**Proyecto Final: Simulador del manejo de memoria virtual paginada**

[1. Objetivos y descripción breve del proyecto](#_s5cqcwwwxx3n)

[2. Estructuras de datos](#_5qld8ehzhzv3)

[3. Información de los servicios que el cliente solicita al servidor](#_x0cbnrdh3068)

[4. Ejemplo detallado](#_oqy0gba5foxs)

5. [Asignación de políticas de memoria por equipo](#_s3zm3vq72e4x)

[6.. Lenguaje para programar el servidor](#_8zmsy8syyiol)

[7. Entregables](#_ujuiwgapnvib)

[8. Rúbrica de evaluación](#_j0xnf4igd1gm)

### Objetivos y descripción breve del proyecto

Este proyecto de programación tiene como objetivo simular una de las principales partes de un sistema operativo: el manejo de la memoria virtual paginada, y medir su rendimiento.

El simulador debe ser programado como un servidor que corre en una terminal linux de tu laptop, de manera similar al que se vió en la reciente práctica “cliente-servidor” del laboratorio. Un programa cliente que te será proporcionado y que correrá en otra terminal de tu laptop, enviará al servidor solicitudes de servicio análogas a “system calls”, tales como cargar procesos a memoria; accesar o modificar localidades de la memoria virtual de un proceso que el simulador/servidor tendrá que traducir a direcciones reales de memoria; o liberar la memoria y área de swapping ocupada por un proceso cuando éste termina. Para verificar el manejo correcto de la memoria, el servidor deberá desplegar periódicamente el estado de los marcos de página reales y “swappeados” al área de swapping y otros datos que serán descritos más adelante.

Dado que el proyecto es solo una simulación, no requiere ninguna creación o ejecución de procesos real.

A cada equipo de 4 personas le tocarán dos estrategias de reemplazo de páginas *parecidas* a las que se ven en el curso, o bien alguna nueva ideada por los instructores. Al final de la simulación se deben imprimir varias métricas de desempeño por proceso y globales, entre otras una comparación del rendimiento de las dos políticas de reemplazo asignadas.

Para ayudarte a programar tu servidor, en la sección 6 de este documento encontrarás ligas a un cliente y un servidor muy sencillos programados en Python, similares a los vistos en la práctica de laboratorio; y para ayudarte a probar tu servidor que simula el manejo de memoria virtual paginada, dentro de unos días podrás bajar un segundo cliente que enviará a tu programa un conjunto de solicitudes de servicio parecidas a las que aparecen en el ejemplo de la sección 4 de este documento.

Unos días antes de la fecha de entrega, cuando ya debes tener todo probado, otro cliente enviará a tu servidor solicitudes aleatorias que simulan una situación real.

Hay varias simplificaciones con respecto a un sistema operativo y manejador de memoria reales:

* La computadora tiene una sola CPU y un solo espacio de memoria real;
* Se puede ignorar el tiempo de cambio de contexto;
* En la realidad, el sistema operativo puede estar recibiendo solicitudes de servicio de varios usuarios casi al mismo tiempo, que provienen de varios procesos-cliente; sin embargo para esta simulacióń no interesa saber qué usuario envía las solicitudes y asumimos que todas ellas vienen de un solo proceso cliente.
* No ocurren solicitudes simultáneas, por ejemplo que exactamente al mismo tiempo el servidor reciba dos diferentes solicitudes.
* El tamaño de la memoria real en la simulación es mucho más pequeño que lo usual.
* El área de swapping tiene un tamaño fijo, aunque en la realidad suele estar limitado por el espacio libre en el disco duro. *Tanto en la realidad como en la simulación, si se llenan la memoria real y el área de swapping, se suspende la creación de nuevos procesos hasta que alguno termine.*
* Al comenzar un proceso, se cargan de inmediato a memoria principal todas las páginas que va a requerir; o si el tamaño del proceso excede la memoria real disponible, el resto de las páginas necesarias se cargan al área de swapping. Esta operación no genera “page faults”, al contrario de como se ha visto en clase.

