72.11 Sistemas operativos - 2do Cuatrimestre 2023

INTER PROCESS COMMUNICATION

Grupo 13

Rocio D´Intino - 62341 - rdintino@itba.edu.ar

Brian D´Arco - 56413 - bdarco@itba.edu.ar

Matías Ignacio Luchetti - 62337 - mluchetti@itba.edu.ar

Índice

Introducción

El siguiente trabajo tiene como principal objetivo el manejo de diversos mecanismos de IPC, manejo de memoria, procesos, scheduling y sincronización. Lo anteriormente mencionado se logró a través del uso de un kernel, cuya base se obtuvo del TPE Arquitectura de Computadoras.

En primer lugar se especificarán las instrucciones de compilación y ejecución. Asimismo, se detallarán los comandos implementados, al igual que su funcionamiento e interacción. En adición, se mencionarán tanto las limitaciones del trabajo como las decisiones tomadas a lo largo del mismo. Por último, se explicarán aquellas modificaciones realizadas a las aplicaciones de test provistas por la cátedra.

Instrucciones de compilación y ejecución

1. Ejecutar la imagen de Docker provista por la cátedra:
   `docker run -v ${PWD}:/root --security-opt seccomp:unconfined -ti agodio/itba-so:2.0`
2. Situarse dentro de la carpeta *./x64barebones/Toolchain* y ejecutar el comando `make all`
3. Regresar a la raíz */root/x64barebones* y ejecutar el comando `make all`
4. Tras compilar, correr el script run.sh dentro de la carpeta *x64Barebones* con el comando `./run.sh`
5. Para limpiar los archivos binarios ejecutar dentro de la carpeta *x64Barebones* el comando `make clean`

Observación: por default el trabajo se compila con el Memory Manager propio. Si se desea compilar con el Buddy system asegurarse que en la carpeta *~/x64Barebones/Kernel/Makefile* se encuentre descomentada la línea de 'USE_BUDDY'.

Comandos implementados

Al iniciar la *shell*, es posible correr los siguientes comandos:

- help: No recibe argumentos. Imprime los comandos y su funcionamiento.
- divzero: No recibe argumentos. Genera una excepción del estilo división por cero.
- invopcode: No recibe argumentos. Genera una excepción del estilo código de operación invalido.
- time : No recibe argumentos. Imprime la hora.
- pong: No recibe argumentos. Ejecuta el juego de pong para dos jugadores.
- inforeg: No recibe argumentos pero se debe tomar una "screenshot" (CTRL + R) para así imprimir los registros en dicho momento.
- clear: No recibe argumentos. Limpia la shell.
- testMm: No recibe argumentos. Ejecuta el test de memoria provisto por la cátedra.
- testProcesses: Recibe un argumento representando la cantidad máxima de procesos a crear. Ejecuta el test de procesos provisto por la cátedra.
- testPriorities: No recibe argumentos. Ejecuta el test de prioridades provisto por la cátedra.
- cat: No recibe argumentos. Imprime el stdin como lo recibe. Espera un señal EOF (end of file) para terminar.
- loop: No recibe argumentos. Imprime un saludo con su PID.
- wc: No recibe argumentos. Cuenta las líneas ("new lines") del stdin, e imprime su resultado al recibir una señal EOF.
- filter: No recibe argumentos. Imprime el stdin filtrando las vocales.
- kill: Recibe un argumento: un PID. Mata el proceso vinculado con dicho PID.
- ps: No recibe argumentos. Imprime todos los procesos que se están corriendo seguido de la información correspondiente.
- phylo: No recibe argumentos. Ejecuta el problema de los filósofos.
- nice: Recibe dos argumentos: un PID y un número, provocando un cambio de prioridad del proceso vinculado con ese PID.
- block: Dado un PID cambia su estado a bloqueado si no lo estaba, y viceversa.
- mem: No recibe argumentos. Imprime el estado de la memoria.

Observación: al presionar las teclas CTRL+D envía un EOF, mientras que CTRL+C las mata directamente volviendo a la shell.

Decisiones tomadas durante el desarrollo del trabajo

1. Scheduling:

El algoritmo implementado por el scheduler es el "priority-based round robin". Las prioridades oscilan entre 1 y 5, representando la cantidad de timer-ticks asignados a cada proceso. La implementación incluye una variable estática que cuenta los ticks desde el último cambio de contexto. Cuando este valor coincide con la cantidad de ticks asignados al proceso en ejecución, se produce un cambio de contexto y el contador se reinicia. De este modo, es posible modificar instantáneamente la prioridad de un proceso.

En cuanto a la clasificación de procesos, se distinguen entre mortales e inmortales. Los procesos inmortales, como el "idle" y la "shell", no pueden ser detenidos o pausados por el usuario, mientras que los procesos mortales pueden ser interrumpidos mediante la combinación de teclas CTRL+C. Se ha implementado la gestión de procesos hijos que están vinculados a sus padres a través de una tabla, permitiendo que el padre espere a que todos sus hijos terminen antes de continuar con su ejecución.

