Trabajo Práctico 2: : Construcción del Núcleo de un Sistema Operativo y estructuras de administración de recursos.

Sistemas Operativos (2C2020)

Integrantes:
Franco Navarro (59055)
Joaquín Garcia del Rio (59051)
Lucia Digon (59030)
Santiago Preusche (59233)

Decisiones tomadas durante el desarrollo, por ejemplo, detalles del algoritmo de scheduling y administrador de memoria.

Physical Memory Manager

En nuestro TP llegamos a implementar el alocador buddy y free list con éxito.

En el caso del buddy allocator, generamos un árbol binario con una cantidad fija de nodos. Todos los nodos y el árbol en común se guardan en una posición de memoria libre que es distinta a la memoria que va a usar el usuario, pero que el pure64 garantiza que no está reservada para otra funcionalidad. En cuanto a su funcionalidad, cada nodo del buddy tree tiene un estado, el mismo puede ser ocupado, parcialmente ocupado o libre, la misma siendo una definición recursiva donde un nodo el libre si ambos nodos son libres, análogamente un nodo está ocupado si ambos de sus hijos están ocupados, y parcialmente ocupado en cualquier otro caso. Entonces, por ejemplo cuando el usuario hace un malloc, el buddy le da una dirección de memoria que a la vez está guardada en un bloque del tamaño adecuado y modifica el estado del mismo para que el mismo se figure como ocupado, actualizando el estado de todos los nodos del árbol luego de hacerlo.

En el caso del free list, la misma genera una lista de bloques libres y otra de usados. La primera comienza con un solo bloque que apunta a toda la memoria que el usuario puede usar y a medida que el usuario va realizando mallocs y frees ambas listas se van modificando. Una decisión que tomamos al momento de implementar free list fue almacenar la información de los bloques de memoria antes de los mismos, es decir continua a la memoria que se le otorga al usuario para que la modifique. En ese caso buddy fue un poco más eficiente, ya que reservamos todos los bloques fuera de esta sección de memoria. La razón por la cual hicimos esto fue que nos dimos cuenta después, cuando estábamos haciendo el buddy que eso iba a hacer una mejor forma de hacerlo, y por simplicidad y honestamente para no romper nada, lo dejamos así. No creemos que para el alcance de este TP sea necesario tanto espacio, por eso también decidimos que dejarlo así no sería un gran problema si luego era mencionado en el informe.

Procesos, Context Switching y Scheduling

El algoritmo que implementamos en nuestro trabajo práctico es el clásico Round Robin, utilizando las interrupciones del timer tick para llamar al scheduler cada vez que se interrumpe. Sin embargo, en nuestro caso, en vez de realizarse el cambio de contexto cada intervalos regulares, los mismos dependen de la prioridad del proceso que está ejecutando. Es decir, cuanta mayor sea la prioridad de un proceso, entonces más timer ticks va “durar” en ejecución. Creemos que esta forma es bastante práctica puesto que ningún proceso termina con starvation, debido a que es una lista cíclica y la lista no cambia de orden en ningún momento. Otra razón por la cual elegimos esta estrategia es porque nos parecía una alternativa más simple a la de cambiar la prioridad de otros procesos utilizando aging y modificando el orden de la lista.

Otra decisión que tomamos fue utilizar una sola lista para todos los procesos. Muchos eligen crear dos listas distintas, una para procesos ready y otra para procesos bloqueados, pero nosotros no vimos necesidad de hacerlo ya que el orden es cíclico y en vez sí un proceso está bloqueado solo lo ignoramos.

Inter Process Communication y Sincronización

Para la implementación de los semáforos, tomamos la implementación mostrada en clase, agregando algunas cosas. Mantuvimos la misma estructura de xchg, acquire y release provistos por la cátedra. Además optamos por agregar a la estructura del semáforo una lista con todos los procesos bloqueados por el mismo. Debido a esto, las funciones de sem_wait y sem_post se las tuvo que modificar para que cuando son llamadas hagan block o unblock de los procesos con los que trabajan y si es necesario agregarlos y sacarlos de la dicha lista. Y con el fin de mantener constancia de todos los semáforos activos del sistema, utilizamos una lista para guardarlos y así poder acceder a ellos, buscarlos y eliminarlos.

Por otro lado, en cuanto a la implementación de pipes, tuvimos como base lo provisto por la cátedra y también utilizamos una lista para mantener constancia de los pipes activos. Para su uso, tuvimos que agregar un parámetro en la creación de un proceso que es el pipe_id asociado. En caso de ejecutarse sin un pipe se le asigna un 0 a esta variable para así poder diferenciarlos. Además tuvimos que agregar una syscall para consultar este valor al estar corriendo un proceso. Otra medida que tuvimos que adoptar fue la de bloquear el segundo proceso (el que viene inmediatamente después de "|") hasta que el primero termine de ejecutarse y se mate, ya que recién ahí es correcto ir al pipe a leer lo que el otro proceso escribió.

Instrucciones de compilación y ejecución.

