SO2 (14/11/2025)

Primer control de laboratorio

Crea un fichero que se llame "respuestas.txt" donde escribirás las respuestas y el código para los apartados de los ejercicios del control. Indica para cada respuesta, el **número de ejercicio y el número de apartado** (por ejemplo, 1.a, 1.b, ...).

Importante: para cada uno de los ejercicios tienes que partir del código suministrado de Zeos.

1. (6 puntos) Mailboxes?

Queremos añadir una nueva funcionalidad a ZeOS que permita enviar información entre 2 procesos.

Tienes que añadir 2 nuevas llamadas a sistema:

```
int writeTo(int pid, char* src, int size);
int readFrom(int* pid, char *dst);
```

Cada proceso tendrá una página física asociada para **recibir información**, así como un contador del número de bytes que se han enviado, un puntero al escritor activo y una lista de procesos pendientes para escribir.

La función *writeTo* permite enviar *size* bytes desde *src* al proceso lector *pid*. Si el proceso lector estuviera bloqueado y no tiene ningún escritor activo, debemos desbloquearlo. A continuación, el proceso actual debe quedarse bloqueado en la lista de procesos pendientes del proceso lector a que el *readFrom* lo desbloquee. Al desbloquearse, el proceso debe mapear la página física del proceso lector en la dirección lógica 0x3FF000, copiar la región de memoria desde *src*, desmapear la página física, desbloquear al lector y devolver el número de bytes escritos. Para simplificar, sólo permitimos escrituras de hasta 4096 bytes.

La función *readFrom* permite leer la información enviada por otros procesos. Para ello, si no hay nadie bloqueado para escribir, el proceso actual debe bloquearse. Caso que haya alguien, debe marcarlo como el actual escritor activo, desbloquearlo y bloquearse a que este escritor escriba los datos. Al desbloquearse, debe mapear la página física en la dirección 0x3FD000, copiar tantos bytes como se hayan escrito a partir de la dirección *dst*, desmapear la página, marcar que ya no hay escritor activo, copiar el pid del escritor en el parámetro *pid* y devolver el número de bytes leídos.

Estas llamadas deben implementarse usando los números de servicio 40 y 41 respectivamente. Puedes ignorar la gestión de errores de estas llamadas.

En particular, tienes que:

a) (0.5 puntos) Añade los wrappers necesarios.
b) (0.5 puntos) Define una estructura nueva con la información necesaria por proceso.
c) (0.5 puntos) Indica los cambios necesarios en las estructuras del sistema al añadir estas dos nuevas llamadas.
d) (0.5 puntos) Implementa la función *initializeData*(PCB) per inicializar la estructura de un proceso dado.
e) (0.5 puntos) Implementa la función *mapPhysicalPageFrom*(PCB, address) para mapear la página física del proceso en la dirección pasada.
f) (0.5 puntos) Implementa una función *unmapPage*(PCB, address) que permita desmapear las páginas usadas en el writeTo y el readFrom respectivamente.
g) (1 punto) Implementa *sys_writeTo.*
h) (1 punto) Implementa *sys_readFrom.*

i) (1 punto) Indica los cambios necesarios en las llamadas a sistema preexistentes.

2. (4 puntos) Switch baby, switch...

Estamos hartos de tener que dividir la función de task_switch en 2, por lo que queremos modificar la implementación actual para eliminar la función en ensamblador que guarda los registros EBX, ESI y EDI de forma artificial. Por lo tanto, renombraremos la función *inner_task_switch* para que sea la función *task_switch*. Y cambiaremos la función *switch_stack* para que tenga esta interfaz:

```
void switch_stack(int* save_sp, int* new_sp);
```

Esta función debe: 1) guardar los registros EBX, ESI y EDI en la pila, 2) guardar la **posición de la pila en la que ha quedado el último registro** en el primer parámetro, 3) cambiar a la pila del segundo parámetro, y 4) restaurar los registros. Al salir de esta función debe volver a la instrucción posterior a la que (en que el otro proceso) hubiera invocado a *switch_stack*.

a) (1 punto) Muestra el contenido de la pila de sistema justo al empezar la rutina *switch_stack* implementada originalmente, junto con los comandos que has usado para obtenerlo.
b) (0.5 puntos) Modifica la compilación para que el el fichero con la rutina *inner_task_switch* se compile sin optimización.
c) (1 punto) Implementa los cambios propuestos.
d) (0.5 puntos) Indica el contenido de la pila que encontraremos justo antes del punto 3.
e) (0.5 puntos) Indica los cambios necesarios en la inicialización del sistema.
f) (0.5 puntos) Indica los cambios necesarios en las llamadas a sistema ya existentes.

