Laboratorio 3: Planificador de procesos

Sistemas Operativos - FaMAF - UNC

- Vesión 2021-2023: Ignacio Moretti
- Versiones 2014, 2016-2020: Carlos Bederián
- Versión 2015: Facundo Ramallo, Pablo Ventura

Objetivos

El planificador apropiativo de `xv6-riscv` utiliza un algoritmo sencillo para distribuir tiempo de procesador entre los procesos en ejecución, pero esto tiene un costo aparejado. Los objetivos de este laboratorio son **estudiar** el funcionamiento del scheduler original de xv6-riscv; **analizar** los procesos que se benefician/perjudican con esta decisión de diseño; por último **desarrollar** una implementación reemplazando la política de planificación por una propia que deberá respetar ciertos condiciones y analizar como afecta a los procesos en comparacion con el planificador original.

Primera Parte: Estudiando el planificador de `xv6-riscv` y respondiendo preguntas

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

Analizar el código del planificador y responda en el informe:

1. ¿Qué política de planificación utiliza `xv6-riscv` para elegir el próximo proceso a ejecutarse? **Pista:** `xv6-riscv` nunca sale de la función `scheduler` por medios "normales".
2. ¿Cuánto dura un *quantum* en `xv6-riscv`?
3. ¿Cuánto dura un cambio de contexto en `xv6-riscv`?
4. ¿El cambio de contexto consume tiempo de un *quantum*?
5. ¿Hay alguna forma de que a un proceso se le asigne menos tiempo? **Pista:** Se puede empezar a buscar desde la system call `uptime`.
6. ¿Cúales son los estados en los que un proceso pueden permanecer en xv6-riscv y que los hace cambiar de estado?

Segunda Parte: Contabilizar las veces que es elegido un proceso por el planificador y anlaizar cómo el planificador afecta a los procesos

Como primera actividad de desarrollo deberan incorporar a la `struc proc` un contador que registre la cantidad de veces que fue elegido ese proceso por el planificador.

Luego modificar la función `procdump` (que se invoca con `CTRL-P`) para que imprima, además de lo que imprime, este contador de cantidad de veces que fue elegido ese proceso por el planificador. Y cree una system call `pstat(pid)` que tome un pid y devuelva su prioridad, la cantidad de veces que fue elegido por el scheduler y la ultima vez que fue ejecutado.

Para ver cómo el planificador de `xv6-riscv` afecta a los distintos tipos de procesos en la práctica, deberán integrar a `xv6-riscv` los programas de espacio de usuario `iobench` y `cpubench` (que adjuntamos en el aula virtual). Estos programas realizan mediciones de operaciones de escritura/lectura y operaciones de cómputo, respectivamente.

Importante: Aunque `xv6-riscv` soporta múltiples procesadores, debemos ejecutar nuestras mediciones(`iobench` y `cpubench`) lanzando la máquina virtual con un único procesador. (i.e. `make CPUS=1 qemu`)

1. Mida la respuesta de I/O y el poder de cómputo obtenido durante 3 minutos para los siguiente Casos y grafique los resultados obtenidos en el informe.

   Caso 1: 1 `iobench` solo. En este caso queremos investigar como se comporta un solo proceso iobench corriendo solo (sin otros procesos en paralelo) en xv6-riscv. Apartir de las metricas obtenidas describir este escenario.

   Caso 2: 1 `cpubench` solo. En este caso queremos investigar como se comporta un solo proceso cpubench corriendo solo (sin otros procesos en paralelo) en xv6-riscv. Apartir de las metricas obtenidas describir este escenario.

   Caso 3: 1 `iobench` con 1 `cpubench`. En este caso queremos investigar como se comporta un solo proceso iobench corriendo cuando además esta corriendo otro poceso cpubench en paralelo en xv6-riscv. Apartir de las metricas obtenidas describir este escenario. En este mismo Caso podemos ver como se comporta 1 cpubench cuando en paralelo corre 1 iobench.

   Caso 4: 1 `cpubench` con 1 `cpubench`. En este caso queremos investigar como se comporta un solo proceso cpubench corriendo cuando además esta corriendo otro pocesos cpubench en paralelo en xv6-riscv. Apartir de las metricas obtenidas describir este escenario.

   Caso 5: 1 `cpubench` con 1 `cpubench` y 1 `iobench`. En este caso queremos investigar como se comporta un solo proceso cpubench corriendo cuando además esta corriendo otro pocesos cpubench y 2 iobench en paralelo en xv6-riscv. Apartir de las metricas obtenidas describir este escenario.

2. Repita el experimento para *quantums* 10 veces más cortos:

Tercera Parte: Rastreando la prioridad de los procesos

Para esta parte deberán crear una rama en su repositorio con nombre `mlfq`.

Habiendo visto las propiedades del planificador existente, lo vamos a reemplazar con un planificador MLFQ de tres niveles. A esto lo deben hacer de manera gradual, primero rastrear la prioridad de los procesos, sin que esto afecte la planificación.

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 siendo `0` el prioridad minima y el `NPRIO-1` prioridad maxima) y manténgala actualizada según el comportamiento del proceso, además agregue el campo en `struct proc` que guarde la cantidad de veces que fue elegido ese proceso por el planificador para ejecutarse y se mantenga actualizado:
   - **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 se 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 lego de un tiempo que los procesos `cpubench` tengan baja prioridad mientras que los `iobench` tengan alta prioridad.

Cuarta Parte: Implementando MLFQ

Finalmente implementar la planificación propiamente dicha para que nuestro `xv6-riscv` 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, corre el que menos veces fue elegido por el planificador.
2. Repita las mediciones de la segunda parte para ver las propiedades del nuevo planificador.
3. Para análisis responda: ¿Se puede producir *starvation* en el nuevo planificador? Justifique su respuesta.

Importante: Mucho cuidado con el uso correcto del mutex `ptable.lock`.

Extras en una rama aparte

Para la ralización de los puntos extras deben crear una rama con el nombre `extras`, no deben estar implementadas en la rama principal del repositorio.

- Del planificador:
    1. Reemplace la política de ascenso de prioridad por la regla 5 de MLFQ de OSTEP: Priority boost.
    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-riscv 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. 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 UN INFORME** con todo el análisis que realicen además tiene que tener formato Markdown y estar incluido en el repositorio del grupo.
- Deberán entregar via commits+push al repositorio del grupo para este laboratorio en bitbucket, con un directorio `xv6-riscv` dentro sobre el cual deberán hacer sus modificaciones. No copiar del laboratorio anterior, comenzar con una copia limpia de `xv6-riscv`.
- El *coding style* deberá respetar las convenciones de `xv6-riscv`.
- Tienen 3 semanas para realizar el laboratorio, fecha de entrega hasta el **Jueves 26/10/2023 15:59h**.