Para compilar se debe escribir "make all", y si es la primera vez compilando hay que hacer un "make clean". Puede pasar que haya alguna imagen en el directorio Image cuyos permisos tengan que ser modificados antes de ejecutar el .sh. Para lograrlo tiene que viajar al directorio y luego hacer un "chmod 777 [nombre de imagen]".

Pasos a seguir para demostrar el funcionamiento de cada uno de los requerimientos.

Physical Memory Manager

Al iniciar el sistema operativo, correr el comando "test_mm", el mismo no termina pero este fue un test otorgado por la cátedra para verificar si funciona. Se podrá verificar que funcionan tanto el buddy como el free_list allocator. Para poder cambiar entre ambos, hay que descomentar un comentario que dice "define BUDDY 3" en el archivo "memoryManager.c". Si se descomenta, se elige el buddy, sino el free_list. No llegamos a implementar esto via argumento de la terminal modificando el makefile porque nos quedamos sin tiempo.

Procesos, Context Switching y Scheduling

Al iniciar el sistema operativo, correr el comando "test_prio" o "test_processes". Ambos códigos provistos por la cátedra y modificados por nosotros.

Sincronización

Al iniciar el sistema operativo, correr el comando "test_sync" y "test_no_sync", fueron códigos provistos por la cátedra y modificados por nosotros. Ambos crean N procesos donde todos modifican una variable, la diferencia es que test_sync la modifica usando nuestros semáforos y test_no_sync no los usa. El resultado es que luego de todos los

incrementos, la variable termina dando un resultado que no tiene sentido en test_no_sync y uno correcto en test_sync.

Inter Process Communication

Al iniciar el sistema operativo, correr el comando “cat | filter”, o “cat | wc”. Esto verifica que existe una comunicación entre los procesos. Cat, como mencionado en la sección de limitaciones del TP no logramos hacer que se lea por entrada estándar. Asique hay que modificar un string en su código ubicado en “shellCommands.c”

Limitaciones.

Una de las limitaciones que presenta nuestro trabajo práctico es que la función loop, al ser infinita, no te deja escribir más debido a la forma que está hecha la shell.

Otra limitación es que el comando cat, no está implementado para recibir texto por entrada estándar y está puesto para que siempre corra con un texto ya definido. Por este motivo, siempre imprime lo mismo y las funciones filter y wc solo sirven para probarse de la forma “cat | wc” o “cat | filter” ya que no se implementó leer por terminal para recibir los parámetros de estas dos funciones .

Por otro lado, no se están matando automáticamente los pipes cuando ya se eliminan los procesos que la utilizan.

Por una cuestión de tiempo, el problema de los filósofos (aplicación phylo) no fue implementado.

Y finalmente, en cuanto a las evaluaciones de pvs-studio y cppcheck, el primero nos tira errores en la carpeta BMFS del bootloader y como es algo que nunca tocamos lo ignoramos, y con respecto al segundo este nos tira solo indicaciones de estilo y muchas de ellas son falsos positivos asi que tambien decidimos ignorarlas.

Problemas encontrados durante el desarrollo y cómo se solucionaron.

Hubo varios problemas con los cuales nos encontramos a lo largo de este TP. Principalmente hubo 3 ocasiones en las cuales tuvimos que estar noches enteras solucionando y viendo que estábamos haciendo mal.

En el caso del memory manager, tuvimos varios problemas implementando el buddy allocator y el free list. Sobre todo con el tema de las direcciones y generar los nodos en memoria, muchos problemas con casteos y mal uso de punteros a veces. Logramos solucionar muchos de estos problemas con prueba y error y revisitando mucha teoría. Además, al principio no estábamos testeando adecuadamente, en vez de utilizar los testeos provistos por la cátedra, estábamos creando los nuestros, que no eran muy útiles. Cuando nos pusimos a programar el scheduler recién ahí vimos que había problemas medio raros a la hora de reservar memoria y hacer frees y que seguramente eran problemas de memory manager. En ese momento fue cuando decidimos utilizar el testeo que la cátedra nos dió y pudimos ir corrigiendo bien los administradores de memoria y verificando que realmente funcionaban.

Otra parte que nos costó bastante fue el scheduler, especialmente la parte de manejo del stack: pusheando mal los registros, popeando mal. Lo que nos ayudó bastante a corregir

estos problemas fueron una de las clases grabadas que subieron y bastante tiempo debuggeando. El manejo de los distintos punteros fue lo que más nos costó de esta parte.

La última parte que nos costó un poco fue, en la parte de IPC, el tema de manejar la salida por entrada estándar o conectarlo al buffer del pipe.

Citas de fragmentos de código reutilizados de otras fuentes.

Los fragmentos de código para realizar el memcpy y strcmp fueron sacados de internet. Sin embargo, perdimos el enlace del cual fueron sacados.

Modificaciones realizadas a las aplicaciones de test provistas, en caso de que su sistema no soporte alguna funcionalidad.

En test sync, tuvimos que cambiar que nuestros procesos no los podemos crear y que reciban parámetros, por lo cual el n y el value los agregamos como variables de la función y además, creamos una función igual al inc llamada dec, que la única diferencia es que en lugar de ir sumando con 1, va sumando con -1.