3. Entrega

Sube al Racó los ficheros “respuestas.txt” junto con el código que hayas creado en cada ejercicio. Para entregar el código utiliza:

```
> tar zcfv examen.tar.gz respuestas.txt ejercicio1 ejercicio2
```

SOLUCION

1.a

WRAPPERS normales con EAX=40 y 41 respectivamente

1.b

```
struct pipe {
      int frame;            //physical frame to store data
      int numbytes;         //number of bytes in page
      struct task_struct* readingFrom; //active writer
      struct list_head writers;//list of pending writers
}
```

1.c

Modificar sys_call_table añadiendo las entradas necesarias

En el PCB>

```
      ...
      struct pipe pipeinfo;
      ...
```

1.d

```
      void initializeData(struct task_struct*p) {
            p->pipeinfo.frame = alloc_frame();
            p->pipeinfo,numbytes=0;
            p->readingFrom = NULL;
            INIT_LIST_HEAD(&p->pipeinfo.writers);

}
```

1.e

```
int mapPhysicalPageFrom(struct task_struct* p, int address) {
   int frame = p->pipeinfo.frame;
   page_table_entry *myPT = get_PT(current());

   set_ss_page(myPT, address>>12, frame);
}
```

1.f

```
int unmapPage(struct task_struct*p, int address) {
   page_table_entry *myPT = get_PT(current());
   del_ss_page(myPT, address>>12);
   set_cr3(get_DIR(p));
}
```

1.g

```
int sys_writeTo(int pid, char* src, int size) {
      struct task_struct*p = findTaskByPid(pid);
     if (p->state = BLOCKED
            && readingFrom=NULL) //Unblock READER
            update_process_state(p, &readyqueue);
      //BLOCK
       update_process_state(current(),&p->pipeinfo.writers);
      sched_next();

      mapPhysicalPageFrom(p, 0x3FF000);
      copy_data(src, 0x3FF000, size);
      p->pipeinfo = size;
       unmapPage(current(), 0x3FF000);
```

```
      update_process_state(p, &readyqueue); //Unblock READER

      return size;
}
1.h
int sys_recvFrom(int* pid, char* dst){
      if (list_empty(&current()->pipeinfo.writers)) {
                  update_process_state(current(), &blockedq)
                  sched_next()//BLOCK Generic
      }
      t = list_first(&current()->pipeinfo.writers)
      struct task_struct* p = list_head_2_task_struct(t),
      readingFrom = p;
      update_process_state(p,  &ready)//Unblock WRITER
      update_process_state(current(), &blockedq)
      sched_next()// Block Generic

       int size = current()->pipeinfo.numbytes;
      mapPhysicalPageFrom(current(), 0x3FD000);
      copy_data(0x3FD000, dst, size);
      unmapPage(current(), 0x3FD000);

      readingFrom = NULL; // Read COMPLETED!

      *pid = t->pid; //copy_to_user(pid, &t->pid, sizeof(int));
      return size;
}

1.i
init_task1
      initializeData(task1);
fork
      initalizeData(newchild);
exit
      free_frame(current()->pipeinfo.frame);
      //unblock writers
      for_each_safe(p, tmp, current()->pipeinfo.writers) {
          list_del(p);
          update_process_state(list_head_2_task_struct(p),
&readyqueue);
      }

2.a
2n par, 1r par, @ret, EBP, EBX, ESI, EDI
gdb system; break switch_stack; pila
2.b
sched.o: sched.c
      gcc ...
2.c

ENTRY(switch_stack)
    pushl %ebp
    movl %esp, %ebp
    pushl %ebx
    pushl %esi
    pushl %edi
```

```
    movl 8(%ebp), %eax
    movl %esp, (%eax)
    movl 12(%ebp), %eax
    movl (%eax), %esp
    popl %edi
    popl %esi
    popl %ebx
    popl %ebp
    ret
```

2.d
2n par, 1r par, @ret, EBP, EBX, ESI, EDI

2.c
Hay que modificar el init_idle para inicializar la pila con variables dummy para los 3 registros y guardar el puntero al 3r registro en el PCB.

2.d
Hay que inicializar en el fork el CONTEXTO del hijo, así como el punto de entrada en el pcb

```
uchild->task.register_esp=register_ebp - 4 * sizeof(DWord);
  int* ctxt = (int*)uchild->task.register_esp;
  ctxt[0] = 0; //EBX
  ctxt[1] = 0; //ESI
  ctxt[2] = 0; //EDI
  ctxt[3] = register_ebp; //EBP
  ctxt[4] = (int)&ret_from_fork;
```