Test de Evaluación Trabajo Práctico YANERO – 2° Recuperatorio

Mecánica de Evaluación:

Las evaluación de trabajo práctico constará de una serie de pruebas **obligatorias** y **no obligatorias**. El cumplimiento de las pruebas de carácter obligatorio hace corresponder un **4 (cuatro)** como nota grupal. Asimismo, cada prueba no obligatoria que sea cumplida sumará puntos los cuales estarán especificados en las mismas.

Es posible también que el ayudante asignado al grupo pueda sumar calificación en concepto de *calidad de trabajo práctico*. Esto involucra, uso apropiado de las herramientas que proporciona la cátedra, calidad de código, modularización, eficiencia, etc.

Una vez finalizada la evaluación del trabajo práctico, la nota asignada al grupo será la nota inicial de cada uno de los miembros. Durante el coloquio, cuando cada uno exponga sus conocimientos, la nota individual podrá ser modificada de acuerdo a lo que el evaluador disponga.

Consideraciones a la hora de la evaluación:

- No esta permitido editar código.
- No esta permitido editar makefiles, por lo que el código debe compilar correctamente.
- La instalación (donwload de código, compilación y configuración) no debería tardar más de 15 minutos.
- La evaluación durará aproximadamente 60 minutos.
- El coloquio no debería durar más de aproximadamente 40 minutos (10 min. x integrante aprox.)
- El desempeño y participación de todos los miembros del grupo durante las pruebas formará parte de la evaluación.
- Cualquier característica del sistema no cubierta por esta pruebas que el alumno desee verificar, debe ser informado al ayudante, el cual decidirá en base a su criterio la evaluación de dicha característica.

Requerimientos de Evaluación

El objetivo es controlar la correcta aplicación de las restricciones impuestas para el código y diseño del trabajo práctico.

- Controlar que en la compilación no existan bibliotecas externas, que no hayan sido desarrolladas por el grupo y que no hayan sido permitidas o especificadas en el Trabajo Práctico (***libfuse*** y ***libmemcached***).
- Corroborar el uso de ***select***, ***poll*** o ***epoll***.
- Corroborar el uso de ***Mapping File Into Memory*** o ***Unlocked Stream Operations***.
- Corroborar el uso ***posix_madvise*** o ***posix_fadvise***.
- Corroborar el uso de mallocs dentro del proceso Remote Cache.

Configuración del Sistema:

Para las evaluaciones, a menos que se indique lo contrario:

- El RFS, el FSC y memcached se encontrarán en distintas PCs.
- **Tiempo de retardo operaciones RFS:** 0 segundos

Pruebas

<table>
<tr><th>Prueba 1</th><th>Lectura de Directorios</th></tr>
<tr><td>Desarrollo</td><td>
Esquema: Sin Cache (o buscar la forma en que no impacte, tal vez usando el libredummy_engine.so provisto por la cátedra)

Archivo de Volumen: 1kb-3mb.disk

<u>Pasos:</u>

1. Listar el punto de montaje.
2. Listar todos los archivos: <code>ls -R <puntoMontaje></code>. Debería aparecer:

<pre>a.txt b.txt c.txt dir1 directo.bin indirecto-simple.bin otro-indirecto-simple.bin lost+found

mnt/dir1:
dir2 hola.txt

mnt/dir1/dir2:
dir3 hola.txt

mnt/dir1/dir2/dir3:
dir4 hola.txt

mnt/d1r1/dir2/dir3/dir4:</pre>
3. Intentar listar un subdirectorio inexistente.
4. Intentar acceder a un subdirectorio inexistente, usando el comando cd <path>
</td></tr>
</table>

<table>
<tr><th>Prueba 2</th><th>Creación y Eliminación de Archivos y Directorios</th></tr>
<tr><td>Desarrollo</td><td>
Esquema: Sin Cache (o buscar la forma en que no impacte, tal vez usando el libredummy_engine.so provisto por la cátedra)

Archivo de Volumen: 1kb-3mb.disk

<u>Pasos:</u>

1. Crear un directorio con “<code>mkdir dir2</code>”.
2. Corroborar que existe y ingresar al mismo usando el comando cd dir1
3. Crear muchos directorios: “<code>mkdir -p a/b/c/d/e f/g/h/i/j</code>”
4. Corroborar su creación: “<code>ls -R a f</code>” (o <code>tree</code> si está instalado)
5. Eliminar un directorio: “<code>rmdir dir1</code>” y comprobar su borrado.
6. Eliminar una rama de los directorios: “<code>rm -vr a</code>” y comprobar su borrado.

