

# Primitivas de Memoria Virtual para Programas 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

- Memoria compartida, distribuida en varias computadoras
- Acceso a una memoria compartida coherente
- Se divide el espacio en páginas:
  - En escritura, si la página reside en otras memorias físicas, se busca una copia actualizada y se invalida las otras copias.
  - Las de sólo lectura pueden estar en varias memorias físicas al mismo tiempo.
- Funciona similar a la virtualización con swapping:
  - Un Page Fault ocurre cuando la página no se encuentra en la propia memoria física.
  - El MMU puede buscar la página en el disco o en otra memoria física.
- Necesita trap, prot1, unprot y map2

# Memoria virtual compartida

- Memoria compartida, distribuida en varias computadoras
- Acceso a una memoria compartida coherente
- Se divide el espacio en páginas:
  - En escritura, si la página reside en otras memorias físicas, se busca una copia actualizada y se invalida las otras copias.
  - Las de sólo lectura pueden estar en varias memorias físicas al mismo tiempo.
- Funciona similar a la virtualización con swapping:
  - Un Page Fault ocurre cuando la página no se encuentra en la propia memoria física.
  - El MMU puede buscar la página en el disco o en otra memoria física.
- Necesita `trap`, `prot1`, `unprot` y `map2`



# Memoria virtual compartida

- Memoria compartida, distribuida en varias computadoras
- Acceso a una memoria compartida coherente
- Se divide el espacio en páginas:
  - En escritura, si la página reside en otras memorias físicas, se busca una copia actualizada y se invalida las otras copias.
  - Las de sólo lectura pueden estar en varias memorias físicas al mismo tiempo.
- Funciona similar a la virtualización con swapping:
  - Un Page Fault ocurre cuando la página no se encuentra en la propia memoria física.
  - El MMU puede buscar la página en el disco o en otra memoria física.
- Necesita `trap`, `prot1`, `unprot` y `map2`

# Memoria virtual compartida

- Memoria compartida, distribuida en varias computadoras
- Acceso a una memoria compartida coherente
- Se divide el espacio en páginas:
  - En escritura, si la página reside en otras memorias físicas, se busca una copia actualizada y se invalida las otras copias.
  - Las de sólo lectura pueden estar en varias memorias físicas al mismo tiempo.
- Funciona similar a la virtualización con swapping:
  - Un Page Fault ocurre cuando la página no se encuentra en la propia memoria física.
  - El MMU puede buscar la página en el disco o en otra memoria física.
- Necesita `trap`, `prot1`, `unprot` y `map2`

# Memoria virtual compartida

- Memoria compartida, distribuida en varias computadoras
- Acceso a una memoria compartida coherente
- Se divide el espacio en páginas:
  - En escritura, si la página reside en otras memorias físicas, se busca una copia actualizada y se invalida las otras copias.
  - Las de sólo lectura pueden estar en varias memorias físicas al mismo tiempo.
- Funciona similar a la virtualización con swapping:
  - Un Page Fault ocurre cuando la página no se encuentra en la propia memoria física.
  - El MMU puede buscar la página en el disco o en otra memoria física.
- Necesita `trap`, `prot1`, `unprot` y `map2`

# Memoria virtual compartida

- Memoria compartida, distribuida en varias computadoras
- Acceso a una memoria compartida coherente
- Se divide el espacio en páginas:
  - En escritura, si la página reside en otras memorias físicas, se busca una copia actualizada y se invalida las otras copias.
  - Las de sólo lectura pueden estar en varias memorias físicas al mismo tiempo.
- Funciona similar a la virtualización con swapping:
  - Un Page Fault ocurre cuando la página no se encuentra en la propia memoria física.
  - El MMU puede buscar la página en el disco o en otra memoria física.
- Necesita `trap`, `prot1`, `unprot` y `map2`

# Memoria virtual compartida

- Memoria compartida, distribuida en varias computadoras
- Acceso a una memoria compartida coherente
- Se divide el espacio en páginas:
  - En escritura, si la página reside en otras memorias físicas, se busca una copia actualizada y se invalida las otras copias.
  - Las de sólo lectura pueden estar en varias memorias físicas al mismo tiempo.
- Funciona similar a la virtualización con swapping:
  - Un Page Fault ocurre cuando la página no se encuentra en la propia memoria física.
  - El MMU puede buscar la página en el disco o en otra memoria física.
- Necesita `trap`, `prot1`, `unprot` y `map2`

# Memoria virtual compartida

- Memoria compartida, distribuida en varias computadoras
- Acceso a una memoria compartida coherente
- Se divide el espacio en páginas:
  - En escritura, si la página reside en otras memorias físicas, se busca una copia actualizada y se invalida las otras copias.
  - Las de sólo lectura pueden estar en varias memorias físicas al mismo tiempo.
- Funciona similar a la virtualización con swapping:
  - Un Page Fault ocurre cuando la página no se encuentra en la propia memoria física.
  - El MMU puede buscar la página en el disco o en otra memoria física.
- Necesita `trap`, `prot1`, `unprot` y `map2`

# Memoria virtual compartida

- Memoria compartida, distribuida en varias computadoras
- Acceso a una memoria compartida coherente
- Se divide el espacio en páginas:
  - En escritura, si la página reside en otras memorias físicas, se busca una copia actualizada y se invalida las otras copias.
  - Las de sólo lectura pueden estar en varias memorias físicas al mismo tiempo.
- Funciona similar a la virtualización con swapping:
  - Un Page Fault ocurre cuando la página no se encuentra en la propia memoria física.
  - El MMU puede buscar la página en el disco o en otra memoria física.
- Necesita `trap`, `prot1`, `unprot` y `map2`

# 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 (suponiendo 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 (suponiendo 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 (suponiendo 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 (suponiendo 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 threads mutadores y el recolector.