### Estructuras de datos

Para la simulación, su programa podrá contar por ejemplo con las siguientes estructuras de datos:

* Un área de memoria ***M*** contigua que simulará la parte de la memoria real de una computadora reservada a marcos de página. *Por ejemplo*, puede ser un arreglo o lista de marcos de página, donde cada marco incluiría los datos que ustedes estimen convenientes
* Una segunda área de memoria ***S*** contigua, que simulará el área de disco reservada para el swapping de marcos de página
* Otras estructuras de datos que encuentren ustedes convenientes o indispensables para lograr la simulación y que se encuentran en áreas de memoria distintas de las áreas **M** y **S.**

### Información de los servicios que el cliente solicita al servidor:

Sigue a continuación el protocolo - es decir los mensajes que el cliente estará enviando al servidor y las respuestas correspondientes.

Los siguientes mensajes le llegan al servidor al principio de la simulación, y sirven para inicializar parámetros de la simulación: tamaño en kilobytes de la memoria real y del área de swapping, esta última desde luego mucho mayor que la real; y tamaño fijo de las páginas (reales y virtuales) en kilobytes.

**RealMemory** m // tamaño de la memoria real, en k’s

**SwapMemory** n // tamaño del área de swapping, en k’s

**PageSize** p // en bytes

Respuestas del servidor al cliente: después de cada uno de estos comandos, el servidor regresa al cliente un mensaje que indica qué fué lo que se recibió; por ejemplo “Recibido - real memory tal y tal”. El cliente siempre despliega (es decir, envía a stderr) el mensaje enviado al servidor, y el mensaje-resultado que el servidor envía de regreso al cliente

Una vez concluida la inicialización, el cliente podrá enviar al servidor cualquiera de los siguientes comandos:

**PolíticaMemory** *mm*  // la política de memoria a simular. Será el primer comando una vez concluída la inicialización.

Abreviaciones de políticas de memoria que les pueden tocar: xxx (las que les asignemos más adelante en la sección 5)

A continuación otros comandos que pueden venir en cualquier orden:

**P n p**  (cargar un proceso)  
Se trata de una solicitud de “n” bytes para cargar un proceso a la memoria.

“p” es un número entero arbitrario que indica el identificador de proceso.

Ejemplo:

**P 534 1 (Se solicita asignar 534 bytes al proceso 1)**  
  
NOTA En esta versión de memoria virtual con paginación, un proceso se carga completo en una sola solicitud al manejador de memoria, incluyendo páginas que contienen el código del proceso y otras para su área de datos. La longitud máxima de un proceso **puede** exceder el tamaño de la memoria real. *Observen que esto es diferente a la forma como se revisó en clase, donde aprendimos que los procesos se pueden cargar inicialmente a memoria por partes en base a un esquema por demanda.*  
PROCESS: El programa deberá asignar los marcos de página necesarios, *no necesariamente contiguos, pues podría no encontrarse en la memoria real un área contigua de la longitud requerida.* **Suponiendo a manera de ilustración que cada página midiera 16 bytes***,* 534 bytes requerirían 34 marcos [(534/16) = 33.375)]. Si no hubiera marcos de página vacíos suficientes en **M***,* se tendrán que swappear-out las páginas contenidas en algunos marcos ya ocupados, de acuerdo con el algoritmo de reemplazo que se les haya asignado. *Las páginas a reemplazar podrían pertenecer a cualquier proceso; es decir, se utiliza un enfoque de “reemplazo global”*. Si hay swapping, su programa deberá registrar dónde queda cada página en el área **S** de swapping, pues posteriormente puede ser necesario volver a cargar esas páginas. Además puede ser necesario registrar el “*timestamp”* u otra información como podría ser el *“bit de referencia”* y *“bit de modificación”* al momento que se asigna o se utiliza en memoria cada página del proceso, en concordancia con lo que requiera la estrategia de reemplazo que se esté utilizando.

