CENTRO UNIVERSITARIO
DE TECNOLOGÍA Y ARTE DIGITAL

GRADO EN INGENIERíA SOFTWARE
SISTEMAS OPERATIVOS, 2019-2020

PRÁCTICA FINAL

Esta práctica vale un punto de la nota final. ES IMPRESCINDIBLE entregarla y obtener al menos un 5.

En esta práctica se pide desarrollar un simulador de gestión de la memoria virtual de un sistema ficticio de los años 80, TYRANNOSAUR, un ordenador con arquitectura de 16 bits.

TYRANNOSAUR usaba páginas de memoria de 4 KB y por tanto podía manejar 16 páginas de memoria virtual (haz la cuenta). El número de marcos era 4 y tenía una TLB de 4 líneas.

La tabla de memoria virtual de cada proceso es de un único nivel, y su estructura tan simple como un array de 16 enteros de 16 bits, que contienen el número de marco en el que se encuentra instanciada la página correspondiente o el número 255 si no está cargada. Para simplificar la práctica NO se van a actualizar las tablas de memoria virtual, solo se simulará la TLB.

Al arrancar el proceso kerneltyr hará un `fork()` y el hijo ejecutará mediante `exec()` o `system()` otro proceso llamado tlb que simula la TLB. Para que quede muy claro, hay que desarrollar en C sobre *Linux Mint* dos procesos: kerneltyr y tlb. El proceso kerneltyr guarda la pid del hijo en una variable, porque es la pid de tlb y servirá para comunicarse.

El proceso tlb dispone de un array de 4 elementos, del tipo T_TLB, cuya definición es

```
typedef struct {
    short int pagina;
    short int marco;
    short int valida;
    int tiempo;
} T_TLB;
```

El campo `valida` indica si la traducción es posible porque el marco está cargado. Si vale 1 la traducción es posible, si vale 0 el marco está libre con independencia de lo

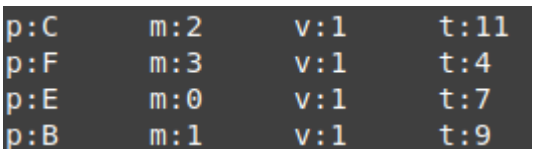
que contengan los campos `pagina` y `marco`. El campo tiempo es un contador que sirve para llevar la cuenta de cuando se cargó la página. El proceso tlb tendrá una variable `tiempoglobal` que inicializará a valor 0. También creará otra con el nombre `numfallos` inicializada a 0.

En el arranque, tlb inicializa a 0 todos los campos `valida` y `tiempo` y a 255 el campo `pagina`. El campo `marco` se debe rellenar empezando por el número 2 en la fila 0, como solo hay 4 marcos, el número es igual a (2+numero fila TLB) módulo 4.

Después de crear el proceso hijo con `fork()`, kerneltyr abrirá un pipe con nombre llamado `/tmp/FIFOTLB` que servirá de canal de comunicación entre los dos procesos. Puede encontrarse un ejemplo de uso en el fichero zip con el enunciado y los datos de prueba. Por su parte, el proceso tlb quedará a la espera de que exista el pipe nombrado y a la escucha.

A continuación, kerneltyr comienza la lectura del fichero `accesos_memoria.txt` que contiene una lista de direcciones de memoria en hexadecimal, una por línea. Se repetirá el siguiente protocolo

- kerneltyr escribe con formato hexadecimal en el named pipe la dirección virtual que ha leído del fichero y después hace un `sleep()` de 2 segundos.
- tlb la lee e incrementa el valor de `tiempoglobal`.
- tlb obtiene el número de página.
- Si no es `valida` o no está en la TLB incrementa el valor de `numfallos` y escribe una línea con el texto "`(valor del contador tiempoglobal), Fallo de TLB (número acumulado de fallos), VADDR (direccion virtual recibida HEX), pagina (número de página HEX), offset (offset)`". Se incrementa en 1 el contador `tiempoglobal`.
- Si la página no está cargada y hay un marco de memoria libre le asigna el primero que encuentre empezando por el comienzo de la tabla. Recuerda que en arranque todos los marcos están vacíos.
- Si todas las entradas de la TLB están llenas, la TLB expulsará a la página cuyo valor de tiempoglobal sea el menor. El proceso tlb escribe en pantalla "`(valor del contador tiempoglobal), Expulsada la página (número de la página HEX)`".
- El proceso traduce la dirección. Asigna al campo `tiempo` de la entrada correspondiente de la TLB el valor `tiempoglobal`.
- Por pantalla se escribe "`(valor del contador tiempoglobal), ACIERTO, VADDR (valor de la dirección virtual HEX) pagina (HEX) offset (HEX) marco (Decimal) => PHYSADDR (valor de la dirección física traducida HEX)`".
- El proceso tlb vuelca el contenido de la TLB por pantalla con el siguiente formato: `página (HEX) marco valida tiempo`.



