Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

CI-0122 Sistemas Operativos
Grupo 01
I Semestre

III Tarea programada: [Caching: TLB y memoria virtual]

Profesor:
Francisco Arroyo

Estudiantes:
Luis Esteban Ramírez Arroyo | B76144
Esteban González Ureña | B73404

12 de julio del 2019

Índice

1. Introducción

La tarea programada consiste en simular el uso de almacenamientos secudarios, por parte del sistema operativo, así mismo, la utilización de la memoria virtual, para evitar los errores de agotamiento de memoria. Se implementó la excepción de los Page Fault (saltos de página), y estos suceden al trasladar la memoria virutal a la física, por medio del caché de nachOS.

Esta tarea nos facilita el observar el manejo de memoria del sistema operativo al ejecutar programas que abarcan más memoria de la que el sistema soporta.

2. Objetivos

- Implantar el TLB administrado por software.
- Implantar memoria virtual.
- Evaluar el desempeño de su sistema.
- Escribir un conjunto de programas de usuario "útiles"para demostrar fallos de TLB y páginas.

3. Descripción

- En la tercera fase de Nachos se investigará el uso de caching que se utilizará en esta tarea con dos propósitos:
  - Primero, se va a emplear un traductor de direcciones basado en software, Translation Lookaside Buffer (TLB), como una memoria cache para la tabla de páginas, con el fin de dar la ilusión de un acceso rápido al traductor de direcciones virtuales que funcionaría en un espacio de direcciones muy grande.
  - Además, se va a utilizar el disco como parte de la memoria principal, con el fin de proveer un espacio de direccionamiento (casi) ilimitado, con el desempeño cercano al de la memoria física.
  - Para esta asignación no se provee código nuevo, el único cambio necesario de compilar el código ya existente (threads y userprog) con las banderas DVM y $DUSE_TLB$), incluidas dentro del Makefile del proyecto "vm"(-DVM y -$DUSE_TLB$); la tarea será escribir el código para manejar el TLB e implantar la memoria virtual.
  - Para esta fase, el hardware no sabe nada de las tablas de páginas. En su lugar, solo trabaja con un cache, cargado por software (el sistema operativo) de las entradas de las tablas de páginas, denominado TLB.
  - En casi todas las arquitecturas modernas se incorpora un TLB a fin de aligerar las traducciones del espacio de direcciones.
    - Dada una dirección de memoria (una instrucción que se desea buscar o un dato para cargar o guardar), el procesador primero busca en el TLB para determinar si el mapeo de la dirección virtual a la dirección física ya se conoce.
    - Si es así entonces la traducción puede efectuarse rápidamente.
    - Pero si el mapeo no se encuentra en el TLB (fallo TLB) entonces se deben acceder las tablas de páginas y/o de segmentos.
    - En muchas arquitecturas, entre ellas Nachos, un fallo TLB causa una trampa al kernel del sistema operativo, que se encarga de efectuar la traducción, carga la traducción en el TLB y regresa al programa que efectuó el fallo.
    - Esto permite al kernel del sistema operativo elegir entre cualquier combinación de tabla de páginas, tabla de segmentos, tabla invertida, etc., necesaria para llevar a cabo la traducción de la dirección.
    - En sistemas que no utilicen TLB basado en software, entonces el hardware realiza la misma labor, pero en este caso el hardware debe especificar el formato exacto para las tablas de páginas y segmentos.
    - Por esta razón las TLB manipuladas por software son más flexibles, al costo de ser un poco más lentas para manejar los fallos TLB. Si estos fallos son muy infrecuentes, el impacto en el desempeño de un sistema TLB manejado por software es mínimo.
- La ilusión de tener memoria ilimitada es provista por el sistema operativo, utilizando la memoria principal como un cache para el disco (swap).
- Para esta asignación, la traducción de las direcciones de las páginas, permite la flexibilidad de traer páginas del disco en el momento en que se necesiten.
- Cada entrada en el TLB posee un bit de validez:
  - Si el bit está encendido entonces la página virtual está en memoria, solo se debe actualizar el TLB.

    - Si está apagado o la página virtual no está en el TLB, entonces se necesita una tabla de páginas administrada por software (SO, alias usted) para saber si la página está en memoria (cargando el TLB cuando se efectúe la traducción) o si la página debe ser traída del disco (swap).