El *Tiempo y el Timestamp* deben iniciar en cero al comenzar la simulación, y después de cada comando actualizarse de acuerdo al tiempo real transcurrido.  
  
*Hint.* Para facilitar su debugging, podrían registrar en las áreas **M** y **S** o en alguna otra estructura de datos al menos el número del proceso y número de página virtual de ese proceso que está cargado en cada marco.

OUTPUT: Imprimir el comando de INPUT y el estado de la memoria real y del área de swapping una vez cargado el proceso, como se indica más adelante en el ejemplo de la sección 4.

**A d p m**

Es una solicitud para accesar la dirección virtual “d” del proceso “p”. Si “m” es 0, la dirección correspondiente sólo se lee; si es 1, también se modifica.“d” puede tener un valor desde cero hasta la dirección virtual máxima del proceso expresada en bytes.

Ejemplo:

**A 17 5 0 (accesar para lectura la dirección virtual 17 del proceso 5)**  
PROCESS: Localizar dónde se encuentra la página que contiene esa dirección. En el ejemplo de páginas de 16 bytes, se trata de la página 1 del proceso, pues la longitud de página es 16 y las páginas de un proceso se numeran desde cero. Esa página 1 puede estar en algún lugar de la memoria real, y no necesariamente junto a la página 0; o bien puede estar en el área de swapping pues ya la habíamos “corrido” de su marco en memoria real en alguna operación anterior. Si ese es el caso, ha ocurrido un “*page fault”*, y es necesario cargarla de nuevo (swapping-in) en alguna parte de la memoria real, para lo cual podría ser necesario swappear-out algún otro marco de página si es que todos están ocupados (si no hay marcos libres). Nuevamente, podría ser necesario registrar en alguna parte el *“timestamp”* en que ocurre el acceso o alguna otra información pertinente.  
  
OUTPUT: El comando de INPUT; la dirección en memoria real correspondiente a la dirección virtual “d” del proceso “p” y el estado de la memoria real y del área de swapping como se indica en el ejemplo más adelante.

**L p**

Liberar las páginas del proceso “p”.

PROCESS: Se liberan todas las páginas del proceso “p”, tanto las que estaban en memoria real como aquellas que se encontraban en el área de swapping, quedando varios marcos de página o pedazos del área de swapping vacíos y disponibles para otras operaciones.

OUTPUT: El comando de INPUT y el estado de las áreas M y S

**C**

comentario. Es una línea de comentarios. Desplegarla

**F**   
Fin. Es siempre la última línea de un conjunto de especificaciones; **pero pueden seguir otras líneas más para otro conjunto de solicitudes, empezando por el comando que indica la política.** Habrá al menos dos especificaciones, una para cada política de reemplazo.

OUTPUT: El comando de INPUT; reporte de estadísticas en el formato indicado más adelante en el ejemplo de la sección 4, que incluye:

- *turnaround time* de cada proceso que se consideró, desde que se comienza a cargar un proceso (P) hasta que se terminan de liberar todas sus páginas (L). Puedes obtenerlo por medio de una diferencia de timestamps.

- *número de page faults por proceso*. Recuerda que un page fault ocurre únicamente cuando un marco de página necesario requerido por el comando “A”, no se encuentra en memoria real. → *En esta versión del manejador de memoria virtual, el comando P (cargar un proceso completo a memoria real) no produce page faults.*

- *número total de operaciones de swap-out y de swap-in* que fueron necesarias por cualquier motivo, por cada proceso. Requerir alguna página del área de swapping, hace un swap-in. Un proceso no se carga de esa área, sino de otra parte que no interesa. Entonces no causa swapins. Una pagina que sale de memoria rumbo al swapping, 1 swapout

- *rendimiento = 1 - numero de fallas/numero de comandos A para cada proceso*

*-turnaround promedio*

*- rendimiento de todos los procesos: numero total de page faults/numero de comados A*

**E**  
Exit. Última línea del archivo.

PROCESS: se termina la simulación

OUTPUT: El comando de INPUT y mensaje de despedida...

