Laboratorio 3: Planificación https://www.famaf.proed.unc.edu.ar/pluginfile.p...

**Laboratorio 3: Planificación**

Sistemas Operativos - FaMAF - UNC

Versiones 2014, 2016-2020: Carlos Bederián

Versión 2015: Facundo Ramallo, Pablo Ventura

**Objetivos**

El planificador apropiativo de xv6 utiliza un algoritmo sencillo para distribuir tiempo de procesador entre los procesos en ejecución, pero esto tiene un costo aparejado. Estudiaremos cómo funciona este scheduler, analizaremos los procesos que se benefician y perjudican con esta decisión de diseño, e intentaremos mejorar la situación reemplazando la política de planificación por una propia.

**Primera Parte: Estudiando el planificador de xv6**

Comenzaremos este laboratorio leyendo código para entender cómo funciona la planificación en xv6:

1. Analice el código del planificador: ¿Qué política utiliza xv6 para elegir el próximo proceso a correr? Pista: xv6 nunca sale de la función scheduler por medios “normales”. (que un proceso termine seria???)

preentive, nonpreentive, context switch

ver que tipo de planificador: round robin

que politica usa: recorre TODOS los procesos disponibles en la proccess table y ejecuta el primero que esta en estado RUNNABLE(ready)

y cuanto duran los Quantum:10000000\*tick(cuanto es un tick????)

TICR:address donde guardo el valor 10000000

lapic[index] = value;

2. Analice el código que interrumpe a un proceso al final de su quantum:

a. ¿Cuánto dura un quantum en xv6?

10000000\*tick(cuanto es un tick????)

b. ¿Hay alguna forma de que a un proceso se le asigne menos tiempo? Pista: Se puede empezar a buscar desde la system call uptime.

se puede disminuyendo el valor del tick(linea69-lapic.c)

**Segunda Parte: Viendo cómo el planificador afecta a los procesos**

Pasamos a ver cómo el planificador de xv6 afecta a los distintos tipos de procesos en la práctica. Para ello se deberán integrar a xv6 los programas de espacio de usuario iobench(mide cuan interactivo es, proceso que levanta datos de entrada salida constantemente y mide cuantas veces por segundo lo puede hacer) y cpubench(p poder de computo, punto flotante), que realizan mediciones (no muy precisas) de respuesta de entrada/salida y de poder de cómputo,

