Inclusión de una llamada simple

Se debe añadir una nueva llamada que devuelve el identificador del proceso que le invoca.
Se trata de un servicio que sirve para familiarizarnos con el mecanismo que se usa para incluir una nueva llamada al sistema.
Simplemente retorna el id del proceso leído del BCP

Llamada que bloquea al proceso un plazo de tiempo

Se debe incluir una nueva llamada denominada sis_dormir que permita que un proceso pueda quedarse bloqueado un plazo de tiempo.
Dicho plazo se especifica en segundos, creando un campo nuevo en el registro BCP.
Cuando queremos dormir un proceso primeramente leemos el argumento del tiempo para dormir del registro de entrada el cual asignamos al campo del BCP creado anteriormente, mandamos el proceso a una nueva lista creada, que contiene todos los procesos dormidos en el sistema, y por último hacemos un cambio de contexto para que se ejecute el siguiente proceso. La operación de dormir un proceso se realiza en la llamada al sistema sis_dormir.

El despertar de un proceso depende de la rutina de interrupción de reloj, de forma que que cada vez que se activa int_reloj , se resta un 1 en el campo de tiempo dormido de cada proceso de la lista de procesos dormidos, si ocurre que un proceso ha llegado a tiempo dormido = 0, se le manda a la lista de procesos listos..

Mutex

Se deben implementar dos tipos de mutex:

- mutex no recursivos en el que si un proceso que posee un mutex intenta bloquearlo de nuevo se devuelve un error ya que se estaría produciendo un interbloqueo.
- mutex recursivos en los que cuando un proceso posee un mutex puede bloquearlo todas las veces que quiera, pero el mutex solo se desbloqueara cuando el proceso lo desbloquee tantas veces como lo bloqueó anteriormente.

Las funciones a implementar son las siguientes:

- Crear_Mutex: Esta función que se le pasa como parámetro de entrada el nombre y el tipo, crea un mutex y devuelve el descriptor del mutex si se cumple las condiciones para la creación del mutex. Sin embargo, si el contador de mutex es igual al número máximo de mutex se elimina el proceso de la lista de listos y se

inserta al final de la lista de proceso bloqueados por el mutex y se realiza un cambio de contexto.

- Abrir_mutex: Esta función que se le pasa como parámetro de entrada el nombre del mutex comprueba que se cumple todas las condiciones para abrir el mutex en caso de que si, busca el mutex en la lista de mutex y un descriptor libre en el array de descriptor del proceso para guardar el mutex y devuelve el descriptor.

- Lock: Esta función que tiene como parámetro de entrada mutex que es el índice del array de descriptor de proceso.. Esta llamada cambia el estado del mutex a bloqueado si estaba desbloqueado. También bloquea el proceso eliminado el proceso de la lista de procesos listo y insertando al final de la lista de procesos bloqueados por el lock si el estado del mutex es bloqueado. En el caso de que el mutex es recursivo se aumenta a mas 1 en el número de bloqueo pero si fuese no recursivo entonces devolvería un mensaje de error.

- Unlock: Esta función que le pasa como parámetro de entrada mutexid que el índice del array del descriptor de proceso. Esta llamada desbloquea el mutex por lo que también desbloquea todos los procesos bloqueados por este mutex, para ello con un bucle recorre la lista de proceso bloqueados por el lock y cada iteración compara el mutex de la lista con el mutex del índice de array de descriptor, si coincide entonces elimina de la lista de proceso bloqueados por lock y lo inserta al final de la l ista de procesos listos.

- Cerrar_mutex: Esta llamada que tiene como parámetro de entrada mutexid que es índice que nos indica donde está el mutex en el array de descriptores, ese mutex se desbloquea y se hace una búsqueda en la lista de procesos bloqueado por el mutex todos los procesos que fueron bloqueado por este mutex para eliminarlo de esa lista e insertarlo al final de la lista de procesos listos y borramos el mutex del array de d escriptores. También comprobamos si el número de proceso que tiene abierto el mutex es igual a cero si es así, elimina el mutex de la lista de mutex y desbloqueamos los procesos que estaban esperando que hubiera un mutex libre en el sistema.

Round - Robin

El algoritmo que tenemos primeramente en la planificación se trata de un algoritmo FIFO en el que el primero que entra es el primero que sale. En esta parte se pide sustituir la planificación FIFO por un algoritmo de planificación Round - Robin.
En dicho nuevo algoritmo a implementar tenemos un tamaño de rodaja dada por la constante “TTICKS_POR_RODAJA”.

Primeramente, se debe crear un campo nuevo en el BCP que registra el tiempo que tiene cada proceso. Además se debe añadir el código necesario en la rutina de interrupción de reloj para que se ejecute el algoritmo Round - Robin, dicho código simplemente resta una rodaja de tiempo a la variable que lleva la cuenta de cada proceso, y en caso de que

ese tiempo sea igual a 0, se activamos la subrutina de interrupción de software mediante la función activar_int_SW(());

Se debe modificar la rutina de int_sw ()), para que el proceso que ha agotado sus rodajas de tiempo se ponga al final de la lista, y de paso al proceso que ha quedad o primero en la lista. De esta forma nos aseguramos de que cada proceso esté en ejecución solo el tiempo establecido por la constante TICKS_POR_RODAJA .

Manejo básico de la entrada por teclado.

Se debe implementar la rutina sis_leer_caracter()

Para esta rutina se debe crear una nueva estructura en kernel.h declarada como struct buffer_terminal , dicho registro tiene un campo buff[], que es un array de caracteres con el tamaño de buffer preestablecido, tiene además un campo leer, que nos muestra la posición del buffer a leer, un campo escribir, y u n campo num_car , que contiene el número de caracteres en el buffer, dicho campo nos será útil para saber si el buffer está lleno, ya que nuestro buffer es circular,, y no podemos controlar el llenado con los índices..
Además, se debe declarar una lista de procesos bloqueados porque no pueden leer del buffer.
En la llamada sis_leer_caracter(), verificamos si el buffer está vacío, si es así inhabilitamos interrupciones e introducimos el proceso actual en la lista de procesos bloqueados por leer caracteres, y hacemos un cabio de contexto al proceso que está esperando. En caso de que el buffer no esté vacío, fijamos un nivel 2 de interrupción y llamamos a una función auxiliar leer_buffer, donde lee un carácter del buffer teniendo en cuenta detal les como que el buffer es circular y que hay que restar un 1 en la variable que controla la cantidad de caracteres en el buffer. Esta rutina por lo tanto, hace el papel de “consumidora”, y la rutina de interrupción de termina hace el papel de “productora” de caracteres..

En la rutina int_terminal(), verificamos que el buffer esté vacío, si es así desbloqueamos algún proceso de la lista de bloqueados por lectura de carácter, hacemos el cambio de contexto y usamos una función auxiliar llamada escribir_buffer(), donde escribe en el buffer teniendo en cuenta la actualización de índices de lectura y escritura y de la variable que controla el número de caracteres, además también de determinar si debe modificar el índice de escritura a la primera posición del array en caso de que se haya llegado al final del buffer