- Adicionalmente, el hardware coloca el bit de uso cada vez que la página se utiliza y el bit de suciedad si el contenido fue modificado.
- Cuando el programa hace referencia a una página que no se encuentra en el TLB, el hardware genera una excepción de falta de página (PageFault exception), llevando el control al kernel (ExceptionHandler).
- El kernel, verifica en la tabla de páginas del hilo/proceso, si la página no está en memoria, entonces lee la página del disco, corrige la entrada en la tabla para apuntar a la nueva página y luego pasa el control al programa que produjo la excepción.
- Por supuesto, el kernel debe encontrar primero espacio en la memoria principal para esta nueva página que debe ser traída, probablemente desocupando una casilla y escribiendo esa página removida en el disco (swap), si ésta fue modificada antes (dirty).
- Como en cualquier sistema de caching, el desempeño depende de la estrategia para determinar cuáles de las páginas permanecen en memoria y cuáles se mantienen en el disco.
- En un fallo de página, el kernel debe decidir cuál de las páginas va a reemplazar; idealmente debe reemplazar aquella que no se va a utilizar por un largo tiempo, manteniendo en memoria solamente las páginas que van a ser referenciadas pronto.
- Otra consideración importante, es que si la página que se reemplaza había sido modificada, ésta debe ser guardada en el disco antes de traer la nueva.
- En muchos sistemas de memoria virtual (como UNIX) se evita ese gasto de tiempo extra, escribiendo las páginas al disco por adelantado, de manera que cualquier fallo de página posterior podrá ser completado más rápidamente.

4. Diseño

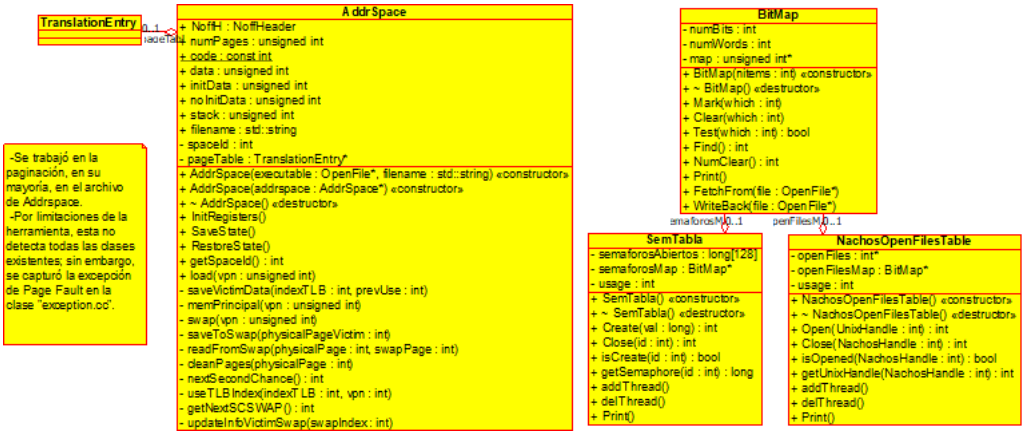

diagram.png

5. Desarrollo

En primer lugar, modificamos el código de Addrspace, para que pueda recibir como parámetro el nombre del ejecutable, que será el que vamos a cargar a memoria.

Para depurar, se agregaron campos para guardar el nombre del archivo ejecutable, número de página donde está ubicado el programa. El caso de Page Fault debe ser atrapado por Exception e identificar el número de página.

Luego, seguimos resolviendo el problema de memoria virtual con la función load en Addrspace, que es la que se encarga de colocar la página en el lugar correspondiente. Las páginas puede ser de distinto tipo, dependiendo su bit de validación y su bit sucio, y con ello, debemos realizar una acción específica. A continuación se mostrarán los diferentes posibles casos:

- **Memoria Princial**
  - Solo si su bit sucio y su bit de validez son falsos.
  - Se selecciona una nueva página.
  - Aplico algoritmo **Segunda Oportunidad**.
  - Si la página nueva está con su bit sucio encendido, hago swap.
  - Actualizo TLB.
- **Swap**
  - Solo si su bit sucio es verdadero y su bit de validez es falso.
  - Aplico swap.
  - Actualizo TLB.
- **Page Table**
  - Solo si su bit de validez es verdadero.
  - Actualizo TLB.