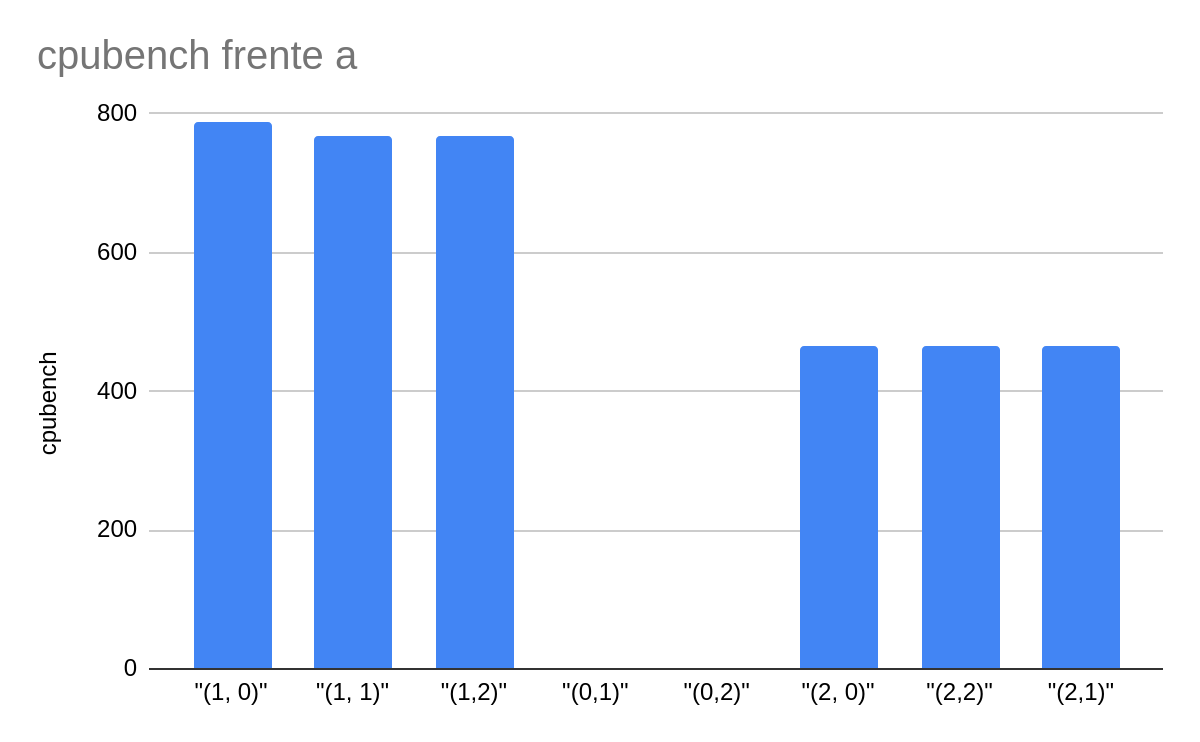
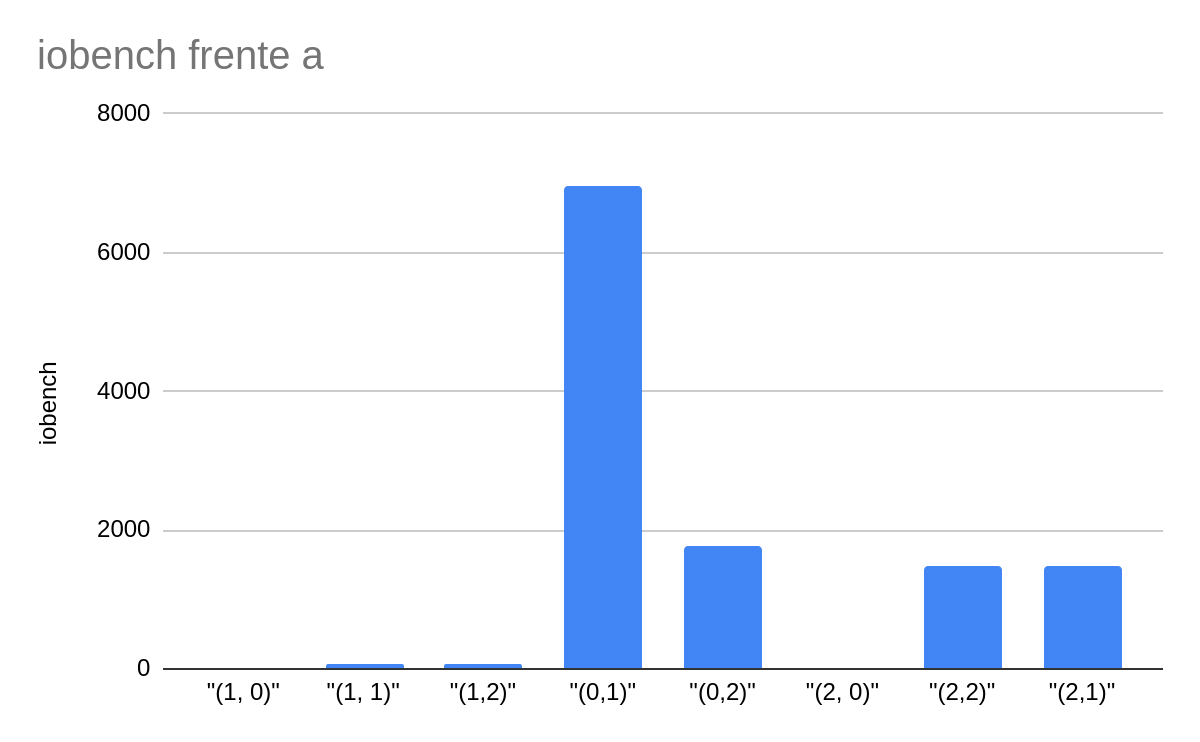
respectivamente.

**Importante**: Aunque xv6 soporta múltiples procesadores, correremos nuestras mediciones lanzando la máquina virtual con un único procesador. (i.e. make CPUS=1 qemu-nox)

1. Mida la respuesta de I/O y poder de cómputo obtenido para las distintas combinaciones posibles entre 0 y 2 iobench junto con entre 0 y 2 cpubench, y

grafique los resultados en el informe.