Algoritmos o políticas de reemplazo: A cada equipo de 4 alumnos definido para el semestre, le serán asignados dos algoritmos de reemplazo de los vistos en clase, o bien alguno nuevo definido por los instructores. Algunos de estos algoritmos hacen uso de timestamps o de los bits o contadores de referencia/modificación y otros no. Deberán correr toda su simulación dos veces, cada una con uno de los algoritmos de reemplazo que les tocaron.

Sigue a continuación un ejemplo completo más detallado, especificando el formato del output en forma de renglones de tabla, y al final la tabla completa que junta todos los renglones.

### Ejemplo detallado

1. Ejemplo de input y output con el algoritmo de reemplazo: “Reemplaza una página contenida en un marco aleatorio”. No muy efectivo pero muy simple. Input en **bold**. Los //comentarios o C comentarios no son parte de su output. Todo lo demás sí

**C comenzamos…**C comenzamos…

**RealMemory** 2 // tamaño de la memoria real, en k’s

RealMemory 2 // el servidor siempre regresa al cliente el comando que recibió, y  
 // el cliente lo despliega

**SwapMemory** 4 // tamaño del área de swapping, en k’s

SwapMemory 4

**PageSize** 16 // en bytes

PageSize 16

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo | comando | dir. real | M | S | Terminados |
| 0.000 |  |  | M[0-127:L] | S[0-255:L] |  |

// Es el output después de la inicialización, M[0-127:L] representa marcos de página libres desde el 0 al 127 (128\*16=2048=2k). Es el renglón de una tabla, que debe irse “juntando” con otros para después desplegar la tabla completa.  
  
**P 2048 1 ignora cualquier comentario en la misma línea del comando, como éste.**P 2048 1Asignar 2048 bytes al proceso 1

//¡se llenó la memoria!

// como es el primer proceso, se cargó en memoria contigua

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo | comando | dir. real | M | S | Terminados |
| 0.000 | P 2048 1 |  | M[0:1.0], M[1:1.1] (...) M[127:1.127] | S[0-255:L] |  |

// M[r:p.v] representa que en el el marco real **r** está cargado el marco virtual **v** del proceso **p**. Si su política de reemplazo requiriera registrar un tiempo **t**, o un bit de modificación **m** o un bit de acceso **a** o un contador **c**, se incluirían ahí mismo en la forma **M[r:p.v;t,m,a,c]**. En su output aparecerán todos los marcos del proceso 1 en el área M; aquí pusimos (...) para abreviar espacio.

El tiempo registrado en la primera columna de la tabla indica el momento en que se recibió el comando. Los tiempos que siguen enseguida para los próximos comandos son arbitrarios y pueden ser muy diferentes a los que se reciban en la secuencia de comandos que les toque resolver

**A 1 1 0**

A 1 1 0

Obtener la dirección real correspondiente a la dirección virtual 1 del proceso 1Dirección virtual: 1. Dirección real: 1

// el proceso 1 comienza a ejecutar…

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo | comando | dir. real | M | S | Terminados |
| 0.001 | **A 1 1 0** | 1 | igual | igual |  |

// “igual” significa que el estado de M y S permanece sin cambios. También puede ponerse como “=”

**A 33 1 1**

A 33 1 1

Obtener la dirección real correspondiente a la dirección virtual 33 del proceso 1

y modificar dicha dirección  
Dirección virtual: 33. Dirección real: 33

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo | comando | dir. real | M | S | Terminados |
| 0.002 | **A 33 1 1** | 33 | igual | igual |  |

// en la política que estamos considerando, no interesa conocer si una página ha sido //modificada o no; en caso de una política como MRM (most recently modified), habría que //registrar el bit de modificación desde el primer comando, y lo que se desplegaría en el área M //sería: M[0:1.0; ,0], M[1:1.1; ,1], M[2:1.2; ,0] etc. indicando que no interesa el tiempo (“,”) y que //el bit de modificación es 0 para la página 0 del proceso, 1 para la 1 etc.