7. Ingresar al directorio <code>f/g/h/i/j</code>
8. Crear archivos vacíos: “<code>touch file1 file2 file3</code>”
</td></tr>
</table>

	9. Comprobar su existencia y su tamaño ejecutando: `ls -lh` 10. Eliminar los archivos: `rm file1 file2` y comprobar su borrado. 11. Montar el disco con ext2 nativo y corroborar que el disco haya quedado en el estado adecuado.

Prueba 3	**Límites en Creación de Archivos y estado SuperBloque**
Desarrollo	**Esquema:** Sin Cache (o buscar la forma en que no impacte, tal vez usando el libredummy_engine.so provisto por la cátedra) **Archivo de Volumen:** 1kb-3mb.disk **<u>Pasos:</u>** 1. Anotar la cantidad de inodos libres en el disco: `./check-free-blocks-and-inodes.sh <disco>` 2. Crear dos archivos 3. Revisar la cantidad nuevamente, debería haberse reducido en dos (si usan mmap, podría no estar actualizado y habría que desmontar primero) 4. Crear archivos hasta llegar al máximo de archivos: Ejecutar el script: `./create-empty-files.sh <puntoMontaje> 370` 5. No debería poder crearlos todos. Revisar resultado y la cantidad de inodos libres. 6. Crear un archivo hasta llenar el fs `./create-data-file.sh big-file.txt 5000 random` 7. Debería dar un error indicando que no queda espacio (no necesariamente quedarán 0 bloques libres, dado que existen bloques reservados) 8. Montar el disco con el fs nativo y comprobar que no se pueden crear archivos ni queda más espacio para escribir nuevos datos.

Prueba 4	**Lectura de Archivos chicos y medianos**
Desarrollo	**Esquema:** Sin Cache (o buscar la forma en que no impacte, tal vez usando el libredummy_engine.so provisto por la cátedra) **Archivo de Volumen:** 1kb-3mb.disk **<u>Pasos:</u>** 1. Verificar el estado de los archivos, calculando md5sum de: a. ***a.txt*** *(archivo de 0 bytes)* b. ***b.txt*** *(archivo de 0 bytes)* c. ***directo.bin*** *(archivo binario de 12287 bytes)* d. ***indirecto-simple.bin*** *(archivo binario de 262143 bytes)* *<u>Nota:</u> los hash originales se encuentran en el archivo 1kb-3mb.md5sum ubicado en la carpeta disks*

	2. Verificar el size de todos los archivos (con la función stat)

Prueba 5	**Lectura de Archivos Grandes**
Desarrollo	**Esquema:** Sin Cache (o buscar la forma en que no impacte, tal vez usando el libredummy_engine.so provisto por la cátedra) **Archivo de Volumen:** 1kb-300mb.disk **Pasos:** 1. Verificar el estado de los archivos, calculando el md5sum de: a. ***indirecto-doble.bin*** *(archivo de 67383000 bytes)* b. ***indirecto-triple.bin*** *(archivo de 157286400 bytes)* *Nota: los hash originales se encuentran en el archivo 1kb-300mb.md5sum ubicado en la carpeta disks* 2. Verificar el size de todos los archivos (con la función stat)

Prueba 6	**Escritura en Archivos Chicos y Medianos**
Desarrollo	**Esquema:** Sin Cache (o buscar la forma en que no impacte, tal vez usando el libredummy_engine.so provisto por la cátedra) **Archivo de Volumen (a modificar con fuse):** 1kb-3mb.disk **Pasos:** 1. Escribir desde el principio 2000 bytes sobre punteros directos: `./write-data-to-file.sh <mnt>/directo.bin 2000 0 1 zeroed` 2. Cerrar la aplicación fuse y montar el mismo archivo de volumen pero con el fs nativo. 3. Calcular el md5sum del archivo **directo.bin.** Debería ser igual al indicado en el archivo "*disks/modified_1kb-3mb.md5sum*". 4. Levantar fuse nuevamente. Escribir 4000 bytes desde punteros indirectos: `./write-data-to-file.sh <mnt>/indirecto-simple.bin 4000 52 1 zeroed` 5. Repetir los pasos 2 y 3 para el archivo **indirecto-simple.bin**. 6. Levantar fuse nuevamente. Agregar 3000 bytes al final del archivo: `./write-data-to-file.sh <mnt>/otro-indirecto-simple.bin 3000 eof 1 zeroed` 7. Repetir los pasos 2 y 3 para el archivo **otro-indirecto-simple.bin**