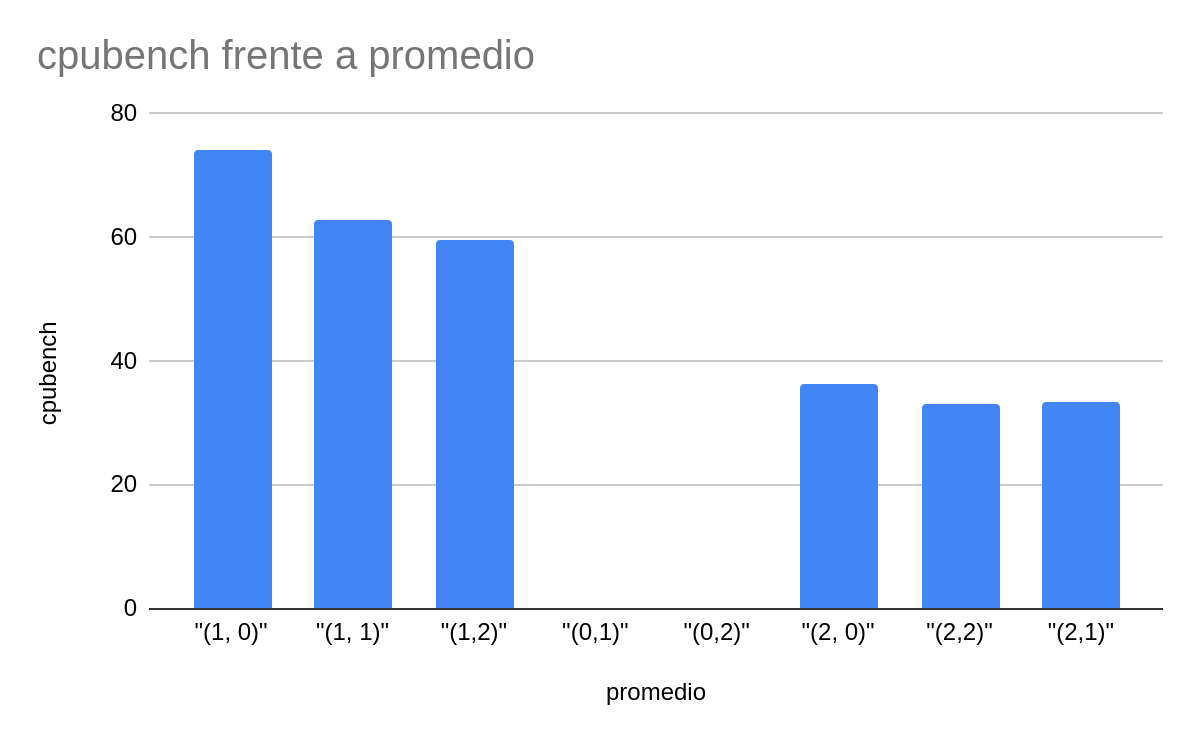
|  | cpubench | iobench |
| --- | --- | --- |
| "(1, 0)" | 787,2 | 0 |
| "(1, 1)" | 768,6 | 81,5 |
| "(1,2)" | 767,7 | 68,6 |
| "(0,1)" | 0 | 6961,9 |
| "(0,2)" | 0 | 1778 |
| "(2, 0)" | 464,7 | 0 |
| "(2,2)" | 464,7 | 1481,666667 |
| "(2,1)" | 464,7 | 1481,666667 |

