Objetivos

- Analizar los segmentos de memoria de un proceso en Linux.
- Observación del rendimiento de un programa con respecto al hw y sw que gestiona la memoria virtual.
- Introducción básica de algunas herramientas que reportan información de rendimiento.

Referencias

[1] Tanenbaum, Bos – Modern Operating Systems - Prentice Hall; 4 edition (March 10, 2014) - ISBN-10: 013359162X
[2] Douglas Comer - Operating System Design - The Xinu Approach. CRC Press, 2015. ISBN : 9781498712439
[3] Silberschatz, Galvin, Gagne - Operating Systems Concepts - John Wiley & Sons; 10 edition (2018) – ISBN 978-1-119-32091-3

Ejercicio 1. *Reserva de memoria dinámica en Linux. Evaluación del comportamiento de acceso a memoria virtual.*

El programa de este ejercicio reserva una gran cantidad de memoria virtual en Linux, dinámicamente. Se desea evaluar su comportamiento en cuanto accesos a memoria, y cómo mejora su rendimiento.

En este ejercicio tenga en cuenta la memoria virtual del sistema Linux, los segmentos utilizados por un proceso (establecidos por el compilador), y el hardware involucrado en la memoria virtual (TLB, caché):

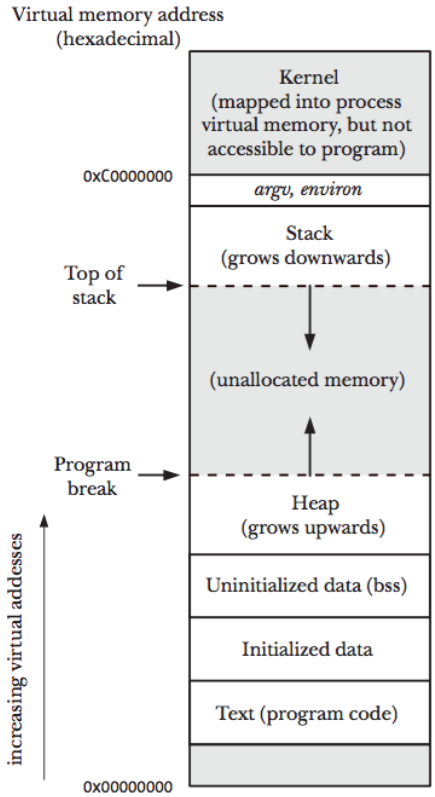

Fig1. Segmentos de memoria de un proceso

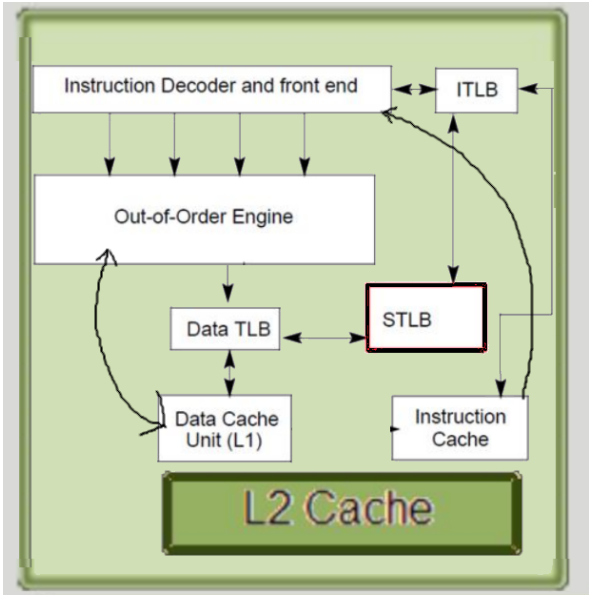

Fig 2.Diagrama *muy* simplificado de un core (cpu)

```
/* programa usamem.c */

#include <stdio.h>
#include <stdlib.h>

#define N 240000
#define BSIZE 4096
#define SMALL 4

char *pp;

main ()
{
        int i, j, k;

        pp = malloc(N*BSIZE);
        if (pp == NULL) {
                printf("Error al reservar memoria.\n");
                exit(1);
        }

        /* RECORREMOS y modificamos todo el segmento solicitado */
        for (j=0; j<BSIZE; j++) {
        for (i=0; i<N; i++) {
                *(pp+i*BSIZE+j) = 2;          // pp[i][j] = 2;
        }
        }

        /* RECORREMOS y VERIFICAMOS QUE HEMOS MODIFICADO todo el segmento solicitado */
        for (i=0; i<N; i++) {
        for (j=0; j<BSIZE; j++) {
                if (*(pp+i*BSIZE+j) != 2) {          // if (pp[i][j] != 2)
                        printf("ERROR! \n");
                        exit(1);
                }
        }
        }

        printf("OK \n");
}
```