Al finalizar la lectura de `accesos_memoria.txt` el proceso kerneltyr envía al proceso tlb la señal `SIGUSR2` y este al recibirla escribe un mensaje indicando que la ha recibido y sale del proceso invocando la función `exit(0)`.

Entrega de la práctica

Cada equipo subirá un .ZIP con: un fichero `autores.txt` con los nombres de los estudiantes, los fuentes `kerneltyr.c` y `tlb.c` y el volcado de la salida de la ejecución de kerneltyr en un fichero `llamado tlb_log.txt`. Pera ello ejecutar:

```
kerneltyr > tlb_log.txt
```

Fecha límite: 12 de enero de 2020. La corrección se hará sobre la máquina de los alumnos en enero de 2020. Los equipos pueden ser de 3 ó 4 alumnos. Solo hay que subir una versión por equipo.

Evaluación

- El programa kerneltyr arranca correctamente, crea el hijo y este lanza tlb (2 pt).
- El proceso kerneltyr crea el pipe nombrado (0,5 puntos).
- El proceso kerneltyr envía correctamente los datos a tlb por el pipe nombrado y este los lee (1 punto)
- El proceso tlb escribe bien por pantalla el contenido de la TLB (0,5 puntos)
- El proceso tlb asigna correctamente un marco libre a una página no cargada y escribe bien el resultado (1,5 puntos).
- El proceso tlb expulsa correctamente al marco más antiguo cuando no hay ninguno libre, lo asigna a una página y escribe bien el resultado en pantalla (1,5 puntos).
- El proceso tlb traduce bien las direcciones y escribe el resultado en pantalla (1 punto)
- El proceso tlb finaliza al recibir SIGUSR2 y tratar la señal debidamente (1 punto).
- Limpieza y documentación del código (1 punto).

FICHERO DE ENTRADA EJEMPLO

```
02E4
F045
ED12
ED16
B7F2
C908
4008
C9E2
```

EJECUCION DEL PROGRAMA

```
kerneltyr, mi pid es 2163 y la de tlb es 2164
p:FF   m:2    v:0    t:0
p:FF   m:3    v:0    t:0
p:FF   m:0    v:0    t:0
p:FF   m:1    v:0    t:0
1, Fallo de TLB 1, VADDR 02E4 pagina 0 offset 02E4
2, ACIERTO, VADDR 02E4 pagina 0 offset 02E4 marco 2 => PHYSADDR 22E4
p:0    m:2    v:1    t:2
```

```
p:FF   m:3    v:0    t:0
p:FF   m:0    v:0    t:0
p:FF   m:1    v:0    t:0
3, Fallo de TLB 2, VADDR F045 pagina F offset 0045
4, ACIERTO, VADDR F045 pagina F offset 0045 marco 3 => PHYSADDR 3045
p:0    m:2    v:1    t:2
p:F    m:3    v:1    t:4
p:FF   m:0    v:0    t:0
p:FF   m:1    v:0    t:0
5, Fallo de TLB 3, VADDR ED12 pagina E offset 0D12
6, ACIERTO, VADDR ED12 pagina E offset 0D12 marco 0 => PHYSADDR 0D12
p:0    m:2    v:1    t:2
p:F    m:3    v:1    t:4
p:E    m:0    v:1    t:6
p:FF   m:1    v:0    t:0
7, ACIERTO, VADDR ED16 pagina E offset 0D16 marco 0 => PHYSADDR 0D16
p:0    m:2    v:1    t:2
p:F    m:3    v:1    t:4
p:E    m:0    v:1    t:7
p:FF   m:1    v:0    t:0
8, Fallo de TLB 4, VADDR B7F2 pagina B offset 07F2
9, ACIERTO, VADDR B7F2 pagina B offset 07F2 marco 1 => PHYSADDR 17F2
p:0    m:2    v:1    t:2
p:F    m:3    v:1    t:4
p:E    m:0    v:1    t:7
p:B    m:1    v:1    t:9
10, Fallo de TLB 5, VADDR C908 pagina C offset 0908
Expulsada pagina 0000
11, ACIERTO, VADDR C908 pagina C offset 0908 marco 2 => PHYSADDR 2908
p:C    m:2    v:1    t:11
p:F    m:3    v:1    t:4
p:E    m:0    v:1    t:7
p:B    m:1    v:1    t:9
12, Fallo de TLB 6, VADDR 4008 pagina 4 offset 0008
Expulsada pagina 000F
13, ACIERTO, VADDR 4008 pagina 4 offset 0008 marco 3 => PHYSADDR 3008
p:C    m:2    v:1    t:11
p:4    m:3    v:1    t:13
p:E    m:0    v:1    t:7
p:B    m:1    v:1    t:9
14, ACIERTO, VADDR C9E2 pagina C offset 09E2 marco 2 => PHYSADDR 29E2
p:C    m:2    v:1    t:14
p:4    m:3    v:1    t:13
p:E    m:0    v:1    t:7
p:B    m:1    v:1    t:9
Recibida la señal SIGUSR2, fin del proceso tlb
```