	8. Validar el tamaño del archivo **otro-indirecto-simple.bin** usando el comando **stat:** Debería ser: 265143 bytes.

Prueba 7	**Escritura en Archivos Grandes**
Desarrollo	**Esquema:** Sin Cache (o buscar la forma en que no impacte, tal vez usando el libredummy_engine.so provisto por la cátedra) **Archivo de Volumen:** 1kb-300mb.disk **Pasos:** 1. Escribir 1 byte desde un puntero doblemente indirecto: `./write-data-to-file.sh <mnt>/indirecto-doble.bin 1 60000000 1` 2. Cerrar la aplicación fuse y montar el mismo archivo de volumen pero con el fs nativo. 3. Calcular el md5sum del archivo **indirecto-doble.bin.** Debería ser igual al indicado en el archivo "disks/*modified_1kb-300mb.md5sum*" 4. Escritura de 44 Mb sobre el archivo de 150 Mb (indirección triple): `./write-data-to-file.sh <mnt>/indirecto-triple.bin 32768 4785 1408` 5. Repetir los pasos 2 y 3 con el archivo **indirecto-triple.bin**. 6. Validar el tamaño del archivo **indirecto-triple.bin** usando el comando **stat** ○ Debería ser: 202926356 bytes.

Prueba 8	**Truncar Archivos**
Desarrollo	**Esquema:** Sin Cache (o buscar la forma en que no impacte, tal vez usando el libredummy_engine.so provisto por la cátedra) **Archivo de Volumen:** 1kb-3mb.disk **Pasos:** 1. Ejecutar el comando `truncate trunk.file -s 0` (crea el archivo si no existe) ○ Validar el **md5sum:** d41d8cd98f00b204e9800998ecf8427e ○ Validar el size con el comando ***stat*** 2. Ejecutar el comando `truncate trunk.file -s 12000` ○ Validar el **md5sum:** 21d9938f335c6bfab0eeaed58673b073 ○ Validar el size con el comando ***stat*** 3. Ejecutar el comando `truncate trunk.file -s 3000` ○ Validar el **md5sum:** 0efa007088f326bbc072c34315f3edb8 ○ Validar el size con el comando ***stat*** 4. Cerrar la aplicación fuse y montar el mismo archivo de volumen pero con el fs nativo. ○ Comprobar que el size y el md5sum sean iguales a los del punto 3.

Prueba 9	**RFS - Concurrencia**
Desarrollo	**Esquema:** Sin Cache (o buscar la forma en que no impacte, tal vez usando el libredummy_engine.so provisto por la cátedra) **Archivo de Volumen:** 1kb-300mb.disk; 4kb-400mb.disk **Máximo de threads del RFS:** 20 **Nota**: Montar el fs a través de fuse desde 3 clientes en *3 VMs distintas* **Pasos:** 1. Montar el archivo 1kb-300mb.disk 2. Ejecutar el siguiente script de lectura en las tres PCs, **al mismo tiempo**, dando una ruta del log **ubicada afuera del fs de fuse**: `./conc-file-reader.sh 10 indirecto-triple.bin 5 0 conc-read.log` 3. Esperar a que la ejecución termine. Se puede hacer ejectuando el comando: `watch "ps -fea | grep md5sum"` 4. Revisar que en el log de cada PC, ocurra el mismo resultado, ejecutando: `uniq conc-read.log` Debería aparecer una única línea en el archivo de cada PC, esto significa que todas las lecturas del mismo archivo dieron el mismo resultado siempre. 5. Montar el archivo 4kb-400mb.disk 6. Ejecutar el siguiente script de escritura en las tres PCs, **al mismo tiempo**, dando una ruta del log **ubicada afuera del fs de fuse**: `./conc-file-creator 10 10240 puntoMontaje prefix_pc > log` Nota: "prefix_pc" debe tener un prefijo distinto en cada PC, ej: prefix_pc1, etc 7. Esperar a que la ejecución termine. 8. Revisar que en el log de cada PC. En el mismo se encontrará por cada archivo escrito, el md5sum original y el leido posteriormente (todos deberían dar OK) 9. Montar el archivo 1kb-300mb.disk 10. Ejecutar el siguiente script de escritura en las tres PCs, **al mismo tiempo**: `PC1: ./write-data-to-file.sh mnt/indirecto-triple.bin 32768 0 640` `PC2: ./write-data-to-file.sh mnt/indirecto-triple.bin 32768 960 640` `PC3: ./write-data-to-file.sh mnt/indirecto-triple.bin 32768 1280 640` 11. Cada ejecución escribe 20mb de ceros, desde distintos lugares, solapandosé. 12. Al terminar, debería dar un **md5sum**: 947b1061c662ee661410b77ce669d812

