Sistemas Operativos (72.11) - 1er cuatrimestre 2024

Trabajo Práctico 2:
Informe
Grupo 7

Integrantes:
Federico Agustín Etchegorry (63175) , fetchegorry@itba.edu.ar
Franco Morroni (60417) , fmorroni@itba.edu.ar
Bernardo Zapico (62318) , bzapico@itba.edu.ar

10 de julio de 2024

Agregamos acá el último commit y su rama correspondiente

- Link: **https://github.com/fmorroni/TP-Arqui.git**
- Commit **4c826382fa5fd4ed79f96ef5d21f94e5a316d6eb**
- Branch: **Main**

1. Decisiones tomadas durante el desarrollo

Para el desarrollo del tp tomamos la siguiente lista de decisiones:

- Memory Manager: se empieza alineando la dirección donde empieza nuestra memoria libre para empezar a reservar memoria. Se decidió realizar dos implementaciones (se puede alternar entre ambas comentando la constante **BUDDY** en *root/Kernel/include/memoryManager.h*):
  - Una de ellas fue el Buddy System, se utilizó 2 referencias de inspiración agregadas en la sección de referencias. Se maneja con un sistema de bloques dónde la información de cada bloque es guardada justo al inicio del bloque, tras esta, se encuentra la memoria útil que usará el usuario al reservar un bloque. Esta memoria del bloque es tenida en cuenta al momento de buscar un orden para reservar. Se busca el orden que necesitamos para almacenar tanto el espacio solicitado por el usuario como el tamaño del bloque, si no hay bloque libre del orden que necesitamos, buscamos uno de orden mayor para dividirlo hasta que tenga el orden que necesitamos, si no hay de orden mayor tampoco, devolvemos NULL. Luego para el free, además de liberar el bloque, tenemos en cuenta que se fusione con su buddy del orden actual de ese bloque si dicho buddy está libre, de fusionarse, se controla si el bloque se puede fusionar con su buddy de orden + 1 y así recursivamente.
  - La otra implementación fue una FreeList, puntualmente la misma que se presenta en FreeRTOS heap4. También se maneja con bloques, aunque estos se encuentran en una lista enlazada. Esta implementación toma el espacio solicitado por el usuario, le suma el tamaño del bloque y la alínea al próximo múltiplo del ancho de palabra (8 bits) para asegurar la eficiencia y busca en la lista enlazada por el próximo bloque libre que tenga memoria suficiente para el usuario y, si puede, lo divide en dos bloques, uno con la memoria justa para el usuario y otro libre para futuras reservas, reduciendo la fragmentación externa. Para liberar simplemente se lo agrega a la lista, al agregar un bloque a la lista de bloques libres se verifica si este debe fusionarse con algún otro bloque adyacente.

  Estas implementaciones luego se adaptaron para que un proceso pueda tener un heap propio, permitiendo liberar la memoria del mismo cuando este finaliza, independientemente de si el proceso hizo los free adecuados o no.
- Scheduling: decidimos implementar un Round-Robin con prioridades. Nuestra idea es tener una lista de los procesos usando nodos PCB dentro de los

cuales tenemos toda la información necesaria sobre los procesos, incluyendo la dirección del inicio del heap propio y los pipes. La prioridad del proceso se definió como la cantidad de quantums seguidos que se ejecuta el mismo. El estado de un proceso puede ser uno de los siguietnes: "*Ready*", "*Running*", "*Blocked*", "*Blocked by user*", "*Waiting for exit*" y "Exited". "Waiting for exit" es el estado asignado a un proceso que se intentó finalizar estando en estado "*Blocked*", y *tiene* el objetivo de que no se elimine el PCB del proceso antes de que el "dispositivo" que lo bloqueó lo termine de usar. Por otra parte, distinguimos el estado "*Blocked by user*", que se asigna cuando el usuario bloquea un proceso mediante la syscall *sysBlock,* ya que en este caso el proceso sí puede ser eliminado sin problema.

