres correctamente.

Comentarios personales

En primer lugar, creemos que esta práctica, pese a ser una de las prácticas más complicadas que hemos realizado en la carrera, es una de las prácticas que mejor nos ha hecho comprender los conceptos que trata, especialmente en el sentido de los mutex, puesto que son un mecanismo bastante complejo y que es necesario tener muy claro cómo estos funcionan a la hora de implementarlos correctamente.

Del mismo modo, y puesto que para nosotros el principal problema encontrado en esta práctica es no tener demasiado claro los conceptos básicos sobre los que se trabajan, creemos que sería conveniente un pequeño repaso de los conceptos que se piden en la antes de la realización de la misma, puesto que si bien, se dan en la asignatura Sistemas Operativos de segundo, tras 6 meses desde su realización, estos pueden estar un poco "olvidados" lo que dificulta aún más la misma.

Como ya hemos comentado, nos parece que la dificultad de la misma no corresponde con la nota de esta en la ponderación final de la asignatura (un 15%) siendo esta bastante escasa puesto que, en nuestro caso, la realización de la práctica nos ha llevado más de 12 horas de trabajo lo que puede resultar un tanto desalentador por 1,5 puntos a lo sumo.

Según nuestra opinión, el realizar una parte de la misma en forma de practica guiada por el profesor llevaría a una mejora de la comprensión inicial de la misma, ya que quizás el alumno pueda sentirse abrumado al empezar a trabajar con la practica y no comprender exactamente como funciona la gran cantidad de material auxiliar suministrado.

Por otro lado creemos que seria de gran utilidad el recomendar al alumno el uso de algún IDE como *Netbeans*, que en nuestro caso ha sido de una grandísima utilidad debido a que nos permitía depurar el código y ver en que fallaban nuestras ejecuciones, sobre todo en la parte de los Mutex, debido a su gran carga de lógica en el código de los mismos.