**P 32 2**

P 32 2

Asignar 32 bytes al proceso 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo | comando | dir. real | M | S | Terminados |
| 0.003 | **P 32 2** |  | ==  M[5:2.0], ==  M[78:2.1] == | S[0:1.5], S[1:1.78], S[2-255:L] |  |

// la memoria está llena con las páginas del proceso 1; habrá que hacerle hueco al proceso 2 **//** página 5 (obtenida aleatoriamente) del proceso 1 swappeada al marco 0 del área de //swapping   
// página 78 (idem) del proceso 1 swappeada al marco 1 del área de swapping

// (para hacer hueco y poder cargar el proceso 2 pues no había ningún marco libre).

// 5 y 78 son dos números aleatorios de acuerdo a algoritmo de reemplazo  
// como el proceso 1 estaba en memoria contigua

// empezando desde 0, los marcos de página obtenidos aleatoriamente coinciden  
// con los números de página del proceso

// Se asignaron los marcos de página 5 y 78 al proceso 2

// == significa que lo que estaba por ejemplo antes de M[5:2.0] no cambió.

**A 15 2 0**

A 15 2 0

Obtener la dirección real correspondiente a la dirección virtual 15 del proceso 2  
Dirección virtual:15. Dirección real: 95

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo | comando | dir. real | M | S | Terminados |
| 0.004 | **A 15 2 0** | 95 | igual | igual |  |

// es la página 0 del proceso 2, que se encuentra en el marco 5

// Este empieza en la dirección 16\*5 = 80. Más 15 = 95

**A 82 1 0**

A 82 1 0

Obtener la dirección real correspondiente a la dirección virtual 82 del proceso 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo | comando | dir. real | M | S | Terminados |
| 0.005 | **A 82 1 0** | 1282 | ==  M[5:2.0], ==  M[78:2.1] ==  M[80:1.5] == | S[0:L], S[1:1.78], s[2:1.80], S:[3-255:L] |  |

// Dirección virtual 82 del proceso 1 corresponde a la página virtual 5 del proceso (82/16=5 y //sobran 2)

// Se localizó la página 5 del proceso 1 que estaba en la posición 0 de swapping

// Como no había espacio en M  
// para cargar esa página 5 fué necesario antes sacar otra, y resultó  
// aleatoriamente la que estaba en el marco 80. Ese marco 80 ya

// estaba ocupado por otra página del mismo proceso 1 que fue necesario   
// swappear al marco 2 de **S**, quedando S como S[0:L], S[1:1.78], s[2:1.80], S:[3-255:L]

// y la página virtual 5 de 1 quedó en M en el marco 80 que se liberó.

// y se cargó al marco 80.

// Dirección virtual: 82**.** Dirección real: 1282 (80\*16+2)  
// Posición 0 del swapping queda libre y puede reutilizarse después si es necesario.   
  
**L 2**

L 2

Liberar los marcos de página ocupados por el proceso 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo | comando | dir. real | M | S | Terminados |
| 0.006 | **L 2** |  | ==  M[5:L], ==  M[78:L], ==  M[80:1.5], == | igual | 2 |

**//** Se liberan los marcos de página de memoria real: [5, 78]

**P 32 3**

P 32 3

Asignar 32 bytes al proceso 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo | comando | dir. real | M | S | Terminados |
| 0.007 | **P 32 3** |  | ==  M[5:3.0], ==  M[78:3.1] ==  M[80:1.5] == | igual |  |

//Se asignan los marcos [5, 78]

// pues justo los acabamos de liberar

**L 1**

L 1

Liberar los marcos de página ocupados por el proceso 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo | comando | dir. real | M | S | Terminados |
| 0.008 | **L 1** |  | M[0-4:L]  M[5:3.0],  M[6-77:L]  M[78:3.1]  M[79-127:L] | S[0-255:L] | 2,1 |

**//**Se liberan los marcos de memoria real: 0-4, 6-77 y 79-127

//Se liberan los marcos [1,2] del área de swapping. El 0 ya estaba libre, por tanto toda el área //de swapping queda libre.