- Semaphores: se pueden inicializar semáforos con nombre (createSemaphore) o sin nombre (semInit), ambos retornan el id del semáforo, el cual se usa para hacer wait y post. También se puede usar la función *openSemaphore* para conseguir el id de un semáforo ya creado a partir de su nombre, o crearlo e inicializarlo con cierto valor si este no existe. El id de los semáforos es un entero equivalente a la posición del mismo en un array global de semáforos, con lo cual las operaciones son en orden 1. Las syscalls sys*PostSem* y *sysWaitSem* permiten incrementar o decrementar en 1 el valor de un dado semáforo. En caso de que al decrementar el valor se encuentra en 0, se procede a encolar a el PCB del proceso en una lista FIFO (y luego de agregarlo se bloquea el proceso) para que al momento de realizar el post se desencole un proceso en vez de incrementar el contador (y se coloca ese proceso en ready). Dentro del wait y el post se encuentran las funciones *_enter_region* y *_leave_region* (en Assembler) que controlan un spin-lock para acceder al valor del semáforo. En *_enter_region* se tomó la implementación de la cátedra a la que se le agregó un llamado a la int 22h (un yield) en caso de que el lock esté siendo ocupado por otro proceso antes de regresar a *_enter_region* nuevamente. De esta manera se evita el busy-waiting. Para destruir el semáforo se puede invocar *destroySemaphoreByName* (con el nombre del semáforo como parámetro) o *destroySemaphore* (con el identificador). No disponemos de un *close* dado que los procesos no necesitan "cerrar" el semáforo, solo destruirlo en caso de ser el único proceso utilizándolo.
- Pipes (usar el símbolo "!" en lugar de "|"): se planteó el caso productor-consumidor para controlar mediante semáforos los accesos a escritura y lectura del buffer de un dado pipe. Al igual que con semáforos, los pipes son identificados por un id que representa su posición en un array de pipes. A cada proceso se le agregó un file descriptor de lectura, escritura y error (este último no utilizado), los cuales en casi todos los casos indican el id del pipe con el que se comunicaran al hacer *sysRead* o *sysWrite*.
  Sin embargo, el file descriptor 0 (asignado por defecto como file descriptor de escritura para los procesos creados con *sysCreateProcess*) corresponde a

stdout, el cual no es un pipe. Cuando un proceso escribe a stdout se omiten los pipes y se imprime directamente en pantalla, sin almacenarse los caracteres en ningún buffer.
Por otro lado, el stdin (fd 1) es un pipe al cual solo escribe el handler de teclado mediante una función de escritura especial no bloqueante, ya que al no ser un proceso no puede ser bloqueado.
Otra aclaración relevante es que cualquier proceso puede escribir y/o leer de cualquier pipe si conoce su id.

2. Instrucciones de compilación y ejecución

1- Requisitos

Para el funcionamiento del kernel se le pedirá al usuario que se encuentre en un entorno de linux, que tenga el administrador de contenedores docker y el emulador de sistemas informáticos QEMU.

2- Preparando el entorno

Una vez instalado todo el software, se le pedirá al usuario descargar una imagen provista por la cátedra para el docker, y crear un contenedor en la carpeta con el código fuente con el nombre “so_builder”. Se puede hacer esto corriendo los siguientes comandos:

```
docker pull agodio/itba-so:2.0
```

```
docker run -d -v ${PWD}:/root --privileged -ti --name so_builder agodio/itba-so:2.0
```

Luego compilar el proyecto con:

```
./compile.sh
```

y finalmente correr el proyecto con:

```
sudo ./run.sh
```

Con eso ya tendremos acceso a la consola de nuestro sistema.

3. Hotkeys

Nota: Los keybinds se representan como <modkey-char>, donde modkey puede ser:

- c : ctrl
- m : alt
- s : shift

y char puede ser cualquier carácter representable.

- Generales:
  - F1 : Saca captura de los valores actuales del registro
  - <c-c> : Mata al proceso actual en foreground
  - <c-d> : Envía EOF
- De consola:
  - TAB : Autocompleta el comando (solo si hay un único comando para completar)
  - <c-l> : Limpia la pantalla
  - <c-k> : Retroceder en el historial de comandos
  - <c-j> : Avanzar en el historial de comandos
  - <c-w> : Borra una palabra
  - <c-+> : Agranda la font (Solo en shell)
  - <c--> : Achica la font (Solo en shell)

4. Pasos para testear el funcionamiento de requerimientos

- Observación, con “&” luego del último parámetro se puede correr un proceso en background.
- Para poder activar y desactivar el modo BUDDY para el memory manager es necesario ir al archivo memoryManager.h y comentar el “#Define BUDDY”. Nota: Hacer make clean y volver a compilar para que los cambios se hagan efectivos.
  - Para testear el memory manager alcanza con correr el comando “testMM <número de KB a utilizar>”.
  - Para consultar el estado de la memoria se necesita correr el comando “mem”.
- Para testear los procesos de Scheduling
  - Para crear un proceso se puede correr un comando como ps.
  - Para bloquear, desbloquear y eliminar procesos se pueden usar “block”, “unblock” y “kill” cada uno con el pid del proceso sobre el que se quiere efectuar la acción.
  - Con “ps se pueden ver los estados de los procesos