También debemos tomar en cuenta el caso en el que no haya más espacio en la memoria. Debemos proceder a liberarla, ya sea enviando una página de memoria al Swap o eliminarla.

6. Manual de usuario

Requerimientos de Software

- **Sistema Operativo:** Linux
- **Arquitectura:** 64 bits
- **Ambiente:** Consola

Compilación

Para compilar el programa, primeramente se debe ubicar con la consola en la carpeta ../nachOS-64/code/vm
Una vez ubicado en esa carpeta, debe digitar 'make depend' y presionar la tecla ENTER. Como se muestra a continuación.

```
make depend
```

Seguidamente, digite 'make' y vuelva a presionar ENTER.

```
make
```

Por último, para realizar pruebas se ejecuta el siguiente comando, por lo que debe escribir exactamente lo mismo en consola:

```
./nachos -x ../test/nombreDelEjecutable
```

Si se desea correr el programa mostrando los mensajes de DEBUG, puede correr el programa agregando la bandera de debug, de la siguiente manera:

```
./nachos -x ../test/nombreDelEjecutable -d
```

Especificación de las funciones del programa

El programa solamente lee archivos que sean de la forma Nombre.c. Algunas pruebas que puede realizar son las siguientes:

```
./nachos -x ../test/halt
```

```
./nachos -x ../test/sort
```

```
./nachos -x ../test/sort-ok
```

```
./nachos -x ../test/matmult
```

7. Casos de Prueba

Prueba:
El resultado obtenido en la prueba Halt.c se puede observar en la figura 1.
El resultado obtenido en la prueba Sort.c se puede observar en la figura 2.
El resultado obtenido en la prueba Sort-ok.c se puede observar en la figura 3.
El resultado obtenido en la prueba Matmult.c se puede observar en la figura 4.

Viendo los resultados, se nota donde un programa tan sencillo como el Halt solamente genera 3 fallos de página. Sin embargo, un problema más grande como los sorts, en específico el sort-ok generó 4403 fallos de página.

La cantidad de fallos de página pueden verse alterados, si uno cambia la cantidad de páginas físicas en NachOS. Esta variable se encuentra en ../nachOS-64/code/machine. en el archivo machine.h

```
1          const int NumPhysPages = 32;
```

En las figuras 5 y 6 se muestran pruebas cambiando el valor de esta variable, por el doble y la mitad, respectivamente. Se realizó en la prueba de sort.c

Es claro que la cantidad de fallos de página son muy distintos cuando se cambia la variable de cantidad de páginas físicas. Esto se debe a que entre menos páginas físicas se tengan más rápido ocurrirá un fallo de página, por lo tanto se aumenta la cantidad de fallos. En el caso opuesto, o sea, aumentando la cantidad de páginas físicas, se tiene más espacio, por lo tanto hay más espacio antes de que ocurra el primer fallo de página.

Sin embargo, hay que tomar en cuenta la anomalía de Belady, donde si se está utilizando un algoritmo FIFO (first - in - first - out), el aumentar la variable de páginas físicas, causará un aumento en el fallo de páginas. Sin embargo, con estas pruebas no es el caso.

Referencias

[1] SILBERSCHATZ, A., AND PETERSON, J. L. *Operating system concepts*. Addison-Wesley Reading, MA, 2012.

