Práctica 1

MiniKernel

Índice

- Descripción de entorno.
- Hardware virtual
- Sistema operativo: kernel
- Programas de usuario
- Funcionalidad

Entorno de prácticas

- Hardware virtual (`minikernel/HAL`)
- Programa cargador (`boot/boot`)
- Sistema operativo: kernel (`minikernel/kernel`)
- Programas de usuario (`usuario`)
- Compilación: `make`
  - Errores en programas de usuario que usan servicios no implementados
- Ejecución del sistema operativo:

```
boot/boot minikernel/kernel
```

  - Problemas en algunos terminales:

```
reset
```

  - O mejor:

```
boot/boot minikernel/kernel > salida
```

Hardware virtual

- Hardware virtual sobre el que se desarrolla el mini S.O.
- Características del "procesador virtual":
  - 2 modos de ejecución: usuario y sistema
  - 2 dispositivos de E/S guíados por inter. : Reloj y Terminal
  - 2 tipos de excepciones: aritméticas y de acceso a memoria
  - 6 vectores de interrupción:
    - 2 excep., int. reloj y de terminal, llamada a sist. e int. software
  - 3 niveles de interrupción. De mayor a menor prioridad:
    - N3 (int. reloj)
    - N2 (int. teclado)
    - N1 (llamada o int. software)
    - Inicialmente, N3 y modo sistema.
  - 6 registros generales (sólo usados directamente en llamadas)

Hardware virtual

- Ofrece una capa de servicios al S.O.:
  - Módulo Hardware Abstraction Layer (`minikernel/include/hal.h`)
- Proporciona operaciones para:
  - Manejo de controladores de dispositivos
    - P.ej. `iniciar_cont_reloj(int ticks_por_seg);`
  - Gestión de interrupciones (instalar un manejador)
    - P.ej. `fijar_nivel_int(int nivel);`
  - Gestión de contextos de procesos
    - P.ej. `cambio_contexto(contexto_t *contexto_a_salvar, contexto_t *contexto_a_restaurar);`
  - Gestión de memoria
    - P.ej. `crear_pila(int tam);`
  - Otras operaciones
    - P.ej. `printk(const char *, ...);`

Sistema operativo: kernel

- Se proporciona una versión inicial que hay que modificar (`minikernel/kernel`)
  - Iniciación: **completa** (aunque se pueden añadir cosas)
  - Tratamiento de int. externas y software: vacío
  - Tratamiento de excepciones: **completo**
  - Infraestructura de llamadas al sistema: **completa**
  - Llamadas al sistema: hay tres implementadas
    - crear_proceso, terminar_proceso, escribir
  - Estructuras de datos ya definidas, pero modificables:
    - BCP, tabla de procesos, tipo cola de procesos, cola de listos

Iniciación del kernel

- Se proporciona una versión inicial que hay que modificar
- Extracto correspondiente a la iniciación del S.O.:

```
instal_man_int(EXC_ARITM, exc_arit);
… /*todos los manejadores */

iniciar_cont_int();          /* inicia cont. interr. */
iniciar_cont_reloj(TICK);/* fija frecuencia del reloj */
iniciar_cont_teclado();     /* inici cont. teclado */
iniciar_tabla_proc();       /* inicia BCPs */
crear_tarea((void *)"init"); /* crea proceso inicial */
p_proc_actual=planificador(); /* activa proc. inicial */
cambio_contexto(NULL, &(p_proc_actual->contexto_regs));
```

Tratamiento de llamadas

minikernel/include/llamsis.h:

```
#define NSERVICIOS 3 /* Numero de llamadas
disponibles */
#define CREAR_PROCESO 0
#define TERMINAR_PROCESO 1
#define ESCRIBIR 2
```

minikernel/include/kernel.h:

```
servicio tabla_servicios[NSERVICIOS]={
      {sis_crear_proceso},
      {sis_terminar_proceso},
      {sis_escribir}};
```

Tratamiento de llamadas

- Se usan registros

```
static void tratar_llamsis(){
    int nserv, res;

    nserv=leer_registro(0);
    if (nserv<NSERVICIOS)

    res=(tabla_servicios[nserv].fservicio)();
    else
          res=-1;    /* servicio no existente */
    escribir_registro(0,res);
    return;
}
```