    - Con “nice <pid>” <prioridad entre 1-9> se puede cambiar la prioridad de un proceso.
    - Con “loop <seg>” se imprime un saludo cada los <seg> enviados como parámetro.
    - on “testProcesses <cantidad de procesos><imprimir cambios de estado><imprimir ps>” se puede realizar el testeo “test_processes” entregado por la cátedra. De igual forma con testPriority<0: espera breve, 1: espera larga> se puede testear la asignación de prioridades del proceso. Según el parámetro la espera será corta o larga entre procesos.
- Para testear Sincronización
    - Con “testSem <cantidad de iteraciones por proceso><1 para usar el semáforo, 0 para no usarlo><valor objetivo para el resultado final luego de las incrementos y decrementos>” se puede realizar el testeo de la cátedra “test_synchro”.
- Inter Process Communication
    - Cada proceso cuenta con un file descriptor de read y otro de write, para conectarlos con el pipe desde la shell usar el “ ! ”. OJO: no usar “ | ”. Mirar “**limitaciones**” antes de usar los comandos para Inter Process Communication.
    - Con “cat” se imprime el stdin tal como lo recibe
    - Con “wc” se puede contar la cantidad de palabras y líneas del input.
    - Con “filter” podés filtrar las vocales del input. Un ejemplo es “echo vocaleees ! filter”
    - Con “phylo” se imprime en pantalla el problema de los filósofos comensales. Podés agregar un filósofo con “a”, quitar un filósofo con “r” y terminar el proceso con “e”.
- Otras funcionalidades:
    - “help [nro página]” para ver los comandos disponibles.

5. Limitaciones

Nuestro programa tiene las siguientes limitaciones:

- Al imprimir a stdout si se usa el carácter ‘\b’ se pueden borrar caracteres previos al prompt actual.
- Un dado proceso puede tener como máximo 10 procesos esperando por él.
- El tamaño máximo de heap del sistema es de 64MB y el de un proceso individual es de 64KB útiles (en buddy terminan ocupando el doble

físicamente debido a que 64KB es potencia de 2 y al sumarle la estructura de información, termina cayendo en el próximo orden).
- Debido a la implementación de syscalls, las mismas pueden tener como máximo 5 argumentos.
- El estado de “Waiting for exit” fue agregado para poder mantener una implementación donde los procesos se acceden directamente mediante punteros a sus PCB en vez de usar el pid y tener que recorrer toda la lista de procesos, lo cual sería más ineficiente. Sin embargo, esta implementación nos deja con la limitación de que cuando un proceso está bloqueado por un semáforo y este semáforo quedó sin procesos que le hagan post, si se le hace un kill al proceso bloqueado este quedará en estado de “Waiting for exit” permanentemente, ya que el semáforo no recorrerá la lista de procesos bloqueados si no hay nadie que le haga un post.

6. Problemas encontrados durante el desarrollo

A lo largo del desarrollo nos encontramos con los siguientes problemas:

- Para la implementación de semáforos surgió el problema de que era necesario conseguir un identificador para unir procesos no conectados. En un principio pensamos en usar números pero vimos que tendía a ser problemático en muchos casos, por lo que decidimos usar strings, que aunque son ineficientes son más fáciles de identificar al realizar los programas. Sin embargo, se usaron semáforos anónimos en los casos en que era posible para mejorar la eficiencia del código
- Para la implementación de la freeList en el memory manager hubo bastantes dificultades adaptando el código de FreeRTOS a nuestro caso pues este realizaba muchas validaciones, asserts, aumento de variables para estadísticas, llamado a variables en otros archivos y funciones que no necesitábamos. Cómo primer contacto con un memory manager tras el dummy que vimos en la clase práctica fue bastante engorroso, por lo que optamos por primero realizar el buddy y luego hacer esta freeList.
- Hubo algunas dificultades para lograr implementar los procesos en foreground y background, especialmente para matar a los procesos con ctrl + c. Al final optamos por un puntero al PCB (process control block) actual que estuviera en foreground, y creamos una función que matara a este proceso y luego pusiera a su parent PCB en foreground. Este puntero al PCB actual en foreground también sirvió para la implementación del comando PS.

- Debido a la implementación original del driver de video y teclado, fue problemático adoptar syscalls de read y write bloqueantes.

7. Citas de fragmentos de código reutilizados de otras fuentes

A continuación adjuntamos páginas que hemos utilizado de referencia durante el trabajo, ya sea tomando una parte de su código y modificándolo o usándolo de inspiración:

- Clases prácticas de la materia y sus presentaciones.
- Consulta foro SO (T3 - Ejercicio 2 - Filósofos) (20/4/2024)
- https://freertos.org/a00111.html#heap_4
- https://www.geeksforgeeks.org/buddy-memory-allocation-program-set-1-allocation/
- https://github.com/evanw/buddy-malloc
- https://greenteapress.com/thinkos/html/thinkos013.html

8. Revisión de errores con PVS-STUDIO y GCC

Al compilar con GCC y -Wall, obtenemos el siguiente warning:

```
c/videoDriver.c:88:17: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
88 | framebuffer = (RGBColor*)VBE_mode_info->framebuffer;
```

Pero esto es una limitación sobre cómo se construyó el *vbe_mode_info_structure* ya que en el mismo el atributo *framebuffer* es un puntero al buffer de video, pero el mismo fue declarado de 32bits, con lo cuál al castear a *RGBColor** tenemos el warning por tratar de pasar de un entero de 32bits a un pointer (64 bits). Pero como pasamos de un tipo más chico a uno más grande en realidad no es problema.