Compilar el programa estáticamente (sin bibliotecas dinámicas):

```
gcc -static -O0 -o usamem usamem.c
```

Responder:

a. ¿Cuántos bytes solicita el programa, al sistema operativo, reservar? (a través de malloc).

b. ¿Cuál es el tiempo de ejecución del programa?. Ayuda: comando **time**

c. ¿En qué segmento de memoria del proceso se almacena **la variable puntero pp**? ¿En qué segmento de memoria del proceso se almacenan las variables i,j,k?

d. ¿En qué segmento de memoria del proceso el sistema operativo reserva la memoria solicitada con malloc? (y que es apuntada por pp). Mencione los posibles system calls de Linux que malloc podría utilizar para cumplir su objetivo.

e. Incorpore al programa una sentencia printf que reporte la dirección virtual de la función main(), de i y de j. Nota: tenga en cuenta que las direcciones de memoria virtual en PC son de 64bits.

f. Debajo de /proc/PID/ los sistemas Linux presentan muchos archivos con información relativa a un proceso en ejecución. Ejecute el programa, y en otra terminal, ejecute **cat /proc/PID/maps** (reemplace **PID** por el process id de su programa en ejecución), el cual le mostrará los segmentos de memoria del proceso con direcciones virtuales.

   NOTA: las direcciones reportadas por el archivo maps están en hexadecimal.

   Indique cuáles son las direcciones virtuales de los segmentos de:
   - texto (código máquina ejecutable)
   - datos (variables globales inicializadas y sin inicializar)
   - heap
   - stack

   Indique también cuáles son sus tamaños.

   Observe detenidamente el tamaño del segmento cuyo nombre coincide con el que Ud. respondió en el inciso d. ¿Coincide la cantidad de memoria de ese segmento con la reservada por el programa y calculada en a.?

Ejercicio 2. *Rendimiento de accesos a memoria. Observación de los accesos a la MMU y caché.*

Modifique el programa del ejercicio 1, para contar con 3 versiones del mismo programa.
Las modificaciones son como sigue:

Para la versión 2, reemplace el código en color azul con el siguiente fragmento de código equivalente:

```
/* RECORREMOS y modificamos todo el segmento solicitado */
for (i=0; i<N; i++) {
for (j=0; j<BSIZE; j++) {
        *(pp+i*BSIZE+j) = 2;            //  pp[i][j] = 2;
}
}
```

Para la versión 3, reemplace el código en color azul con el siguiente fragmento de código equivalente:

```
/* RECORREMOS y modificamos todo el segmento solicitado */
for (k=0; k<BSIZE/SMALL; k++) {
for (i=0; i<N; i++) {
for (j=0; j<SMALL; j++) {
        *(pp+i*BSIZE+(SMALL*k+j)) = 2;  // pp[i][SMALL*k+j] = 2;
}
}
}
```

Responder:

a. ¿Cuánto demora la ejecución de cada versión?

b. La herramienta **perf** proporciona un número de contadores de rendimiento útiles que permiten al usuario evaluar el impacto de otros programas en el sistema.

   perf proporciona estadísticas generales para eventos de rendimiento, por ejemplo, instrucciones ejecutadas y ciclos de reloj consumidos, accesos a la caché, o TLB, etc. perf permite seleccionar indicadores específicos para reunir estadísticas, en vez de utilizar medidas predeterminados.

   Ejecute cada versión del programa, de manera separada (no en paralelo), utilizando el siguiente comando perf para conocer estadísticas de fallos de caché TLB y fallos de caché de datos.

Resguarde la información obtenida. Cada versión debe ser ejecutada de manera análoga al siguiente ejemplo:

perf stat -e dTLB-store-misses,cache-misses ./usamem_version1

¿Qué es la caché de datos TLB de un procesador, qué contiene, y para qué se utiliza?
¿Qué es la caché de datos de un procesador?

Analizar y responder en base a la información recolectada, tanto de los comandos perf, como de las respuestas sobre TLB y caché. Su respuesta puede ser sólo una hipótesis, o si cuenta con información probatoria, una afirmación. Preguntas:

1. ¿Qué versión tiene el peor rendimiento? ¿Encuentra una relación entre ese rendimiento con los fallos de caché y TLB reportados por perf?.
2. ¿Qué versión tiene el mejor rendimiento?. Analice en grupo de qué manera el programa de mejor rendimiento recorre la memoria reservada.
3. ¿Qué diferencia existe en la manera de recorrer la memoria entre el programa de mejor rendimiento y el de rendimiento medio?. Hipotetice o explique por qué razón el programa de mejor rendimiento supera al de rendimiento medio. En su análisis incluya una relación entre la manera de recorrer la memoria y los datos reportados por perf.
   Realice un diagrama o dibujo de la memoria y la forma en que se recorre.