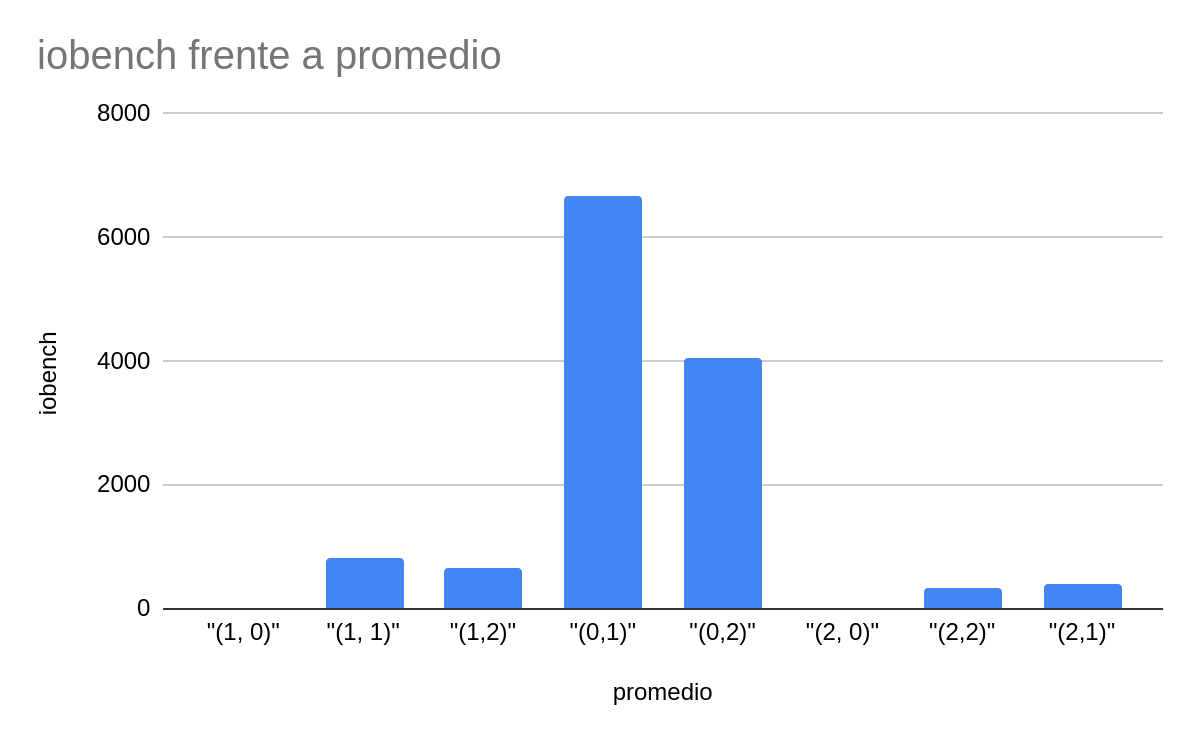


2. Repita el experimento para quantums 10, 100 y 1000 veces más cortos. Tenga en cuenta que modificar el quantum afecta el funcionamiento de iobench y cpubench, o sea que deberá modificarlos para que funcionen igual que con el quantum inicial.

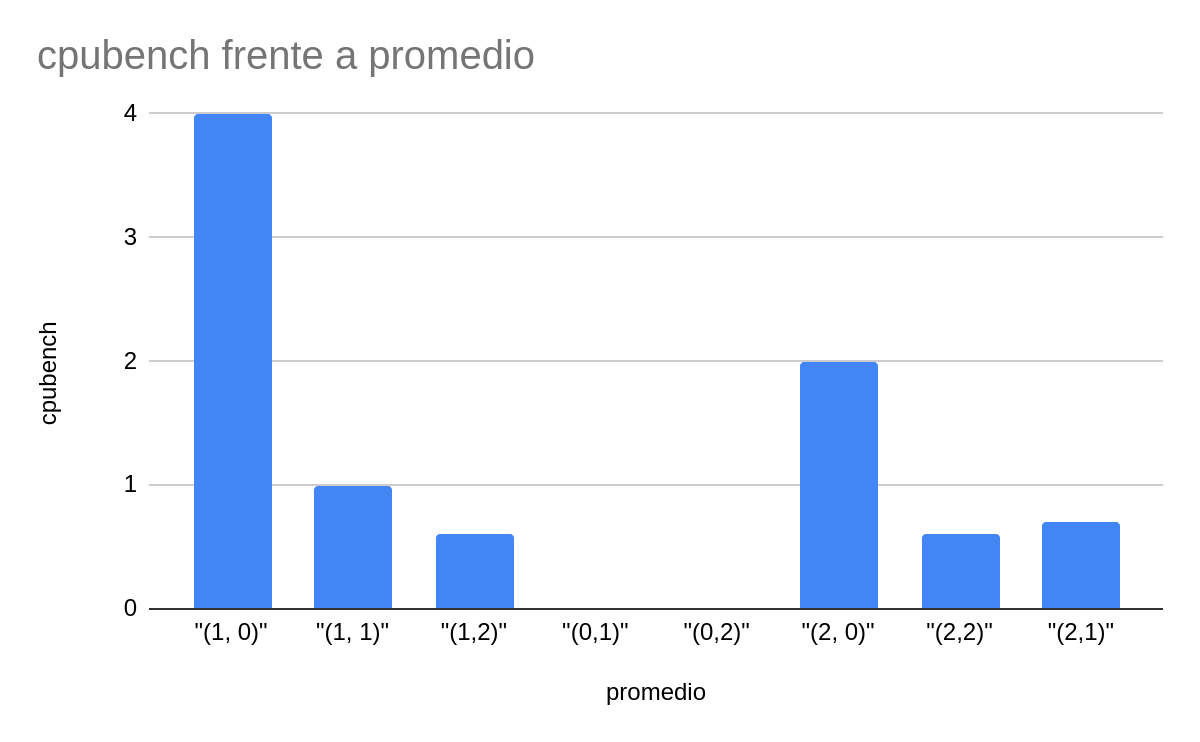
---Quantum 10 veces mas pequeño y minticks=25

**1000000, minticks: 2500**





**100000, 25000 minticks**

****



**10000, 250000 minticks**

**Tercera Parte: Rastreando la prioridad de los procesos**

Habiendo visto las propiedades del planificador existente, lo reemplazaremos con un planificador MLFQ de tres niveles. A esto lo haremos de manera gradual, empezando por rastrear la prioridad de los procesos, sin que esto afecte la planificación todavía.

1. Agregue un campo en struct proc que guarde la prioridad del proceso (entre 0 y NPRIO-1 para #define NPRIO 3 niveles en total) y manténgala

actualizada según el comportamiento del proceso:

MLFQ regla 3: Cuando un proceso se inicia, su prioridad será

máxima.

MLFQ regla 4: Descender de prioridad cada vez que el proceso

pasa todo un quantum realizando cómputo.

