Primitivas de Memoria Virtual para Porgramas de Usuario

Virtual Memory Primitives for User Programs

Leandro Liptak
Patricio Reboratti
Damián Silvani

Programación de Sistemas Operativos
Universidad de Buenos Aires

9 de junio, 2011

Memoria Virtual

- Principalmente, permite
  - Extender el espacio de direccionamiento
  - Compartir páginas entre procesos
  - Proteger páginas de código como *sólo lectura*
  - Implementar `copy-on-write`
- Puede tener otras aplicaciones...
  - Unix traduce un PF en un SIGSEGV, el usuario puede atraparlo
  - Se puede hacer algo mejor?

Memoria Virtual

- Principalmente, permite
  - Extender el espacio de direccionamiento
  - Compartir páginas entre procesos
  - Proteger páginas de código como *sólo lectura*
  - Implementar `copy-on-write`
- Puede tener otras aplicaciones...
  - Unix traduce un PF en un SIGSEGV, el usuario puede atraparlo
  - Se puede hacer algo mejor?

Memoria Virtual

- Principalmente, permite
  - Extender el espacio de direccionamiento
  - Compartir páginas entre procesos
  - Proteger páginas de código como *sólo lectura*
  - Implementar `copy-on-write`
- Puede tener otras aplicaciones...
  - Unix traduce un PF en un SIGSEGV, el usuario puede atraparlo
  - Se puede hacer algo mejor?

Memoria Virtual

- Principalmente, permite
  - Extender el espacio de direccionamiento
  - Compartir páginas entre procesos
  - Proteger páginas de código como *sólo lectura*
  - Implementar `copy-on-write`
- Puede tener otras aplicaciones...
  - Unix traduce un PF en un SIGSEGV, el usuario puede atraparlo
  - Se puede hacer algo mejor?

Memoria Virtual

- Principalmente, permite
  - Extender el espacio de direccionamiento
  - Compartir páginas entre procesos
  - Proteger páginas de código como *sólo lectura*
  - Implementar `copy-on-write`
- Puede tener otras aplicaciones...
  - Unix traduce un PF en un SIGSEGV, el usuario puede atraparlo
  - Se puede hacer algo mejor?

Memoria Virtual

- Principalmente, permite
  - Extender el espacio de direccionamiento
  - Compartir páginas entre procesos
  - Proteger páginas de código como *sólo lectura*
  - Implementar `copy-on-write`
- Puede tener otras aplicaciones...
  - Unix traduce un PF en un SIGSEGV, el usuario puede atraparlo
  - Se puede hacer algo mejor?

Memoria Virtual

- Principalmente, permite
  - Extender el espacio de direccionamiento
  - Compartir páginas entre procesos
  - Proteger páginas de código como *sólo lectura*
  - Implementar `copy-on-write`
- Puede tener otras aplicaciones...
  - Unix traduce un PF en un SIGSEGV, el usuario puede atraparlo
  - Se puede hacer algo mejor?

Memoria Virtual

- Principalmente, permite
  - Extender el espacio de direccionamiento
  - Compartir páginas entre procesos
  - Proteger páginas de código como *sólo lectura*
  - Implementar `copy-on-write`
- Puede tener otras aplicaciones...
  - Unix traduce un PF en un SIGSEGV, el usuario puede atraparlo
  - Se puede hacer algo mejor?

Primitivas de Memoria Virtual

- `trap`
- `prot1` y `protN`
- `unprot`
- `dirty`
- `map2`

Primitivas de Memoria Virtual

- `trap`
- `prot1` y `protN`
- `unprot`
- `dirty`
- `map2`

Primitivas de Memoria Virtual

- `trap`
- `prot1` y `protN`
- `unprot`
- `dirty`
- `map2`

Primitivas de Memoria Virtual

- `trap`
- `prot1` y `protN`
- `unprot`
- `dirty`
- `map2`

Primitivas de Memoria Virtual

- `trap`
- `prot1` y `protN`
- `unprot`
- `dirty`
- `map2`

Memoria virtual compartida

Checkpointing concurrente

GC concurrente

En un Garbage Collector incremental basado en copia como el Baker's GC, la protección de páginas y el manejo de fallos por parte del recolector brinda un mecanismo eficiente de sincronización entre los treads mutadores y el recolector.
Para esto (suponiento un solo thread mutador):