En el chequeo con PVS-STUDIO decidimos ignorar los warning en el bootloader y ToolChain pues no son nuestros. El resto arrojó lo siguiente:

1. `root/Kernel/c/kernel.c 25 note V566 The integer constant is converted to pointer. Possibly an error or a bad coding style: (EntryPoint) 0x400000`

```
2. /root/Kernel/c/kernel.c 26 note V566 The integer constant is converted to pointer. Possibly an error or a bad coding style: (EntryPoint) 0x500000
3. /root/Kernel/c/naiveConsole.c 8 note V566 The integer constant is converted to pointer. Possibly an error or a bad coding style: (uint8_t *) 0xB8000
4. /root/Kernel/c/naiveConsole.c 9 note V566 The integer constant is converted to pointer. Possibly an error or a bad coding style: (uint8_t *) 0xB8000
5. /root/Kernel/c/naiveConsole.c 13 note V566 The integer constant is converted to pointer. Possibly an error or a bad coding style: (uint8_t *) 0xb8000
```

Es entendible que al pvs studio no le gusta este estilo pero a nivel “kernel” lo necesitamos para saber las posiciones dónde imprimir en pantalla o dónde está kernel/userland.

```
6. /root/Kernel/c/pipes.c 29 note V656 Variables 'stdinPipe', 'stderrPipe' are initialized through the call to the same function. It's probably an error or un-optimized code. Check lines: 28, 29.
```

Los llamados son iguales pero dan resultados distintos, el pipeInit() lo crea y guarda en el array. Luego el Array_get obtiene el resultado y lo guarda en su respectiva variable.

```
7. /root/Kernel/c/semaphores.c 87 note V522 There might be dereferencing of a potential null pointer 'toReady'.
```

En estas líneas se hace primero un chequeo de que el primer proceso de la cola no fuera null y después se realiza un unqueue. Por lo que nunca serán NULL.

```
8. /root/Userland/UserModule/c2/shellCommands.c 176 err V609 Divide by zero.
```

Es un positivo intencional, la idea es que al correr este comando se arroje una excepción de “Zero division”.

9. /root/Userland/UserModule/c2/array.c 31 warn V1004 The 'a' pointer was used unsafely after it was verified against nullptr. Check lines: 30, 31.
10. /root/Userland/UserModule/c2/array.c 47 warn V1004 The 'a' pointer was used unsafely after it was verified against nullptr. Check lines: 45, 47.
11. /root/Userland/UserModule/c2/array.c 59 warn V1004 The 'a' pointer was used unsafely after it was verified against nullptr. Check lines: 55, 59.
12. **Muchos warnings** de este tipo en **array.c.**
13. /root/Userland/UserModule/c2/circularHistoryBuffer.c 28 warn V1004 The 'cb' pointer was used unsafely after it was verified against nullptr. Check lines: 27, 28.
14. /root/Userland/UserModule/c2/circularHistoryBuffer.c 41 warn V1004 The 'cb' pointer was used unsafely after it was verified against nullptr. Check lines: 40, 41.
15. /root/Userland/UserModule/c2/circularHistoryBuffer.c 49 warn V1004 The 'cb' pointer was used unsafely after it was verified against nullptr. Check lines: 48, 49.
16. **Muchos warnings** de este tipo en **circularHistoryBuffer.c**
17. /root/Userland/UserModule/c2/shellCommands.c 221 warn V1004 The 'memState' pointer was used unsafely after it was verified against nullptr. Check lines: 216, 221.

Entendemos que los errores de file pointer que arroja falsos positivos de acuerdo al manual de PSV (nosotros usamos funciones de exit personalizadas). Por lo que no tuvimos en cuenta esta caracterización:

“The analyzer may issue a false warning in the following case”:

```
if (p == nullptr){
  MyExit(); }
*p += 42;
```

https://pvs-studio.com/en/docs/warnings/v1004/

9. Modificaciones realizadas a las aplicaciones de test provistas

No se realizaron cambios significativos a los testeos ofrecidos por la cátedra más allá del cambio de nombre de variables y la forma de creación de procesos que es diferente. Por ejemplo: pid = sysCreateProcess(1, argvAux, endless_loop). También agregamos opciones para TestPriority con el fin de poder ver más información mientras se ejecuta el test.