Adicionalmente, se ha incorporado una funcionalidad para ejecutar procesos en segundo plano desde la terminal mediante la adición del carácter "&" después del comando deseado.

2. Semáforos:

Con el objetivo de evitar la inanición, se ha implementado una cola de procesos bloqueados. Esta estructura garantiza que ningún proceso retenga el control del semáforo de manera prolongada, ya que, al regresar a la cola, se brinda la oportunidad a procesos en espera de tomar el semáforo. Si un semáforo está en 0 y un proceso ejecuta la operación signal, y existe al menos un proceso bloqueado esperando dicho signal, entonces el valor del semáforo no se modificará. En lugar de ello, se omitirá este paso y simplemente se despertará al proceso bloqueado.

3. Memory Manager:

Para la implementación del memory manager, se adoptó un enfoque donde cada bloque asignado cuenta con un encabezado de 8 bytes. El bit menos significativo indica si el bloque está asignado o no, y el resto de los bytes se utilizan para almacenar el tamaño del bloque. En la navegación de la lista, se procede deteniéndose en los encabezados, consultando el tamaño y saltando esa cantidad de bytes hacia adelante, donde se encuentra otro encabezado. Este proceso se repite hasta alcanzar el final de la lista, marcado por un bloque denominado "EOL" (End Of List), que posee un tamaño de 0 y se indica como

asignado. El "EOL" se desplaza a medida que la lista crece. Es relevante mencionar que, al liberar un bloque de memoria que tiene otro bloque libre delante, se realiza una fusión de dichos bloques para reducir la fragmentación interna.

En relación con las implementaciones del buddy system, se eligió una estrategia que utiliza un árbol binario para registrar qué bloques están ocupados. Se reserva una porción del heap donde la información de cada bloque se almacena en un nodo TNode. Cada nodo contiene dos datos: uno que indica si el bloque está ocupado y otro que indica si está dividido. Utilizando esta información, se realiza un mapeo con el heap y se devuelve el puntero correspondiente al usuario. Es fundamental destacar que en la zona de memoria devuelta al usuario se incluye en línea un encabezado con el índice del nodo al que pertenece dicho bloque.

Modificaciones realizadas a las aplicaciones de test provistas por la cátedra

1. testMm

   Se ha modificado la función para que sea de tipo void y no reciba parámetros. En el código, se asigna el valor numérico directamente, en lugar de recibirlo como parámetro. Para ejecutarlo, utiliza el comando "testMm" en la línea de comandos.
2. testPriorities

   Los nombres de las funciones se han ajustado para que sean compatibles con el sistema operativo. Para invocarlo, utiliza el comando "test-prio" en la línea de comandos.
3. testProcesses

   Las funciones se han adaptado para que sean compatibles con el sistema operativo. Para ejecutarlo, utiliza el comando "testProcesses" en la línea de comandos. Este programa ahora requiere un parámetro obligatorio que indica la cantidad máxima de procesos a crear.
4. testSync

En la implementación de este test, se ha decidido adaptar el código proporcionado por la cátedra. El test ahora recibe un parámetro: si es 0, el test no utilizará semáforos; si es diferente de 0, sí los utilizará. A diferencia del código original de la cátedra, el semáforo se crea en el propio testSync si el usuario lo solicita. Luego, se crean los procesos hijos, cuya cantidad está definida por una constante llamada TOTAL_PAIR_PROCESSES. Estos procesos hijos son los encargados de incrementar la variable global. Además, si deseas cambiar la cantidad de procesos que incrementan la variable global o el valor por el que se incrementa, debes modificar las constantes TOTAL_PAIR_PROCESSES o la variable inc en

la función myProcessInc, respectivamente. Finalmente, al concluir el test, se imprime una cadena que muestra el valor esperado de la variable global, calculado como TOTAL_PAIR_PROCESSES * MAX, y el valor obtenido en la variable global. Para ejecutarlo, utiliza el comando "testSync" en la línea de comandos.

Problemas encontrados durante el desarrollo del trabajo

El principal desafío que enfrentamos fue el siguiente: experimentamos dificultades en el traslado de parámetros desde la shell hacia las funciones de los comandos. Por ejemplo, al intentar llamar a "kill 3", dicho valor numérico "3" no se transmitía correctamente a la función "kill". Este problema surgía debido a que nos encontrábamos leyendo una posición incorrecta del stack.

Limitaciones

En primer lugar, el scheduler presenta ciertas limitaciones, permitiendo la creación de hasta 20 procesos, cada uno con un stack de 4096 bytes. Respecto a los pipes, se ha decidido imponer un tope de 20 pipes, y en términos de capacidad de almacenamiento, se ha fijado un máximo de 1024 bytes por pipe. En cuanto a los semáforos, se ha establecido un límite de 50 semáforos, con la capacidad de bloquear hasta 20 procesos cada uno. En el caso específico de Phylo, se ha establecido un límite de 15 filósofos. Esta restricción se ha definido considerando que se pueden tener hasta 20 procesos en ejecución simultánea, como se mencionó previamente.