Sistemas Operativos - TP 2

Integrantes:

- **Santiago Burgos, 55193.**
- **Facundo Astiz, 58333.**
- **Tomás Dallas, 56436.**

Decisiones tomadas durante el desarrollo

- Teníamos la idea de implementar un buddy memory allocator como memory manager. Por falta de tiempo y fallos en los intentos de implementaciones, se decidió desestimar y utilizar un memory manager rudimentario, con un array de páginas y procesos. (En la sección de limitaciones se especifica más técnicamente)
- Para el scheduler realizamos una cola circular en la que cada nodo tiene el proceso y el quantum, este se va decrementando y al llegar a 0 se pasa al siguiente proceso en la cola a no ser que se encuentre en estado bloqueado o eliminado en este caso se lo saltea. Se implementó una función yieldProcess para hacer un cambio de proceso forzado en el momento de realizar un bloqueo.
- Para los mensajes se utilizó una cola doblemente enlazada para cada uno de los procesos en la cual los nodos contienen un mensaje, su autor y su longitud. También, esta lista tiene un *header* en donde se encuentra el pid del dueño de la lista, y en el caso de estar bloqueado a la espera de un mensaje contiene también el pid del proceso por el cual esta bloqueado y la cantidad de caracteres que requiere leer. Se eligió implementarlo con una cola para facilitar el ordenamiento por orden de llegada de los mensajes.
- Para los mutexes se utilizó una estructura que tiene un vector que contiene a todos los procesos bloqueados esperando su turno y un valor que indica si el mutex se encuentra libre u ocupado. Cuando un proceso ejecuta un lock a un determinado mutex, si este se encuentra ocupado se agrega al vector de procesos bloqueados y se bloquea a sí mismo. En el caso del unlock se desbloquean todos los procesos y lo ocupara el que el scheduler ejecute primero. Los procesos no mueren de inanición dado que siguen el orden de la lista circular implementada en el kernel. De esta forma, el primero en ocuparlo será el último en volver a ocuparlo.

Instrucciones de compilación y ejecución

Se encuentran en :

https://github.com/tdallas/so-tp2/blob/develop/README.md

Limitaciones y problemas encontrados

Para el memory manager teníamos dos opciones, realizar un memory manager básico, para cumplir los requisitos suficientes de este tp, o ir más allá e implementar algo más arriesgado para cumplir con los requisitos del próximo tp también.
Decidimos encarar esta última opción y desarrollar un buddy memory allocator. Indagamos mucho sobre cuál era la mejor manera de implementarlo, y luego de investigar leyendo sobre el kernel de Linux y demás páginas web. Encontramos que el punto medio entre la manera más simple y eficiente era usando una lista circular doblemente encadenada para almacenar las zonas de memoria libres, y un array representando el heap/árbol que utiliza el buddy memory allocator. Lamentablemente, luego de 1 semana de intentar implementarlo funcionalmente no hubo resultados positivos, a pesar de que entendíamos bastante bien como funcionaba y las estructuras a utilizar, decidimos desestimar esta opción y dejarla para más adelante (entrega tp3) por los cortos tiempos que teníamos, tomando en cuenta que en el medio de la entrega había muchos parciales y demás tps que entregar. Así que implementamos un rudimentario memory allocator utilizando un stack de procesos y un stack de páginas. Pusimos arbitrariamente un máximo de 16 procesos de 64 mb, utilizamos páginas de 64KB y un máximo de 1024 páginas. Utilizamos arbitrariamente una dirección de memoria de comienzo que ya se utilizaba en el tp de arqui.

En el caso de los ipcs se dificulto el momento de realizar un bloqueo y cambiar inmediatamente al siguiente proceso. Siendo el principal motivo de la complicación el retorno a la syscall que causó el bloqueo. Por ejemplo en el caso de los mensajes, cuando se bloqueaba en el momento de leer mensaje necesitábamos de cambiar al siguiente proceso para luego, cuando se desbloqueara, continuar con la syscall en el lugar donde se dejó y retornar al userland. Esto se soluciono con una interrupción que fuerza al scheduler a cambiar de proceso y guardar el contexto del momento que se la llamó desde el kernel.

En general, creemos que el tiempo que tuvimos durante estas semanas que duró el tp no nos alcanzó, perdimos mucho tiempo

investigando/intentando implementar el buddy memory allocator, perdimos tambien unas semanas adaptando el tp viejo de arqui también para este tp, lo que resultó en pocas semanas de trabajo constante sobre las funcionalidades nuevas, esas pocas semanas sumado a todos los parciales y los demás tps nos complicó bastante el correcto desarrollo del mismo. Hubiéramos preferido entregar un tp con más funcionalidades y mejor testeo, pero creemos que implementamos lo mejor que pudimos y nuestro sistema cumple casi todos los *“caminos felices”*.

Tuvimos, además, varias complicaciones con C a la hora de debuggear el sistema, lo que nos hizo el proceso de debugging más arduo y trabajoso.