Invocación de llamadas

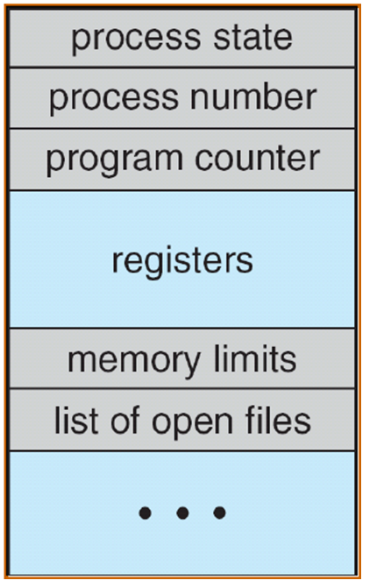
- Desde el modo usuario (`usuario/lib/serv.c`)

```
int llamsis(int llamada, int nargs,...//args) {
 int i;
 escribir_registro(0, llamada);
 for (i=1; nargs; nargs--, i++)
       escribir_registro(i, args[i]);
 trap();
 return leer_registro(0);
}

int crear_proceso(char *prog){
 return llamsis(CREAR_PROCESO, 1, (long)prog);
}
```

Definición de BCP

- En `minikernel/include/kernel.h`

```
typedef struct BCP_t {
    int id;         /* ident. del proceso */
    int estado; /*TERM.|LISTO|EJEC.|BLOQ.*/
    contexto_t contexto_regs;/*regs. */
    void * pila;/* dir. ini. de la pila */
    BCPptr siguiente;  /* punt. a otro BCP */
    void *info_mem;/* descriptor del mapa de
                       memoria */
} BCP;

BCP * p_proc_actual=NULL;
BCP tabla_procs[MAX_PROC];
```



Colas de procesos

- Los procesos se organizan en colas
  - Cola de listos (incluye proceso actual)

```
lista_BCPs lista_listos= {NULL, NULL};
```

- Se pueden crear colas
  - Puntero siguiente en BCP para colas con enlace simple
  - BCP debe estar en una sola cola en cada instante
  - Tipo lista de BCPs que puede usarse para cualquier cola:

```
typedef struct{
        BCP *primero;
        BCP *ultimo;
} lista_BCPs;
```

Cambio de contexto

… Sentencias previas…

```
p_proc_actual->estado=LISTO o BLOQUEADO;
p_proc_anterior = p_proc_actual;
nivel_int = fijar_nivel_int(3);
eliminar_primero(&lista_listos);
insertar_ultimo(&lista_destino, p_proc_actual);
p_proc_actual = planificador();
fijar_nivel_int(nivel_int);
cambio_contexto(&(p_proc_anterior->contexto_regs),
      &(p_proc_actual->contexto_regs));
```

… Sentencias posteriores…

Creación de un proceso


```
proc=buscar_BCP_libre();

p_proc=&(tabla_procs[proc]); /* A rellenar el BCP ... */

/* crea la imagen de memoria leyendo ejecutable */
imagen=crear_imagen(prog, &pc_inicial);
if (imagen)
{
        p_proc->info_mem=imagen;
        p_proc->pila=crear_pila(TAM_PILA);
        fijar_contexto_ini(p_proc->info_mem, p_proc->pila, TAM_PILA,
                pc_inicial, &(p_proc->contexto_regs));
        p_proc->id=proc;
        p_proc->estado=LISTO;

        /* lo inserta al final de cola de listos */
        insertar_ultimo(&lista_listos, p_proc);
        return 0;
}
else
        return -1; /* fallo al crear imagen */
```

Terminación de un proceso


```
/* liberar mapa */
liberar_imagen(p_proc_actual->info_mem);

/* proc. fuera de listos */
p_proc_actual->estado=TERMINADO;
eliminar_primero(&lista_listos);

/* Realizar cambio de contexto */
p_proc_anterior=p_proc_actual;
p_proc_actual=planificador();
liberar_pila(p_proc_anterior->pila);
cambio_contexto(NULL,
     &(p_proc_actual->contexto_regs));
return; /* no debería llegar aqui */
```

Programas de usuario

- Llamadas disponibles como funciones en biblioteca `usuario/lib/serv`
- Hay programas para probar todas las funcionalidades pedidas
- Programa init: Cargado en el arranque del S.O.
  - Incluye la activación de todos los programas de prueba
  - Para realizar una prueba basta con descomentarla

Funcionalidad requerida

1. Inclusión de una llamada simple (`obtener_id_pr`)
2. Llamada que bloquea al proceso un plazo (`dormir`)
3. Servicio de mutex
   - `crear_mutex`
   - `abrir_mutex`
   - `cerrar_mutex`
   - `lock`
   - `unlock`
4. Planificación Round Robin
5. Manejo del teclado (`leer_caracter`)