Ascender de prioridad cada vez que el proceso bloquea antes de

terminar su quantum. **Nota**: Este comportamiento es distinto al del

MLFQ del libro.

2. Para comprobar que estos cambios se hicieron correctamente,

modifique la función procdump (que se invoca con CTRL-P) para que imprima la prioridad de los procesos. Así, al correr nuevamente iobench y cpubench, debería darse que cpubench tenga baja prioridad mientras que iobench

tenga alta prioridad.

**Cuarta Parte: Implementando MLFQ**

Finalmente implementaremos la planificación propiamente dicha para que nuestro xv6 utilice MLFQ.

1. Modifique el planificador de manera que seleccione el próximo proceso a planificar siguiendo las siguientes reglas:

MLFQ regla 1: Si el proceso A tiene mayor prioridad que el proceso

B, corre A. (y no B)

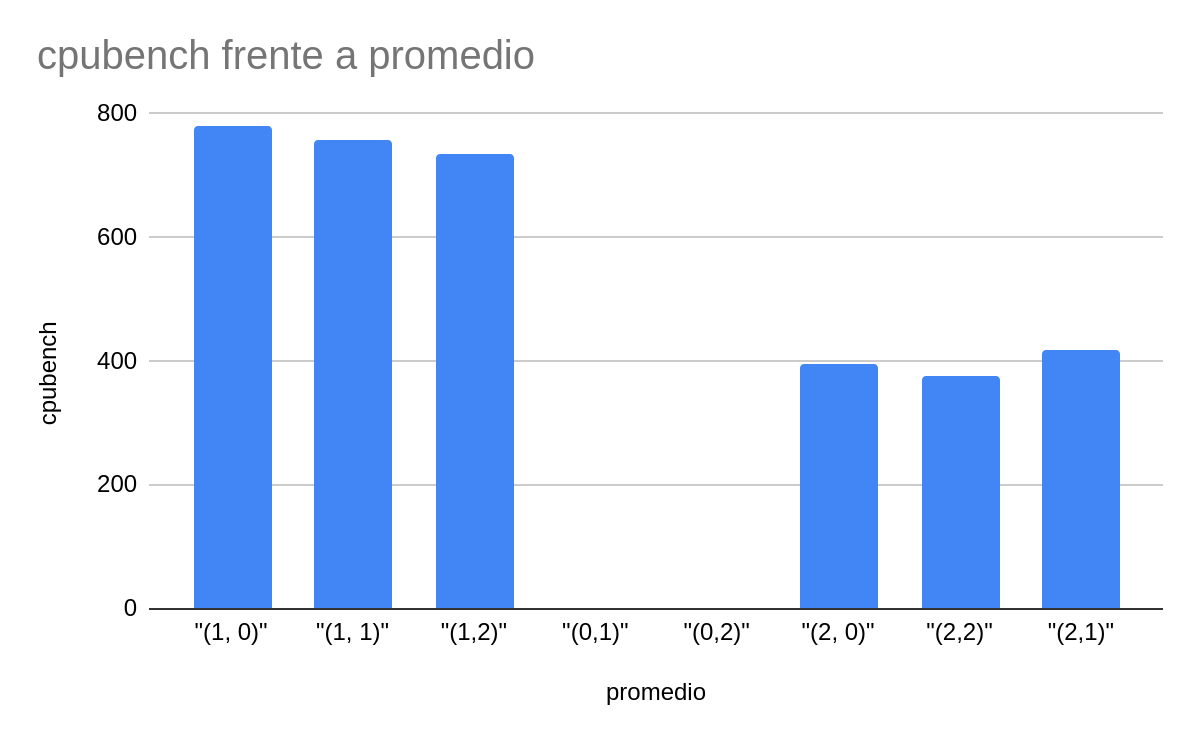
MLFQ regla 2: Si dos procesos A y B tienen la misma prioridad,