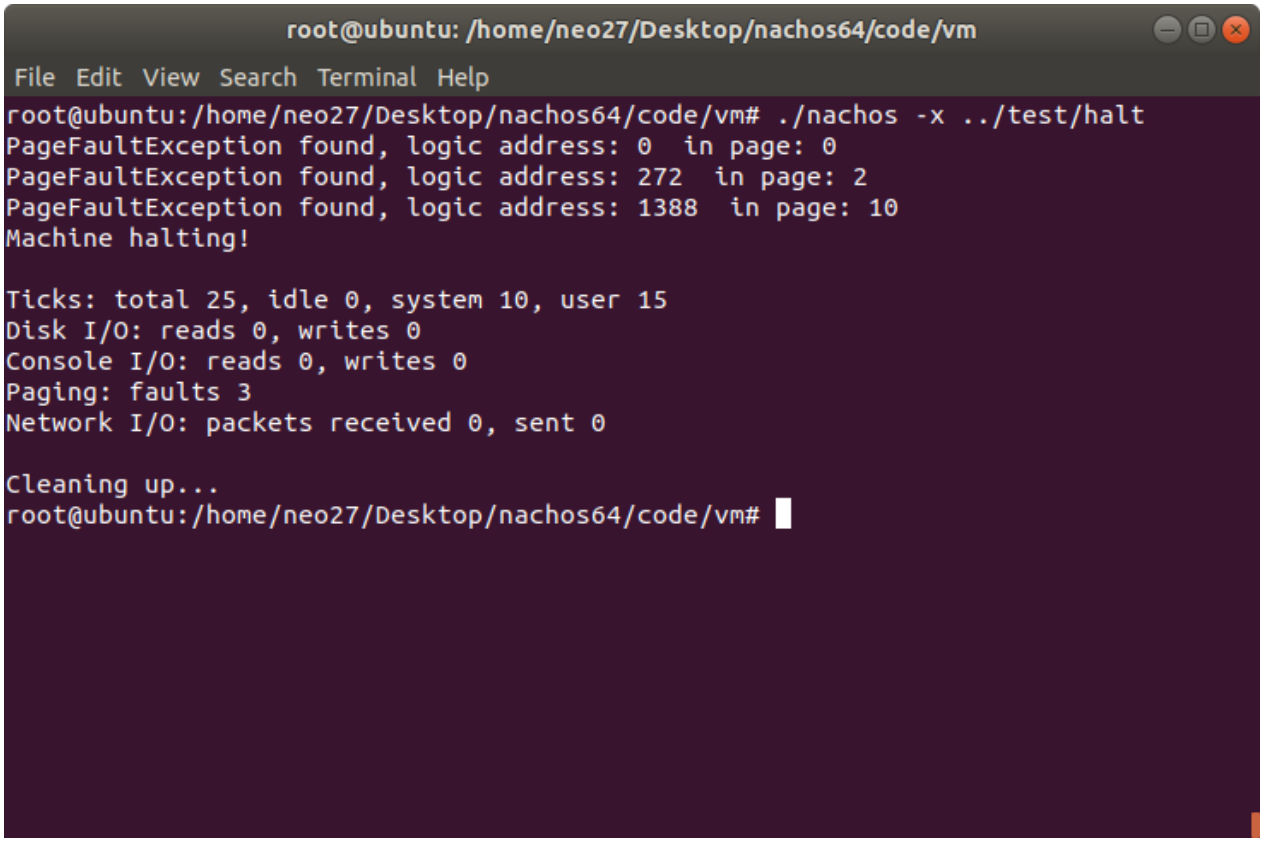

Figura 1: Corrida Prueba Halt

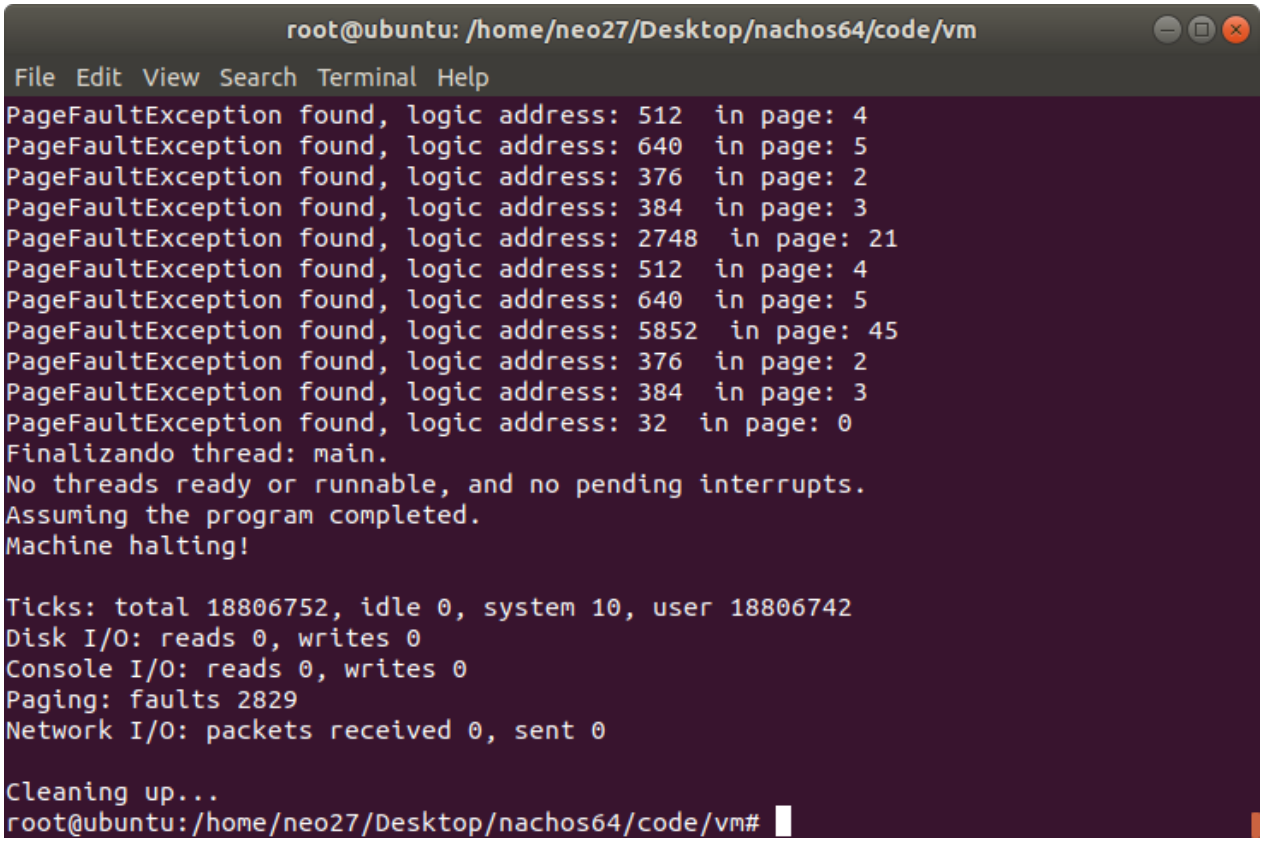