**F**F   
//Fin.

Queda volando el proceso 3, que no terminó. En este momento se libera los marcos 5 y 78 que ocupaba, quedando libres las áreas M y S completas:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo | comando | dir. real | M | S | Terminados |
| 0.009 | **F** |  | M[0-127:L] | S[0-255:L] | 2,1,3 |

Se despliega enseguida la tabla completa, y luego el reporte de salida (Ver la instrucción F).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tiempo** | **comando** | **dir. real** | **M** | **S** | **Terminados** |
| 0.000 |  |  | M[0-127:L] | S[0-255:L] |  |
| 0.000 | P 2048 1 |  | M[0:1.0], M[1:1.1], (...), M[127:1.127] | igual |  |
| 0.001 | **A 1 1 0** | 1 | igual | igual |  |
| 0.002 | **A 33 1 1** | 33 | igual | igual |  |
| 0.003 | **P 32 2** |  | ==, M[5:2.0], ==, M[78:2.1], == | S[0:1.5], S[1:1.78], S[2-255:L] |  |
| 0.004 | **A 15 2 0** | 95 | == | == |  |
| 0.005 | **A 82 1 0** | 1282 | ==, M[5:2.0], ==, M[78:2.1], ==, M[80:1.5] == | S[0:L], S[1:1.78], s[2:1.80], S:[2-255:L] |  |
| 0.006 | **L 2** |  | ==, M[5:L], ==, M[78:L], ==,  M[80:1.5], == | == | 2 |
| 0.007 | **P 32 3** |  | ==, M[5:3.0], ==, M[78:3.1] , ==, M[80:1.5], == | == |  |
| 0.008 | **L 1** |  | M[0-4:L], M[5:3.0], M[6-77:L], M[78:3.1], M[79-127:L] | S[0-255:L] | 2,1 |
| 0.009 | **F** |  | M[0-127:L] | S[0-255:L] | 2,1,3 |

<aquí va otra tabla con los resultados de fallas por proceso y totales y rendimiento. Ver especificación del comando **F>**

**formato de la tabla:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **proceso** | **turnaround** | **# page faults** | **swapins** | **swapouts** | **reendimiento** |
| **1** |  |  |  |  |  |
| **2** |  |  |  |  |  |
| **3** |  |  |  |  |  |

(**OJO!!** → puede continuar una nueva serie de solicitudes desde “RealMemory” en adelante. Se re-inicializa todo antes de proseguir. Algunas de las solicitudes que tendrá que manejar su programa incluyen errores intencionales para probar la robustez y resiliencia de su código y la claridad de sus mensajes de error).

### Asignación de políticas de memoria por equipo

Asignación por equipo:

|  |  |
| --- | --- |
| Número de equipo (4 integrantes) | Algoritmo de reemplazo |
| 4.1, 4.4, 4.7, 4.10, 4.13, ….. | FIFO, LIFO, (First in first out y last in first out) |
| 4.2, 4.5, 4.8, 4.11, 4.14, ….. | LRU y MFU (Least recently used [accessed or modified] y Most frequently used). OJO: dice MFU, no MRU. |
| 4.3, 4.6, 4.9, 4.12, 4.15, ….. | LRM y MRM (Least recently modified y most recently modified)  RND (reemplazo aleatorio) no la necesita ningún equipo para la entrega final, excepto para correr el ejemplo y comprobar sus resultados |

En caso de empate entre varias páginas que según su algoritmo pueden reemplazarse, el desempate es por FIFO.

El cargar una página a memoria como resultado del comando P, no genera un uso o modificación de la página. Una página solo se usa o modifica como resultado del comando A.

Un page fault ocurre al tener que enviar una página de M hacia S, como resultado de comandos P o A

Cualquier equipo puede decidir utilizar un algoritmo de reemplazo diferente a los anteriores (nuevo), previa declaración por parte de ustedes y aprobación de sus instructores del curso.

