Trabajo Práctico 2: *Memoria*

Sistemas Operativos

Departamento de Computación

FCEyN - UBA

Segundo Cuatrimestre 2025

Introducción

El manejo de memoria es un componente fundamental en el diseño y funcionamiento de los sistemas operativos modernos. A lo largo del tiempo, diversas técnicas han sido desarrolladas para administrar eficientemente el uso de la memoria física. Este trabajo práctico se centra en dos aspectos esenciales de la gestión de memoria: el **manejo de espacio libre** y las **políticas de reemplazo de páginas**.

En la primera parte se abordará cómo los sistemas operativos gestionan el espacio libre dentro de la memoria. Esto incluye el abordaje de problemas como la fragmentación, la eficiencia en tiempo y espacio, y la capacidad de adaptarse a cargas de trabajo variables.

La segunda parte del trabajo se enfoca en las políticas de reemplazo de páginas, un componente crítico de los sistemas de memoria virtual. A través del análisis de distintas políticas de reemplazo, se busca comprender los principios detrás de su diseño, sus ventajas, limitaciones y cómo influyen en el rendimiento general del sistema.

Ambas secciones del trabajo se apoyarán en simuladores que permiten observar y experimentar con el comportamiento de los algoritmos presentados.

Simuladores

Dentro del archivo comprimido del TP encontrarán dos programas de Python: `malloc.py`, que permite observar el funcionamiento de un manejador de memoria o *allocator* simple, y `paging-policy.py`, que permite experimentar con diferentes políticas de reemplazo de páginas.

Para más detalles sobre el uso de cada simulador, consultear los apéndices al final del documento: el **Apéndice A** contiene las instrucciones para el simulador del asignador de memoria, y el **Apéndice B** corresponde al simulador de políticas de reemplazo de páginas.

Nota de Referencia: Los simuladores utilizados en este trabajo práctico corresponden a implementaciones provistas por el libro **Operating Systems: Three Easy Pieces (OSTEP)**, de Remzi H. Arpaci-Dusseau y Andrea C. Arpaci-Dusseau. Se otorgan todos los créditos de la implementación a sus autores.

Consignas

Resolver **de forma individual** los siguientes ejercicios. Para cada caso, explicar el razonamiento y, siempre que sea razonable, realizar gráficos representativos de los resultados obtenidos.

Parte 1: Asignación de memoria

1. Ejecutar el simulador con estas opciones: `-n 10 -H 0 -p BEST -s 0` para generar unas pocas operaciones aleatorias. Sin utilizar la opción `-c`, analizar:

   a) ¿Cuál es el resultado de cada `alloc()/free()`?

   b) ¿Cuál es el estado de la free list luego de cada operación?

   c) ¿Qué podés observar de la free list a lo largo del tiempo?

2. Repetir el análisis anterior para las políticas de `WORST FIT` y `FIRST FIT`. ¿Qué cambia?

3. Incrementar la cantidad de asignaciones aleatorias (digamos `-n 1000`). ¿Qué sucede con las asignaciones más grandes a lo largo del tiempo? Ejecutar con y sin coalescing (o sea, con y sin la opción `-C`) y analizar las diferencias en los resultados.

Parte 2: Reemplazo de páginas

1. Generar accesos aleatorios con los siguientes argumentos: `-s 0 -n 10`, `-s 1 -n 10`, y `-s 2 -n 10`. Probar con las políticas `FIFO`, `LRU` y `OPT`. Sin utilizar la opción `-c`, calcular si cada acceso resulta en hit o miss.

2. Considerando 5 marcos de página, generar secuencias de referencias a páginas que tengan el peor rendimiento posible para `FIFO` y `LRU` (es decir, que provoquen la mayor cantidad de misses posible). Para la peor secuencia hallada, ¿cuántos marcos de página podrían mejorar el rendimiento y acercarse a `OPT`?

3. Generar una traza de pedidos con alguna localidad. Explicar cómo se generó dicha traza y cómo se comporta `LRU` en ese caso. Hacer el análisis también para SECOND CHANCE (no incluido en el simulador).

Requerimientos de Entrega

- El trabajo debe entregarse en un único archivo comprimido (`.zip` o `.tar.gz`) que contenga:
  - El informe, en formato PDF.
  - Todos los archivos necesarios para reproducir la experimentación incluida en el informe.
- Fecha de entrega: **26/10/2025**.

- La entrega se realizará a través del siguiente formulario: https://forms.gle/TTZZ7etZPAb7PRn57
- **No se aceptarán entregas más allá de la fecha límite, sin excepción.**

Criterios de Evaluación

- El informe deberá ser claro, prolijo, y conciso (evaluaremos la calidad de su contenido por sobre su cantidad).
- El informe deberá explicar cada una de las decisiones tomadas en las consignas.
- El informe deberá explicar adecuadamente la experimentación realizada, que justifique cada una de las respuestas (reproducible, con gráficos comparativos, que den soporte a las hipótesis, etc).