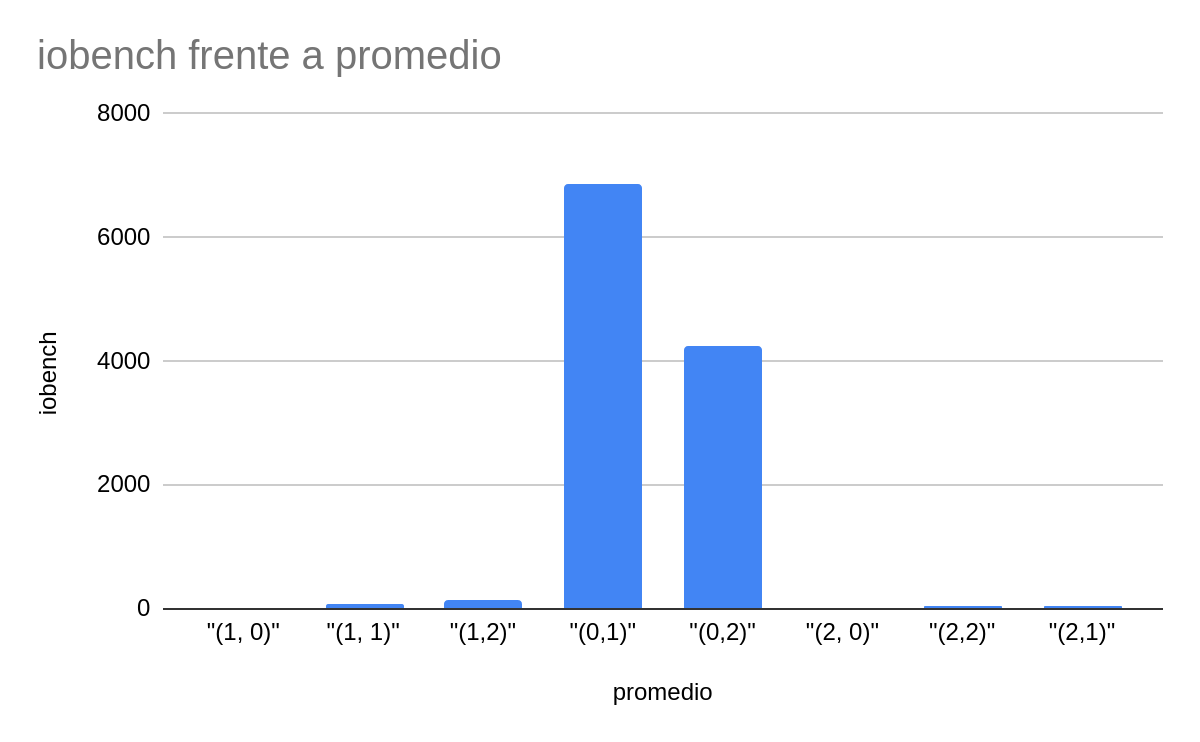
corren en round-robin por el quantum determinado.