- Se crean dos mapeos de las páginas de *from-space* con `map2`
- Para cada página, se protege el mapeo que corresponde al thread del mutador con `protN`

Ante una referencia a un objeto en *from-space* por parte del mutador, el thread recolector:

- Atrapa el fallo con `trap`
- Copia los objetos "vivos" de dicha página a *to-space*
- Actualiza los punteros a dichos objetos en el thread mutador
- Desprotege la páginas en cuestión con `unprot`

GC concurrente

En un Garbage Collector incremental basado en copia como el Baker's GC, la protección de páginas y el manejo de fallos por parte del recolector brinda un mecanismo eficiente de sincronización entre los treads mutadores y el recolector.

Para esto (suponiento un solo thread mutador):

- Se crean dos mapeos de las páginas de *from-space* con `map2`
- Para cada página, se protege el mapeo que corresponde al thread del mutador con `protN`

Ante una referencia a un objeto en *from-space* por parte del mutador, el thread recolector:

- Atrapa el fallo con `trap`
- Copia los objetos "vivos" de dicha página a *to-space*
- Actualiza los punteros a dichos objetos en el thread mutador
- Desprotege la páginas en cuestión con `unprot`

GC concurrente

En un Garbage Collector incremental basado en copia como el Baker's GC, la protección de páginas y el manejo de fallos por parte del recolector brinda un mecanismo eficiente de sincronización entre los treads mutadores y el recolector.
Para esto (suponiento un solo thread mutador):

- Se crean dos mapeos de las páginas de *from-space* con `map2`
- Para cada página, se protege el mapeo que corresponde al thread del mutador con `protN`

Ante una referencia a un objeto en *from-space* por parte del mutador, el thread recolector:

- Atrapa el fallo con `trap`
- Copia los objetos "vivos" de dicha página a *to-space*
- Actualiza los punteros a dichos objetos en el thread mutador
- Desprotege la páginas en cuestión con `unprot`

GC concurrente

En un Garbage Collector incremental basado en copia como el Baker's GC, la protección de páginas y el manejo de fallos por parte del recolector brinda un mecanismo eficiente de sincronización entre los treads mutadores y el recolector.
Para esto (suponiento un solo thread mutador):

- Se crean dos mapeos de las páginas de *from-space* con `map2`
- Para cada página, se protege el mapeo que corresponde al thread del mutador con `protN`

Ante una referencia a un objeto en *from-space* por parte del mutador, el thread recolector:

- Atrapa el fallo con `trap`
- Copia los objetos "vivos" de dicha página a *to-space*
- Actualiza los punteros a dichos objetos en el thread mutador
- Desprotege la páginas en cuestión con `unprot`

GC concurrente

En un Garbage Collector incremental basado en copia como el Baker's GC, la protección de páginas y el manejo de fallos por parte del recolector brinda un mecanismo eficiente de sincronización entre los treads mutadores y el recolector.
Para esto (suponiento un solo thread mutador):

- Se crean dos mapeos de las páginas de *from-space* con `map2`
- Para cada página, se protege el mapeo que corresponde al thread del mutador con `protN`

Ante una referencia a un objeto en *from-space* por parte del mutador, el thread recolector:

- Atrapa el fallo con `trap`
- Copia los objetos "vivos" de dicha página a *to-space*
- Actualiza los punteros a dichos objetos en el thread mutador
- Desprotege la páginas en cuestión con `unprot`

GC concurrente

En un Garbage Collector incremental basado en copia como el Baker's GC, la protección de páginas y el manejo de fallos por parte del recolector brinda un mecanismo eficiente de sincronización entre los treads mutadores y el recolector.
Para esto (suponiento un solo thread mutador):

- Se crean dos mapeos de las páginas de *from-space* con `map2`
- Para cada página, se protege el mapeo que corresponde al thread del mutador con `protN`

Ante una referencia a un objeto en *from-space* por parte del mutador, el thread recolector:

- Atrapa el fallo con `trap`
- Copia los objetos "vivos" de dicha página a *to-space*
- Actualiza los punteros a dichos objetos en el thread mutador
- Desprotege la páginas en cuestión con `unprot`

GC concurrente

En un Garbage Collector incremental basado en copia como el Baker's GC, la protección de páginas y el manejo de fallos por parte del recolector brinda un mecanismo eficiente de sincronización entre los treads mutadores y el recolector.
Para esto (suponiento un solo thread mutador):

- Se crean dos mapeos de las páginas de *from-space* con `map2`
- Para cada página, se protege el mapeo que corresponde al thread del mutador con `protN`

Ante una referencia a un objeto en *from-space* por parte del mutador, el thread recolector:

- Atrapa el fallo con `trap`
- Copia los objetos "vivos" de dicha página a *to-space*
- Actualiza los punteros a dichos objetos en el thread mutador
- Desprotege la páginas en cuestión con `unprot`

GC concurrente

En un Garbage Collector incremental basado en copia como el Baker's GC, la protección de páginas y el manejo de fallos por parte del recolector brinda un mecanismo eficiente de sincronización entre los treads mutadores y el recolector.
Para esto (suponiento un solo thread mutador):

- Se crean dos mapeos de las páginas de *from-space* con `map2`
- Para cada página, se protege el mapeo que corresponde al thread del mutador con `protN`

Ante una referencia a un objeto en *from-space* por parte del mutador, el thread recolector:

- Atrapa el fallo con `trap`
- Copia los objetos "vivos" de dicha página a *to-space*
- Actualiza los punteros a dichos objetos en el thread mutador
- Desprotege la páginas en cuestión con `unprot`

GC concurrente

En un Garbage Collector incremental basado en copia como el Baker's GC, la protección de páginas y el manejo de fallos por parte del recolector brinda un mecanismo eficiente de sincronización entre los treads mutadores y el recolector.
Para esto (suponiento un solo thread mutador):

- Se crean dos mapeos de las páginas de *from-space* con `map2`
- Para cada página, se protege el mapeo que corresponde al thread del mutador con `protN`

Ante una referencia a un objeto en *from-space* por parte del mutador, el thread recolector:

- Atrapa el fallo con `trap`
- Copia los objetos "vivos"de dicha página a *to-space*
- Actualiza los punteros a dichos objetos en el thread mutador
- Desprotege la páginas en cuestión con `unprot`

GC concurrente

En un Garbage Collector incremental basado en copia como el Baker's GC, la protección de páginas y el manejo de fallos por parte del recolector brinda un mecanismo eficiente de sincronización entre los treads mutadores y el recolector.
Para esto (suponiento un solo thread mutador):

- Se crean dos mapeos de las páginas de *from-space* con `map2`
- Para cada página, se protege el mapeo que corresponde al thread del mutador con `protN`

Ante una referencia a un objeto en *from-space* por parte del mutador, el thread recolector:

- Atrapa el fallo con `trap`
- Copia los objetos "vivos"de dicha página a *to-space*
- Actualiza los punteros a dichos objetos en el thread mutador
- Desprotege la páginas en cuestión con `unprot`

GC concurrente

En un Garbage Collector incremental basado en copia como el Baker's GC, la protección de páginas y el manejo de fallos por parte del recolector brinda un mecanismo eficiente de sincronización entre los treads mutadores y el recolector.
Para esto (suponiento un solo thread mutador):

- Se crean dos mapeos de las páginas de *from-space* con `map2`
- Para cada página, se protege el mapeo que corresponde al thread del mutador con `protN`

Ante una referencia a un objeto en *from-space* por parte del mutador, el thread recolector:

- Atrapa el fallo con `trap`
- Copia los objetos "vivos"de dicha página a *to-space*
- Actualiza los punteros a dichos objetos en el thread mutador
- Desprotege la páginas en cuestión con `unprot`

GC generacional

Heap persistente

Extensión de direccionamiento

Compresión de páginas

Detección de heap overflow

Consistencia de la TLB

Tamaño de página óptimo

Acceso a páginas protegidas

Trap handlers sincrónicos

Conclusión