Figura 2: Corrida Prueba Sort

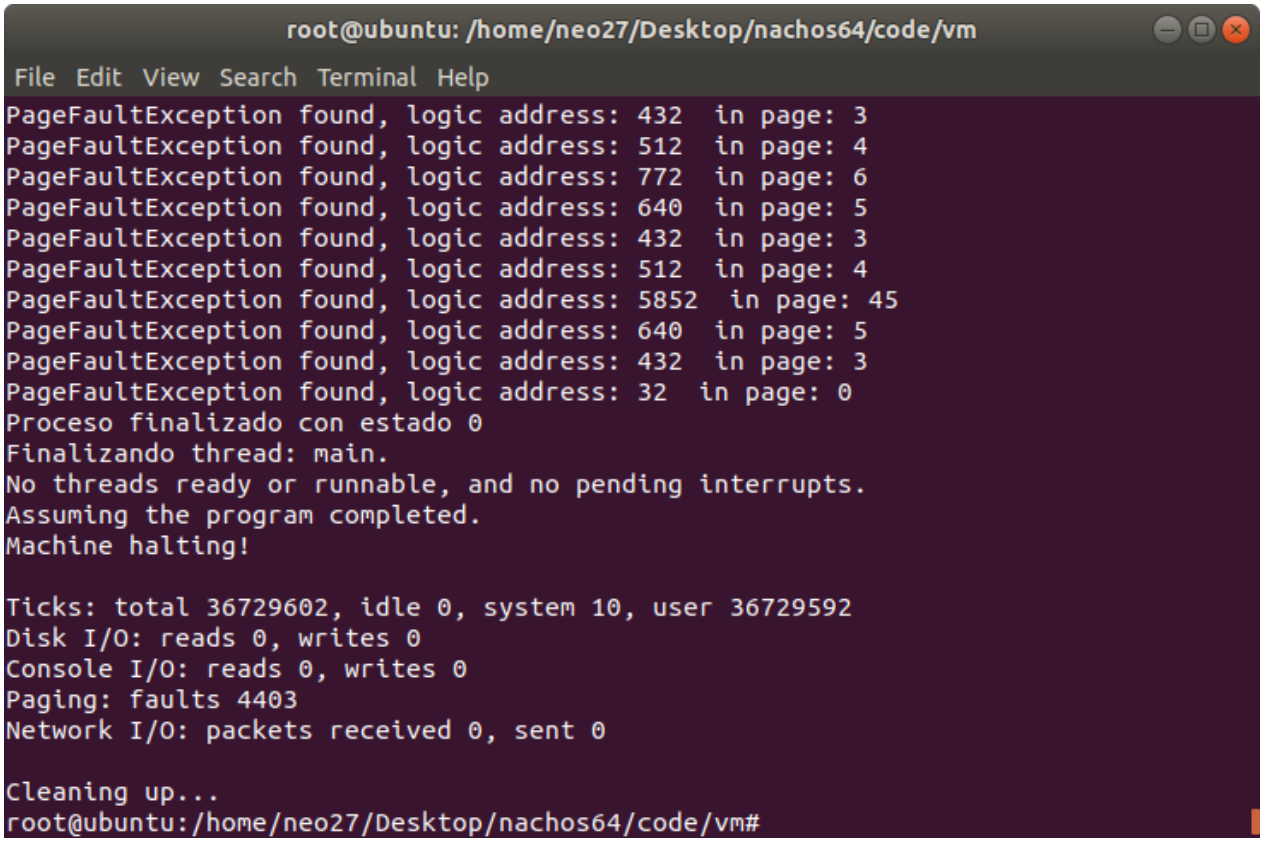