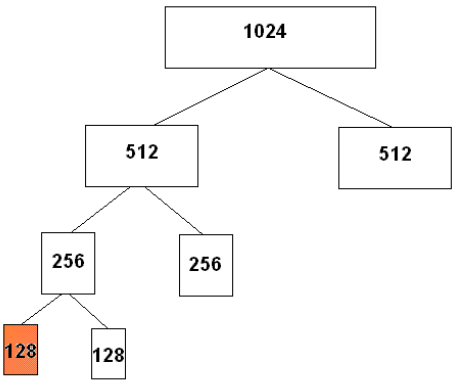
Prueba 10	**Engine Memcached (Uso Básico)**
Desarrollo	**Esquema:** Memcached aislado, levantado con el engine del grupo **Memcached:** mem-size: 1 mb; chunk-size: 16 b, algoritmo de ubicación/reemplazo: cualquiera **Pasos:**

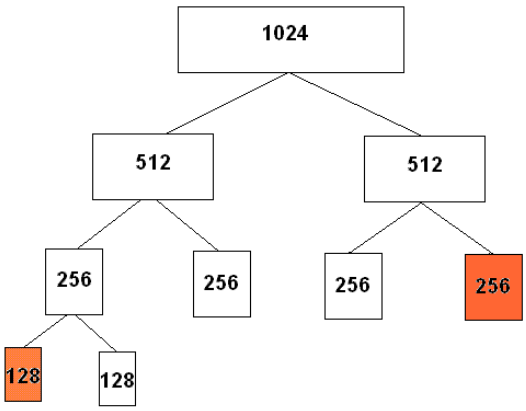
	1. Almacenar 5 (cinco) strings distintos: `./mset key1 string1 [ip] [puerto]` `./mset key2 string2 [ip] [puerto]` `<etc>` 2. Obtener cada uno de los tres valores y comprobar que son los mismos `./mget key1 [ip] [puerto]` => string1 `<etc>` 3. Borrar las key1 y key2 y realizar un nuevo get para comprobar el borrado `./mdelete.sh key1 [ip] [puerto]` (idem para key2) 4. Realizar un flush de todo y comprobar que no quedó ninguna key `./mflush.sh [ip] [puerto]`

Prueba 11	**Engine Memcached - Particiones Dinámicas**
Desarrollo	**Esquema:** Memcached aislado, levantado con el engine del grupo **Memcached:** mem-size: 1 kb; chunk-size: 16 b; algoritmo-remplazo: FIFO; parámetro de búsquedas fallidas: 1, algoritmo-ubicación: cualquiera **<u>Pasos:</u>** 1. Crear un archivo de 128 bytes `./create-data-file.sh partition.file 128 random` Almacenar 8 (ocho) claves, todas con el mismo archivo de contenido. `./mset_from_file clave1 partition.file` `./mset_from_file clave2 partition.file` `<etc>` ***Mientras tanto, ir realizando dumps de la caché*** *en los distintos estados: 0,1,4 y 8 particiones ocupadas* (con la 8va se llenará la caché) 2. Validar que los dumps de la memoria muestren lo esperado en cada caso. 3. Eliminar las particiones: clave1, clave3, clave5 4. Validar que los dumps de la memoria muestren lo esperado en cada caso. 5. Crear un archivo de 368 bytes y almacenarlo 6. Debería haber realizado una compactación, quedando *una* partición libre de 16 bytes (validar con un dump)

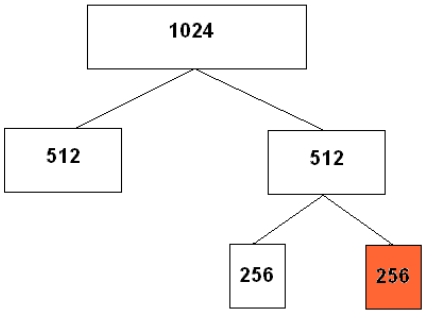
Prueba 12	**Engine Memcached - Algoritmos de Ubicación / Reemplazo**
Desarrollo	**Esquema:** Memcached aislado, levantado con el engine del grupo **Memcached:** mem-size: 1 kb; chunk-size: 16 b; Particiones Dinámicas, algoritmo de reemplazo: FIFO; algoritmo de ubicación: best-fit / worst-fit **Archivo de Volumen:** ninguno **<u>Pasos:</u>** 1. Crear y almacenar 8 particiones de 128 bytes