**El cliente final que utilizarán los equipos será dado a conocer dos días antes de la fecha de entrega.** Podrá ser el mismo para todas las estrategias de reemplazo, o diferente para alguna(s). Ya para entonces deberá estar ampliamente probado su programa, de tal forma que no tenga ningún problema al ejecutar este archivo de entrada “sorpresa”. Este archivo podrá incluir datos erróneos, como procesos demasiado grandes, comandos inválidos ETC. *Es responsabilidad de cualquier programa que hagan en su vida el verificar que sus datos de entrada sean correctos*.

### Lenguaje para programar el servidor

Lenguaje *sugerido:* Python o Ruby o Rust o Go u otro lenguaje igualmente poderoso (es decir, más poderoso y sencillo de programar que Java o C++...). Python en particular incluye llamadas para establecer el ambiente cliente/servidor muy similares a las de la práctica de laboratorio. Además de aprender un nuevo lenguaje, aunque no lo creas terminarás más rápido con estos lenguajes más modernos que utilizando algún otro lenguaje que ya conozcas. Python 2.7 viene integrado con UBUNTU y lo puedes ejecutar directo en la terminal:

1. $ python
2. >>> print “hello world”
3. hello world
4. >>> Pid = 1
5. >>> M = [ ]
6. >>> M.append( marco(bla, bla…) ) # (traten de hacer eso en Java en tan pocas líneas…)

Del software package loader pueden instalar Geany, un IDE (Integrated development environment) sencillo para Python y otros lenguajes. Hay decenas de otros IDEs más poderosos, con la usual penalidad de tener que aprender sus numerosas opciones e interpretar extraños mensajes de error que vienen de opciones no utilizadas... Ruby también lo pueden instalar desde el package loader; Rust (creado por Mozilla) y Go (de Google) desde los sitios de esas organizaciones. De los cuatro, el más rico en librerías y más utilizado es Python, seguido de cerca por Ruby gracias al “Ruby on Rails”. Ruby es más orientado a objetos que Python. Go y Rust son más recientes

Si utilizas Python, localiza en <http://python.org> el tutorial de python para la versión del lenguaje que viene instalado en el Ubuntu que tengas:

$ python --version

Puedes bajar de las siguientes ligas un cliente y servidor en python muy básicos, adaptados de los que aparecen en <https://pymotw.com/2/socket/tcp.html> para el protocolo TCP/IP. Son similares a los que hiciste en C para la práctica de laboratorio. Están escritos para python 2.7, que creo que es la que viene por default en Ubuntu. Hay una nueva versión de python, 3.algo, con algunos cambios con respecto a python 2 e incompatible con la versión previa:

Cliente: <https://docs.google.com/document/d/1sLFDBvDXjulqQWFET8kLA7qkk4-7h-_mgr2SIjZL0Co/edit?usp=sharing>

Servidor: <https://docs.google.com/document/d/1PMPBv8IC5r4vqnaETQz7iPufJ8Rd5Tt6agEQjT9Hzgc/edit?usp=sharing>

Para correrlos, teclea primero en una terminal $python ./simulatorSched y luego en otra terminal $python ./clienteSched. Verás que el cliente envía un par de mensajes al servidor/simulador y el servidor todo lo que hace es enviarle una respuesta al cliente. Esos mensajes de prueba no tienen nada que ver con los comandos del simulador que vas a programar.

Como mencionamos antes, dentro de uno dias liberaremos un segundo cliente más realista, que enviará a tu servidor comandos muy parecidos a los del ejemplo y que pueden ayudarte a checar los algoritmos de tu simulador bajo diferentes políticas. Y más adelante un tercer cliente unos días antes del examen final - si tu simulador ha sido probado exhaustivamente, no deberás tener problema para que corra bien con este tercer y último cliente.