Bibliografía

- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts*. Wiley.
- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems*. Pearson.
- Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*

A. Simulador de asignación de memoria

Descripción

Este programa permite ver cómo funciona un asignador de memoria simple. Una forma de usarlo es generar operaciones aleatorias de asignación/liberación para luego analizar el aspecto de la *free list*, así como el éxito o el fracaso de cada operación. También puede indicarse una secuencia explícita de operaciones de asignación y liberación de memoria.

Uso

Para ejecutar el programa:

```
prompt> python3 malloc.py
```

Para ver todas las opciones:

```
prompt> python3 malloc.py -h
```

Opciones principales

```
-s SEED          semilla para la generación aleatoria
-S HEAPSIZE      tamaño del heap
-b BASEADDR      dirección base del heap
-H HEADERSIZE    tamaño del header
-p POLICY        tipo de búsqueda (BEST, WORST, FIRST)
-l ORDER         orden de la lista (ADDRSORT, SIZESORT+, SIZESORT-,
                 INSERT-FRONT, INSERT-BACK)
-C               coalescing de la free list
-n OPSNUM        número de operaciones aleatorias a generar
-r OPSRANGE      tamaño máximo de asignación
-P OPSPALLOC     porcentaje de operaciones que son asignaciones
-A OPSLIST       lista explícita de operaciones (+10,-0,etc)
-c               muestra el resultado de cada operación y el estado de
                 la free list a cada momento
```

Ejemplo: operaciones aleatorias

```
> python3 malloc.py -S 100 -b 1000 -H 4 -l ADDRSORT -p BEST -n 5
```

Aquí especificamos un heap de 100 bytes (`-S 100`) que comienza en la dirección 1000 (`-b 1000`), indicando un adicional de 4 bytes de header por asignación (`-H 4`). Especificamos que la free list se ordene por dirección (creciente), establecemos `BEST FIT` como criterio de búsqueda (`-p BEST`), y pedimos que se generen 5 operaciones aleatorias (`-n 5`).

Esta ejecución no mostrará el resultado de cada operación ni cómo se ve la free list. Para ello debe agregarse la opción `-c`.

Ejemplo: lista explícita de operaciones

```
> python3 malloc.py -S 100 -b 1000 -H 4 -l ADDRSORT -p BEST -A +3,-0,+5,-1,+8
```

Este comando establece las mismas condiciones que el ejemplo anterior pero especifica las siguientes operaciones:

- Asignar 3 bytes
- Liberar la primera asignación
- Asignar 5 bytes
- Liberar la segunda asignación
- Asignar 8 bytes

B. Simulador de reemplazo de páginas

Descripción

Este simulador permite experimentar con diferentes políticas de reemplazo de páginas, pudiendo modificar la política y la forma en que se especifican o generan las direcciones.

Uso

Para ejecutar el programa:

```
prompt> python3 paging-policy.py
```

Para ver todas las opciones:

```
prompt> python3 paging-policy.py -h
```

Opciones principales

```
-a ADDRESSES    un conjunto de páginas a acceder, separadas por comas;
                -1 para generar aleatoriamente
-f ADDRESSFILE  un archivo con direcciones de páginas
-n NUMADDRS     si la opción -a es -1, es la cantidad de direcciones a
                generar
-p POLICY       política de reemplazo: FIFO, LRU, OPT
                (hay otras que no usaremos)
-C CACHESIZE    cantidad de marcos de página disponibles
-m MAXPAGE      si se generan páginas aleatoriamente, es el máximo
                número de página
-s SEED         semilla para la generación aleatoria
-N              no imprime la traza detallada
-c              muesta si cada solicitud es hit o miss y el estado de
                la memoria a cada momento
```

Ejemplo: solicitud de páginas aleatorias

```
> python3 paging-policy.py -s 10 -n 3
```

Aquí especificamos la generación de 3 referencias aleatorias a páginas (`-n 3`), así como la semilla (`-s 10`).

Si se agrega la opción `-c`, se muestra el resultado de cada solicitud y cómo cambia la memoria. La política default es FIFO.

Ejemplo: lista explícita de páginas

```
> python3 paging-policy.py --addresses=0,1,2,0,1,3,0,3,1,2,1
--policy=LRU -c
```

Este comando muestra el comportamiento de la política LRU para la secuencia de pedidos de página `0 1 2 0 1 3 0 3 1 2 1`.