2. Repita las mediciones de la segunda parte para ver las propiedades del nuevo planificador.

Medición 1

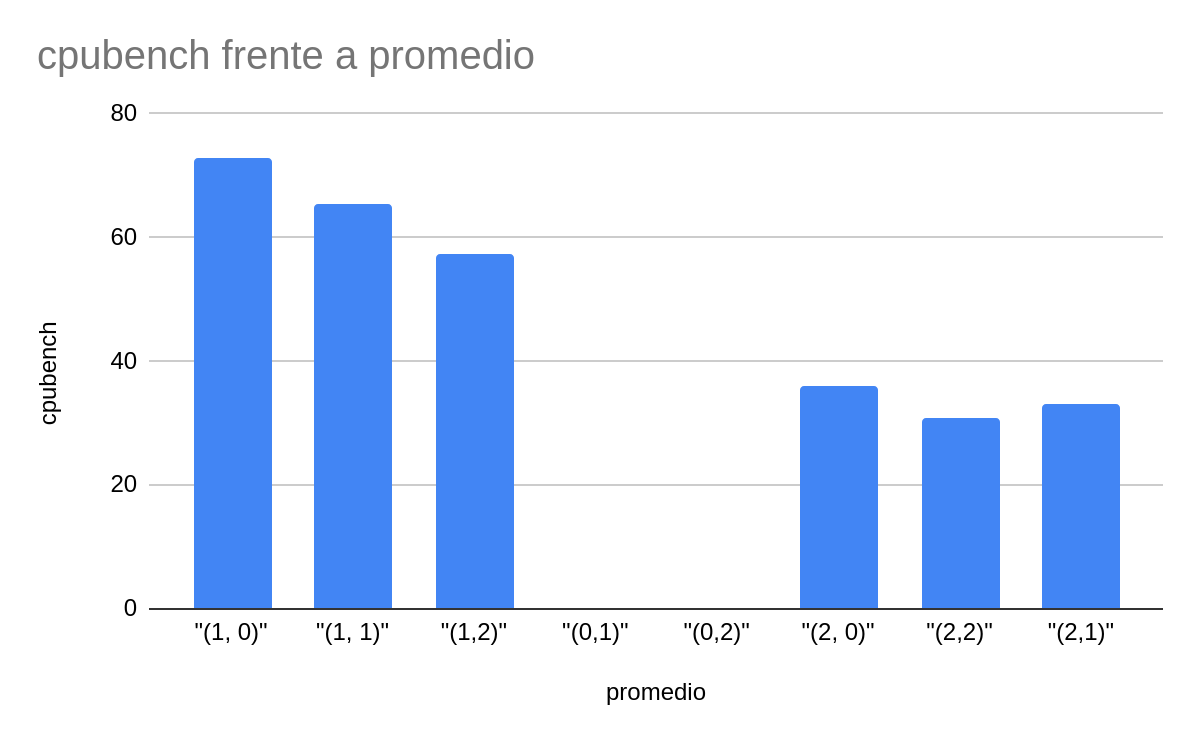
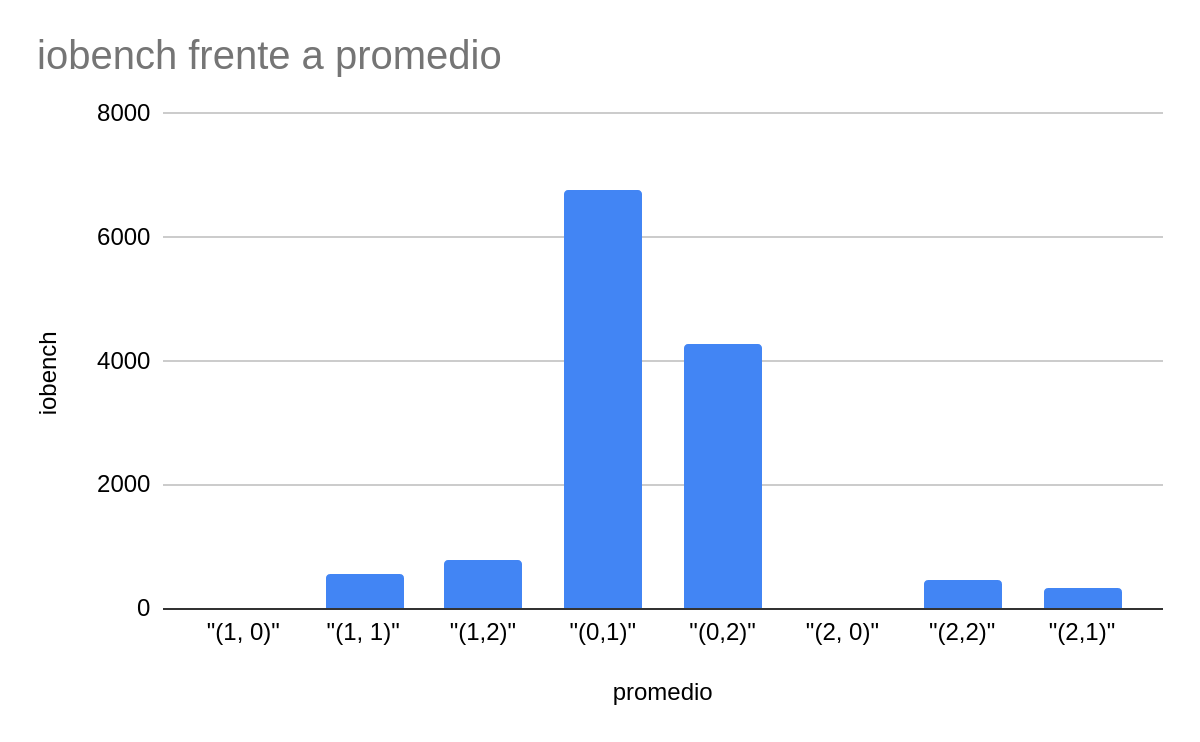
| promedio | cpubench | iobench |
| --- | --- | --- |
| "(1, 0)" | 780,6 | 0 |
| "(1, 1)" | 756,4 | 69,5 |
| "(1,2)" | 735,9 | 127,9 |
| "(0,1)" | 0 | 6873,3 |
| "(0,2)" | 0 | 4251,1 |
| "(2, 0)" | 394,3 | 0 |
| "(2,2)" | 376,7 | 57,5 |
| "(2,1)" | 419,3 | 33,6 |