### Entregables

**Entregables: Domingo 19 de mayo 2019 a las 18:00 horas:** Un reporte en un USB con:

* 1. Lenguaje y versión del lenguaje que utilizaron. Número total de líneas y número de líneas sin los comentarios. Esto es sólo para propósitos informativos y no afecta para nada su calificación.
  2. El output especificado.
  3. El listado del programa, que deberá utilizar programación modular. Ampliamente documentado mediante comentarios en el mismo código, como sigue. Adapten lo siguiente a la nomenclatura usada por el lenguaje que decidan utilizar.
     1. Los autores del programa
     2. Breve descripción sobre qué hace el programa.
     3. Una tabla comparativa resumen que muestre claramente el comportamiento de la corrida de simulación con cada estrategia de reemplazo, comparando el rendimiento obtenido con cada una de ellas, para el archivo de datos proporcionado por sus instructores del curso.
     4. El significado de cada variable global o local. Nombres de variable muy claros; sin embargo, el comentario sobre el significado preciso de la variable puede ser necesario para no tener que deducirlo a partir de la forma como se utiliza la variable.
     5. La función de cada función, rutina, gem, método o como se llame, así como su estado de entrada y estado de salida. Nombres igualmente muy claros. El estado de entrada incluye nombres y valores de variables globales y argumentos de entrada a la función. El estado de salida, lo mismo al terminar la función. Siéntete *función* y pregúntate: ¿qué es lo que me llega, tanto de argumentos como de la situación global de lo que se está haciendo, incluyendo tal vez parte de su historia? ¿cómo lo dejo?
     6. Comentarios de bloques de código que indican qué hace el bloque, si es necesario. Lo mismo para ciertas líneas individuales, si es necesario.  
          
        Prueba de fuego acerca de la estructura y claridad de su programa: *la prueba de la bicicleta*. No es necesario pensar mucho para darse cuenta de cómo funciona una bici. Basta mirarla. Lo mismo debe ocurrir al echarle una mirada a su programa --cualquier programa que hagan en su vida. *Un programa es una obra literaria; debe estar hecho para leerse con facilidad y aprender, no es solo un conjunto de instrucciones para una máquina.* Incluso hubo hace unos años un influyente movimiento llamado [literate programming].  
          
        En una escala de 1 a 100, un programa que funciona bien vale sólo 30. Uno que funciona ante cualquier conjunto posible de datos y genera mensajes entendibles si algo anda mal, 50. Uno que fácilmente se puede comprender cómo funciona s*in recurrir para nada a los programadores originales*, vale 80. Uno que además está tan bien documentado y modularizado que es posible hacerle rápidamente los cambios más probables a futuro, vale 100. Los mejores programas “open source” son de este último tipo, pues precisamente la idea es que cualquier persona pueda entenderlos rápidamente y contribuir a mejorarlos.  
          
        Su programa debe ser robusto y defenderse ante cualquier error, por ejemplo de input incorrecto, procesos más grandes que lo permitido, direcciones fuera del tamaño del proceso, área de swapping agotada ETC. No son todos… Les sugerimos probar su programa con input aleatorio…  
          
        Para este proyecto en particular, *claridad y robustez mata eficiencia*. Sin exagerar...

### Rúbrica de evaluación:

La simulación contempla contrastar 2 políticas de reemplazo.

1. Evaluaremos presencialmente con cada equipo de trabajo su simulación y el reporte asociado. **Publicaremos un archivo en excel para que separen un horario de presentación de su trabajo el día lunes y martes 20 y 21 de mayo, en períodos de media hora.**
2. Verificaremos el reporte escrito y el USB con su código
3. Evaluaremos, además de los criterios de programación ya explicados en este documento; distintos escenarios utilizando nuestro archivo de datos:
   1. Si no funciona su simulación.
   2. Si funciona sólo para 1 de las 2 políticas de reemplazo.
   3. Si funciona para ambas.
4. **Para los programas que funcionen correctamente al 100%, podrá haber un bono extra de 1 punto sobre los 5 puntos que vale el trabajo final; inapelable y totalmente a juicio de los instructores**, según el grado de cumplimiento de los otros criterios previamente establecidos en este documento.