Para esto (suponiendo 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 threads mutadores y el recolector.

Para esto (suponiendo 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 (suponiendo 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 threads mutadores y el recolector.

Para esto (suponiendo 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 threads mutadores y el recolector.

Para esto (suponiendo 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 threads mutadores y el recolector.

Para esto (suponiendo 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 (suponiendo 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

- Cuando una página se hace más accesible, una TLB desactualizada a lo sumo causa un *cache miss*.
- Pero al proteger una página, es necesario invalidar la TLB.
- En multiprocesadores, el problema es más notorio:
  - Se debe interrumpir cada procesador e invalidar sus TLBs.
  - Costoso por software
- Solución: hacer *flush* por lote
- La mayoría de los algoritmos usan `protN`, no `prot1`

# Consistencia de la TLB

- Cuando una página se hace más accesible, una TLB desactualizada a lo sumo causa un *cache miss*.
- Pero al proteger una página, es necesario invalidar la TLB.
- En multiprocesadores, el problema es más notorio:
  - Se debe interrumpir cada procesador e invalidar sus TLBs.
  - Costoso por software
- Solución: hacer *flush* por lote
- La mayoría de los algoritmos usan `protN`, no `prot1`

# Consistencia de la TLB

- Cuando una página se hace más accesible, una TLB desactualizada a lo sumo causa un *cache miss*.
- Pero al proteger una página, es necesario invalidar la TLB.
- En multiprocesadores, el problema es más notorio:
  - Se debe interrumpir cada procesador e invalidar sus TLBs.
  - Costoso por software
- Solución: hacer *flush* por lote
- La mayoría de los algoritmos usan `protN`, no `prot1`

# Consistencia de la TLB

- Cuando una página se hace más accesible, una TLB desactualizada a lo sumo causa un *cache miss*.
- Pero al proteger una página, es necesario invalidar la TLB.
- En multiprocesadores, el problema es más notorio:
  - Se debe interrumpir cada procesador e invalidar sus TLBs.
    - Costoso por software
  - Solución: hacer *flush* por lote
  - La mayoría de los algoritmos usan `protN`, no `prot1`

# Consistencia de la TLB

- Cuando una página se hace más accesible, una TLB desactualizada a lo sumo causa un *cache miss*.
- Pero al proteger una página, es necesario invalidar la TLB.
- En multiprocesadores, el problema es más notorio:
  - Se debe interrumpir cada procesador e invalidar sus TLBs.
  - Costoso por software
- Solución: hacer *flush* por lote
- La mayoría de los algoritmos usan `protN`, no `prot1`

# Consistencia de la TLB

- Cuando una página se hace más accesible, una TLB desactualizada a lo sumo causa un *cache miss*.
- Pero al proteger una página, es necesario invalidar la TLB.
- En multiprocesadores, el problema es más notorio:
  - Se debe interrumpir cada procesador e invalidar sus TLBs.
  - Costoso por software
- Solución: hacer *flush* por lote
- La mayoría de los algoritmos usan `protN`, no `prot1`

# Consistencia de la TLB

- Cuando una página se hace más accesible, una TLB desactualizada a lo sumo causa un *cache miss*.
- Pero al proteger una página, es necesario invalidar la TLB.
- En multiprocesadores, el problema es más notorio:
  - Se debe interrumpir cada procesador e invalidar sus TLBs.
  - Costoso por software
- Solución: hacer *flush* por lote
- La mayoría de los algoritmos usan `protN`, no `prot1`

# Tamaño de página óptimo



# Acceso a páginas protegidas

¿Cómo lograr el acceso a páginas protegidas sólo por parte de algunos threads?

- Proveyendo (SO) `map2` y utilizando `prot1` o `protN`
- Proveyendo (SO) una *syscall* para copiar desde/hacia una página protegida
- Utilizando procesos en lugar de threads si el SO soporta compartición de memoria entre éstos

# Acceso a páginas protegidas

¿Cómo lograr el acceso a páginas protegidas sólo por parte de algunos threads?

- Proveyendo (SO) `map2` y utilizando `prot1` o `protN`
- Proveyendo (SO) una `syscall` para copiar desde/hacia una página protegida
- Utilizando procesos en lugar de threads si el SO soporta compartición de memoria entre éstos

# Acceso a páginas protegidas

¿Cómo lograr el acceso a páginas protegidas sólo por parte de algunos threads?

- Proveyendo (SO) `map2` y utilizando `prot1` o `protN`
- Proveyendo (SO) una `syscall` para copiar desde/hacia una página protegida
- Utilizando procesos en lugar de threads si el SO soporta compartición de memoria entre éstos

# Acceso a páginas protegidas

¿Cómo lograr el acceso a páginas protegidas sólo por parte de algunos threads?

- Proveyendo (SO) `map2` y utilizando `prot1` o `protN`
- Proveyendo (SO) una `syscall` para copiar desde/hacia una página protegida
- Utilizando procesos en lugar de threads si el SO soporta compartición de memoria entre éstos

# Acceso a páginas protegidas

¿Cómo lograr el acceso a páginas protegidas sólo por parte de algunos threads?

- Proveyendo (SO) `map2` y utilizando `prot1` o `protN`
- Proveyendo (SO) una `syscall` para copiar desde/hacia una página protegida
- Utilizando procesos en lugar de threads si el SO soporta compartición de memoria entre éstos

# Trap handlers sincrónicos

# Conclusión