	2. Validar que los dumps de la memoria muestren lo esperado. 3. Almacenar una nueva partición de 128 bytes `./mset_from_file clave9 partition.file` 4. Por estar en FIFO como algoritmo de partición, debería reemplazarse la partición con *clave1* (validar con un dump) 5. Repetir los pasos del 1 al 3, pero con LRU. 6. Realizar una lectura de las particiones clave1 y clave2 `./mget_to_file clave1 /dev/null` `./mget_to_file clave2 /dev/null` 7. Almacenar nuevamente una partición de 128 bytes `./mset_from_file clave10 partition.file` 8. Debería reemplazarse la partición con *clave3* (validar con un dump) 9. Eliminar las particiones clave4 y clave6 10. Crear y Almacenar particiones de 50 y 100 bytes 11. Crear y Almacenar una partición de 20 bytes 12. Ejecutar un dump y validar que se ubicó de acuerdo al algoritmo adecuado (best-fit / worst-fit).

Prueba 13	**Engine Memcached - Buddy System**
Desarrollo	**Esquema:** Memcached aislado, levantado con el engine del grupo **Memcached:** mem-size: 1 kb; chunk-size: 16 b; del-algoritmo: FIFO; **Archivo de Volumen:** ninguno **Pasos:** 1. Crear y almacenar una partición de 125 bytes. 2. Realizar un dump de la cache:  3. Crear y almacenar tres particiones de 256 b, eliminar dos de ellas a fin de obtener la siguiente distribución (verificar con otro dump):



4. Eliminar el dato almacenado en el punto 1.
5. Realizar un dump de la cache:



Prueba 14	**Engine Memcached - Concurrencia**
Desarrollo	**Esquema:** Memcached aislado, levantado con el engine del grupo **Archivo de Volumen:** ninguno **Memcached:** mem-size: 10 mb; chunk-size: 32 b; del-algoritmo: FIFO; parámetro de búsquedas fallidas: 1 **Pasos:** 1. Se ejecutará un script para levantar N procesos concurrentes que realizarán gets, sets y deletes contra memcached, usando claves aleatorias que probablemente se repitan, y usando valores de tamaño aleatorio siguiendo un patrón pre-establecido. 2. Ejecutar el siguiente script: `./hit-memcached.sh 10 30 1 3 3` 3. Esperar un rato hasta que hayan terminado todos los threads (consultar con el comando top si es necesario) 4. Ejecutar un dump de la caché para ver el estado final (no se espera ningún estado específico acá) 5. Elegir al azar tres claves que estén almacenadas, y pedir su contenido. El formato de la clave indica el contenido, ejemplo: **KEY-C_3** debería tener el valor "**CCC**" 6. Repetir todos los pasos anteriores, con el siguiente script:

	`./hit-memcached.sh 100 20 random 5 100` 7. Validar resultados de igual forma que en el paso 5

Prueba 15	**Cacheo de lectura de datos**
Desarrollo	**Esquema:** Sistema con Memcached levantado con el engine del grupo **Memcached:** mem-size: 10 mb; chunk-size: 1024 b; del-algoritmo: FIFO; parámetro de búsquedas fallidas: 1 **Tiempo de retardo operaciones RFS:** 0 segundos **Archivo de Volumen:** 2kb-100mb.disk **Pasos:** 1. Crear en el fs un archivo de 5 Mb (5242880 bytes) 2. Copiar el archivo afuera del fs y medir su ejecución (esto debería cachear todos los datos): `time cp <archivoOrigen> <archivoDestinoFueraDeFuse>` 3. Modificar el tiempo de retardo en runtime, seteándolo en 3 segundos ***Nota:*** *se sugiere 3 segundos, pero modificarlo de ser necesario según convenga de acuerdo al tiempo relativo de copia.* 4. Volver a copiar el archivo afuera del fs (con time, igual que antes) 5. Debería tardar aproximandamente lo mismo que antes 6. Borrar toda la caché: `./mflush.sh` 7. Volver a copiar el archivo afuera del fs 8. Debería tardar mucho más

Herramientas para verificar el estado del FileSystem:

- **Terminal Linux:**
  - ***dumpe2fs <file>****: brinda información de un fs*
  - ***time <comando>****: ejecuta un comando e indica cuanto tiempo duró su ejecución*
  - ***stat <file>***: tamaño del archivo
  - ***ls***: listado de directorios
  - ***tree***: lista directorios recursivamente y los muestra en forma de árbol
  - ***md5sum <file>***: genera un código md5 para el archivo indicado
  - ***md5sum <file1> <file2> … <file_n> -c <file_md5>:*** compara el checksum de los archivos pasados con los que contiene el archivo ***file_md5***
  - ***pidof <nombreProceso>*** : retorna el pid del proceso.