Figura 3: Corrida Prueba Sort-ok

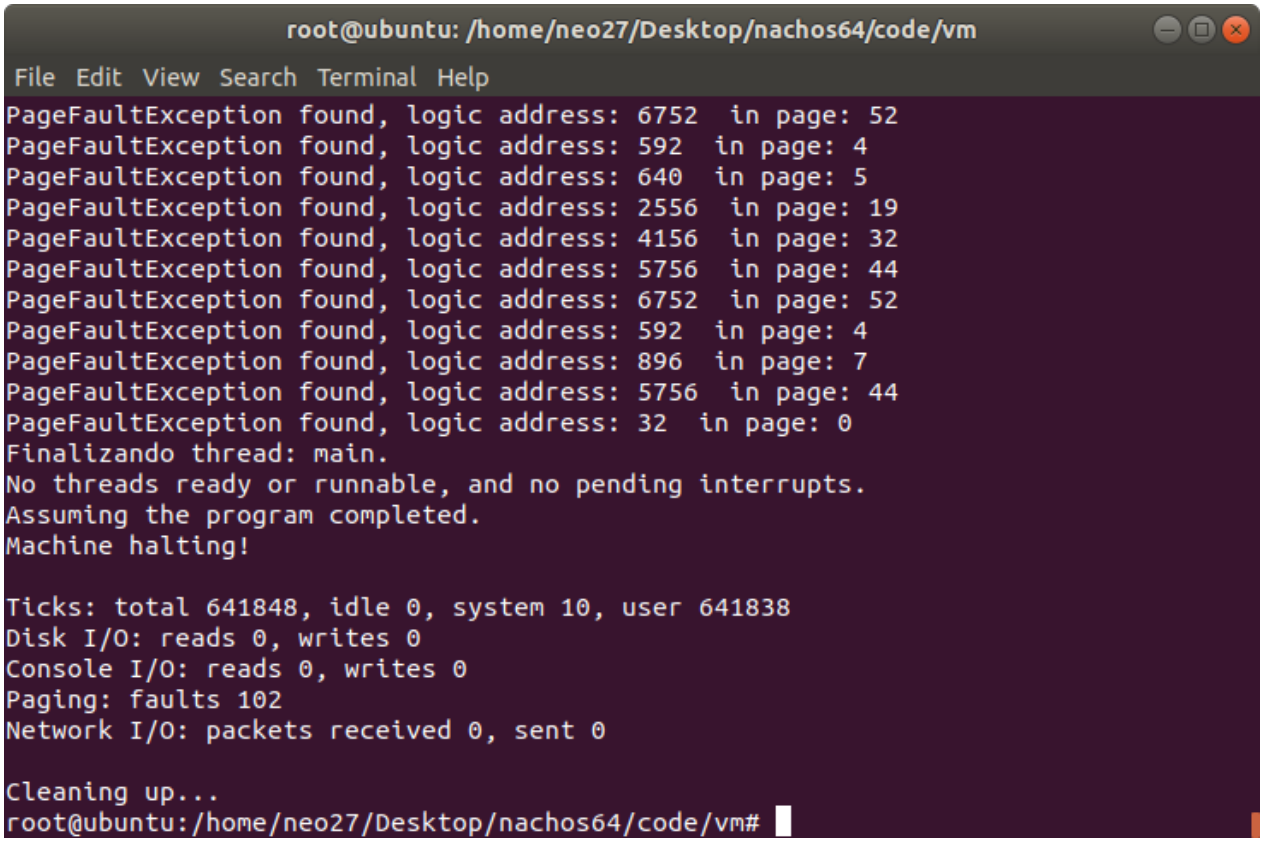

Figura 4: Corrida Prueba Matmult

