# Sistemas Operativos II - Entrega final

Jeremías Rodríguez

2020

## Planchas previas

- Cursé la materia en 2017.
- La plancha 1 se encuentra aquí
- La plancha 2 se encuentra aquí
- La plancha 3 se encuentra aquí .

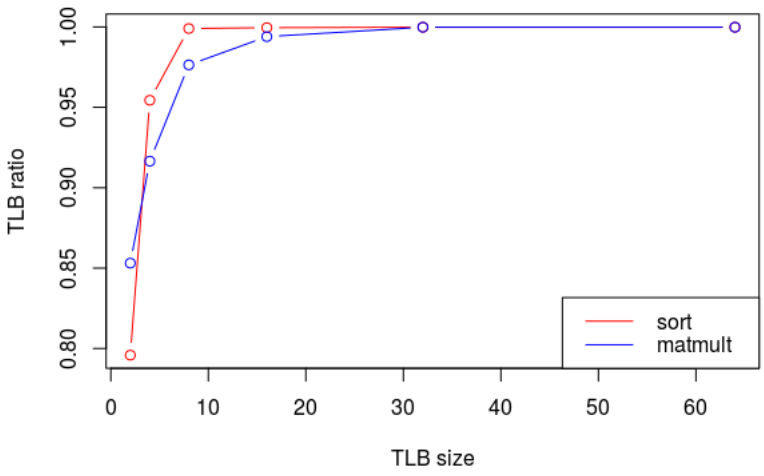
Durante la realización y testeo de la plancha final, encontré algunos errores en estas planchas (por ejemplo en la implementación de locks, o de la syscall exec). Estos errores fueron corregidos directamente en el código de la entrega final.

## Ejercicios 1 y 2

La TLB se encuentra habilitada al usar el ejecutable del directorio vm. Cuando una página virtual no se encuentra en la TLB, se lanza una excepción que es manejada por AddressSpace::handleTLBMiss. También se modificaron otros métodos en AddressSpace para especificar el compartamiento de la TLB en cambios de contexto, etc.

La siguiente tabla resume la performance de TLB en dos programas de ejemplo, sort y matmult:

| TLB SIZE | 2 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|
| Sort hits | 17362765 | 20823379 | 21796245 | 21807871 | 21814338 | 21814338 |
| Sort misses | 4454621 | 994007 | 21141 | 9515 | 3048 | 3048 |
| Sort ratio | 0.795822 | 0.9544 | 0.999031 | 0.999564 | 0.999860 | 0.999860 |
| Matmult hits | 619496 | 665591 | 709059 | 721829 | 726075 | 726075 |
| Matmult misses | 106694 | 60599 | 17131 | 4361 | 115 | 115 |
| Matmult ratio | 0.853077 | 0.9165 | 0.976410 | 0.993995 | 0.999842 | 0.999842 |



Basado en los números que obtuve, parece razonable elegir un tamaño de TLB de 8 o 16 bits. No hay mayor ganancia en elegir tamaños mayores para los programas testeados. De aquí en adelante, usaré TLB size = 8.

## Ejercicios 3

Demand Loading fue implementado en la clase AddressSpace, y se habilita o deshabilita definiendo DEMAND_LOADING. Se usa el campo valid de la page table para indicar que una página nunca fue cargada, si se invoca a loadPage cada vez que una página es referenciada por primera vez. Esta adición permitió ejecutar programas que, en caso de no tener memoria virtual, no podrían haberse corrido, pues a priori ocuparían más memoria de la disponible. Por ejemplo, halt utiliza 11 páginas de memoria de las cuales solo 3 son utilizadas (las otras pertenecen al stack).

## Ejercicios 4, 5 y 6

El uso de memoria virtual está habilitado por default al usar el ejecutable del directorio vm. Agregué la clase Paginador que se encarga de administrar los marcos de memoria física, asignándolos a distintos threads a medida que lo soliciten. De ser necesario se envían páginas a swap, usando alguno de los siguientes algoritmos de reemplazo de páginas que implementé: Random, FIFO o Reloj Mejorado.

Las siguientes tabla compara el desempeño de los últimos dos. Adicionalmente, la clase statistics guarda la traza de páginas referenciadas para calcular el desempeño del algoritmo de reemplazo óptimo. Mi implementación del algoritmo óptimo dada una traza usa fuerza bruta y es poco eficiente, la idea fue sólo usarla para generar los números de la siguiente tabla.

| | sort | | matmult size=20 | | matmult size=30 | |
|---|---|---|---|---|---|---|
| | #SwapIn | #SwapOut | #SwapIn | #SwapOut | #SwapIn | #SwapOut |
| FIFO | 3005 | 3013 | 65 | 80 | 31920 | 31982 |
| Clock | 2429 | 2437 | 45 | 60 | 6490 | 6552 |
| Óptima | 316 | 316 | 28 | 28 | 2972 | 2972 |

Como podemos ver, los números muestran que el algoritmo del reloj mejorado efectivamente mejora la performance respecto a FIFO.

## Comentarios adicionales

- El cálculo del algoritmo de reemplazo óptimo está desactivado por default porque es bastante costoso.
- Agregué varios programas a la carpeta de testing, algunos escritos por mi y otros recolectados de internet (como snake.c que dibuja una viborita que se mueve aleatoriamente en la pantalla). Una de las formas mas exigentes pero efectivas de testear que encontré fue setear el tamaño de la TLB y de la memoria principal a 1.