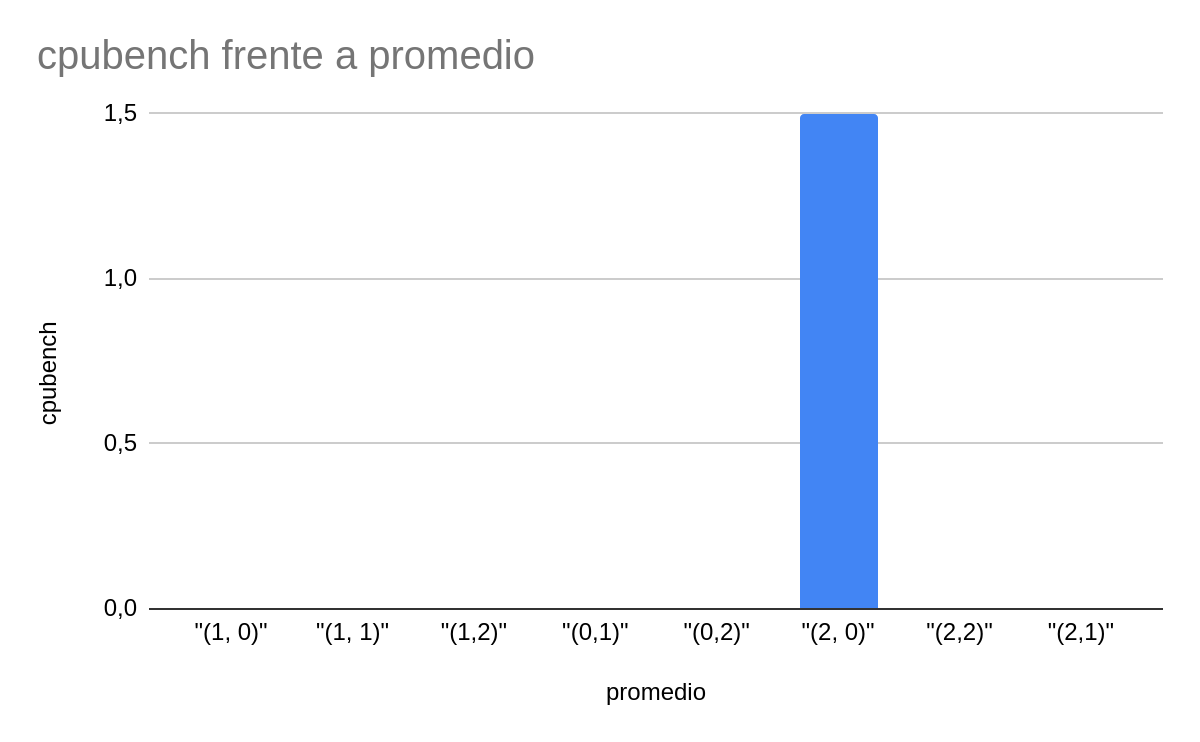
1000000-minticks = 2500

| promedio | cpubench | iobench |
| --- | --- | --- |
| "(1, 0)" | 72,9 | 0 |
| "(1, 1)" | 65,3 | 574,7 |
| "(1,2)" | 57,2 | 801,1 |
| "(0,1)" | 0 | 6754,9 |
| "(0,2)" | 0 | 4268,5 |
| "(2, 0)" | 35,9 | 0 |
| "(2,2)" | 30,7 | 466,6 |
| "(2,1)" | 33,2 | 339,7 |



ticR 100000- minticks 25000

| promedio | cpubench | iobench |
| --- | --- | --- |
| "(1, 0)" | 0 | 0 |
| "(1, 1)" | 0 | 4799,1 |
| "(1,2)" | 0 | 2718,8 |
| "(0,1)" | 0 | 5541,9 |
| "(0,2)" | 0 | 3129,1 |
| "(2, 0)" | 1,5 | 0 |
| "(2,2)" | 0 | 2176,3 |
| "(2,1)" | 0 | 3787,8 |





3. Para análisis: ¿Se puede producir starvation en el nuevo planificador? Justifique su respuesta.

**Importante**: Mucho cuidado con el uso correcto del mutex ptable.lock. 2 de 3 14/10/20 13:09

Laboratorio 3: Planificación https://www.famaf.proed.unc.edu.ar/pluginfile.p...

**Extras**

Del planificador:

1. Reemplace la política de ascenso de prioridad por la regla 5 de

MLFQ de OSTEP: .

2. Modifique el planificador de manera que los distintos niveles de

prioridad tengan distintas longitudes de quantum.

3. Cuando no hay procesos para ejecutar, el planificador consume

procesador de manera innecesaria haciendo busy waiting.

Modifique el planificador de manera que ponga a dormir el

procesador cuando no hay procesos para planificar, utilizando la

instrucción hlt.

4. (Difícil) Cuando xv6 corre en una máquina virtual con 2

procesadores, la performance de los procesos varía

significativamente según cuántos procesos haya corriendo

simultáneamente. Se sigue dando este fenómeno si el planificador

tiene en cuenta la localidad de los procesos e intenta mantenerlos

en el mismo procesador?

5. (Muy difícil) Y si no quisiéramos usar los ticks periódicos del timer por el problema de (1), qué haríamos? Investigue cómo funciona e

implemente un tickless kernel.

De las herramientas de medición:

Llevar cuenta de cuánto tiempo de procesador se le ha asignado a

cada proceso, con una system call para leer esta información desde

espacio de usuario.

**Entrega**

Deberán entregar via commits+push al repositorio del grupo para este laboratorio en bitbucket, con un directorio xv6 dentro sobre el cual

deberán hacer sus modificaciones. No copiar laboratorios anteriores, comenzar en limpio.

El coding style deberá respetar a rajatabla las convenciones de xv6.

3 de 3 14/10/20 13:09