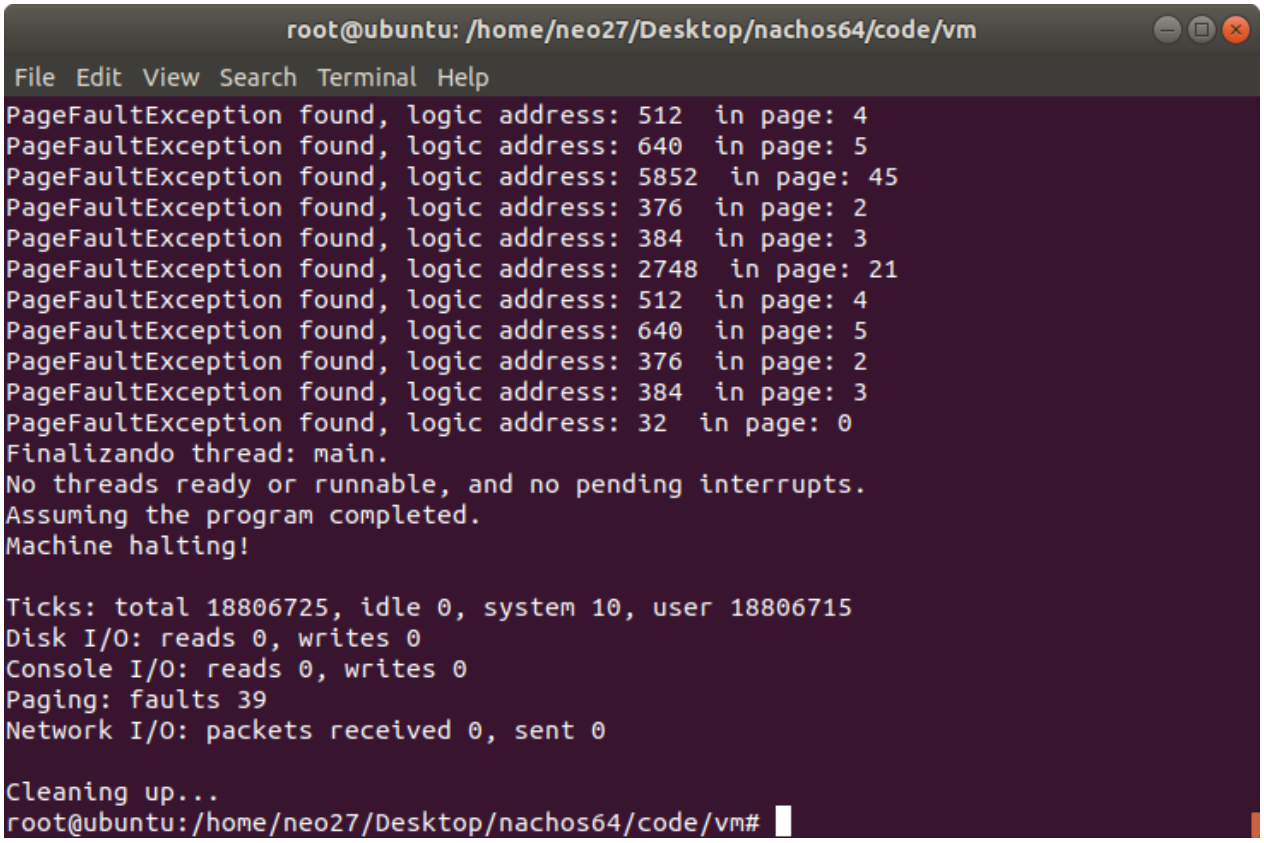

Figura 5: Corrida Prueba Sork-ok con 64 páginas físicas

