TP3: Multitarea con desalojo

static_assert

La macro definida en jos se define a base de un switch, en el cual se evalúa la condición del assert. Si esta es 0, se continúa, en caso contrario, se genera un error en tiempo de compilación. Esto se debe a que no se definen todos los casos en el switch.

env_return

Una vez terminada la función umain() de un proceso el kernel retoma la ejecución en la función libmain(), donde llamará a exit(). Esta a su vez llama a sys_env_destroy(), dentro de la cual se ejecuta envid2env para convertir el id de proceso al puntero al mismo. Finalmente se llama a env_destroy, que a su vez ejecuta env_free para desalocar la memoria del proceso y destruir el proceso. La diferencia con el anterior TP, es que en este caso, una vez destruido el proceso, se verifica si el proceso es el mismo al que se estaba ejecutando y si es así, se llama a sched_yield() para que el kernel pueda ejecutar algún otro proceso en condiciones de hacerlo (estado: ENV_RUNNABLE).

contador_env

El código que asegura que el buffer VGA fisico no sera nunca añadido a la lista de páginas libres se encuentra en page_init() en el caso del ***IO hole*** cuyo rango es [IOPHYSMEM, EXTPHYSMEM) en el cual se encuentra la dirección **0xb8000** del buffer VGA.

sys_yield

```
SMP: CPU 0 found 1 CPU(s)
enabled interrupts: 1 2
[00000000] new env 00001000
[00000000] new env 00001001
[00000000] new env 00001002
Hello, I am environment 00001000.
Hello, I am environment 00001001.
Hello, I am environment 00001002.
Back in environment 00001000, iteration 0.
Back in environment 00001001, iteration 0.
Back in environment 00001002, iteration 0.
Back in environment 00001000, iteration 1.
```

```
Back in environment 00001001, iteration 1.
Back in environment 00001002, iteration 1.
Back in environment 00001000, iteration 2.
Back in environment 00001001, iteration 2.
Back in environment 00001002, iteration 2.
Back in environment 00001000, iteration 3.
Back in environment 00001002, iteration 3.
Back in environment 00001000, iteration 4.
All done in environment 00001000.
[00001000] exiting gracefully
[00001000] free env 00001000
Back in environment 00001001, iteration 3.
Back in environment 00001002, iteration 4.
All done in environment 00001002.
[00001002] exiting gracefully
[00001002] free env 00001002
Back in environment 00001001, iteration 4.
All done in environment 00001001.
[00001001] exiting gracefully
[00001001] free env 00001001
No runnable environments in the system!
```

Las primeras 3 líneas indican que se crearon 3 nuevos procesos (según i386_init). Luego se imprime tres veces "Hello, I am environment ...", según el número de proceso que esté ejecutándose, pero como en umain yield.c, despúes de este print hay un sys_yield (que cede la CPU a otro proceso en estado ENV_RUNNABLE), se imprime una línea de cada proceso en ejecución alternada (según modelo Round Robin). Lo mismo sucede en cada iteración-impresión del programa hasta que alguno termine con sus respectivas iteraciones (5), donde el proceso es desalocado y liberado.

envid2env

sys_env_destroy(0): llama a envid2env(0), el cual devuelve un puntero al curenv (proceso actual), y luego sys_env_destroy llama a env_destroy, el cual libera el proceso. Finalmente, setea curenv a NULL y ejecuta sched_yield().

sys_env_destroy(-1): llama a envid2env(-1), el cual busca el proceso en envs, según el índice obtenido por ENVX (el cual da un índice inválido por ser negativo). Por esto, al comparar el env_id del proceso obtenido con el pasado por parámetro, la función retornará -E_BAD_ENV, el cual es devuelto por sys_env_destroy.

kill(0, 9): 0 es el pid (process id) y 9 es sig (SIGKILL), entonces esta señal es enviada a todos los procesos en el grupo del proceso que hizo la llamada.

kill(-1, 9): la señal es enviada a todos los procesos a los que tenga permisos para enviar señales el proceso que hizo la llamada, excepto a 1 (init).

dumbfork

1. Si antes de llamar a dumbfork se llama sys_page_alloc reservando una página para el proceso padre, el proceso hijo también la poseerá ya que en dumbfork se hace una llamada a duppage que copia exactamente el address space del padre en el hijo.
2. No, no se preserva ya que duppage() llama a sys_page_alloc y sys_page_map con permisos de escritura.
3. duppage() hace tres syscalls: primero aloca una página en addr para el proceso cuyo id es envid con permisos de escritura. Luego mapea addr al curenv a la dirección UTEMP. Copia la página en UTEMP a addr. Finalmente deshace el mapeo que realizó anteriormente, desalocando la página que alocó con sys_page_alloc.
4. Lo que cambiaría con un parámetro que indica si la página debe quedar para solo lectura, es un nuevo if para modificar los permisos que se le pasan a las syscalls.
5. Debido a que el stack crece hacia las direcciones de memoria baja, se utiliza ROUNDDOWN en lugar de ROUNDUP para copiarlo (ROUNDDOWN redondea hacia las direcciones bajas, restándole el valor aplicado el módulo del valor del tamaño de la página). Addr es el puntero dado por la variable end, la cual es generada por el linker y apunta al final del segmento de bss del kernel (la primera dirección virtual a la cual el kernel no le asignó código o variables globales).