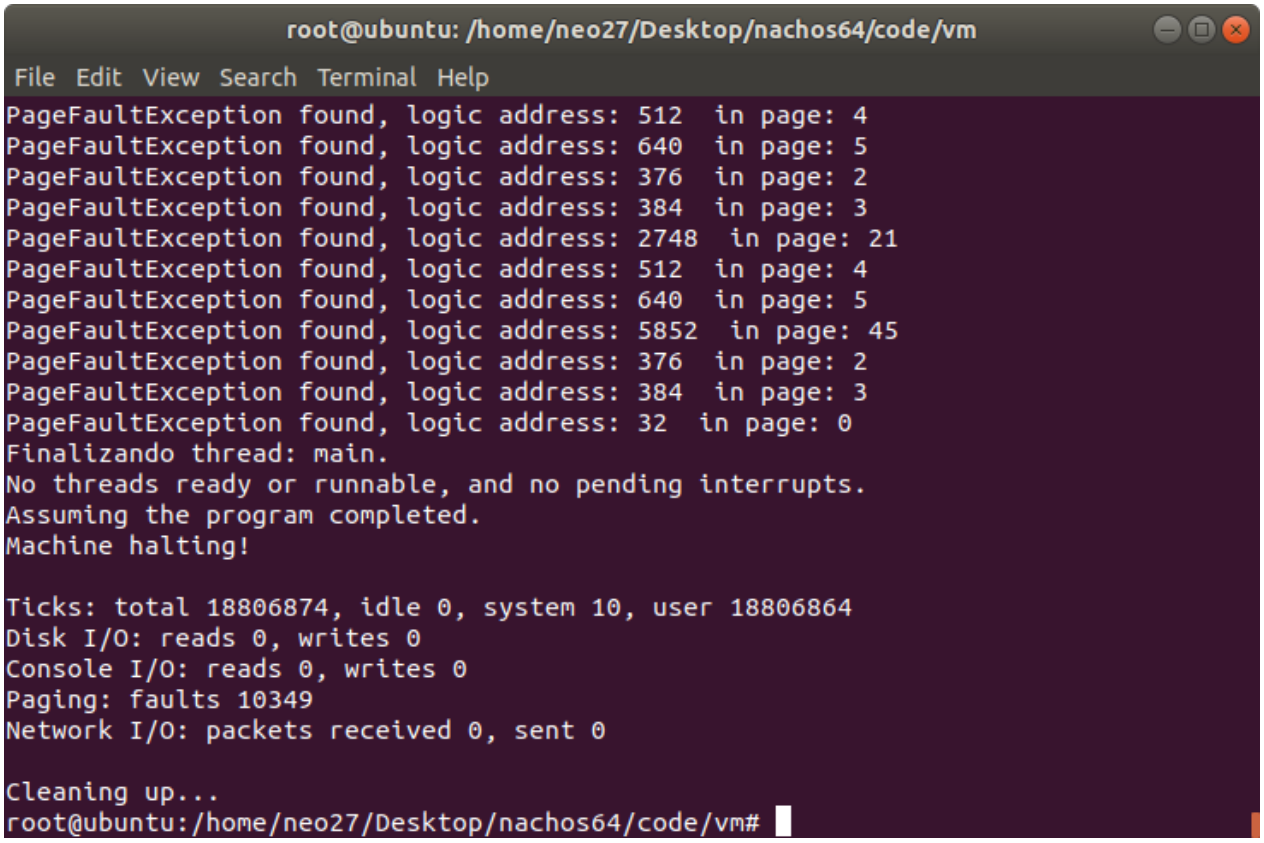

Figura 6: Corrida Prueba Sork-ok con 16 páginas físicas