Diseño de sistemas operativos

Práctica 1
Minikernel

Fernando Pérez Costoya

Índice

- Descripción del entorno de prácticas
- Módulo HAL
- Sistema operativo
- Programas de usuario
- Funcionalidad pedida

Entorno de prácticas

- Hardware virtual (`minikernel/HAL`)
- Programa cargador (`boot/boot`)
- Sistema operativo (módulo `minikernel/kernel`)
- Programas de usuario (directorio `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 un (mini)S.O.
- Características del “procesador virtual”:
  - 2 modos de ejecución: usuario y sistema
    - No usa pila de 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)

Módulo HAL

- Hardware virtual ofrece una capa de servicios al S.O.:
  - Módulo HAL*: Hardware Abstraction Layer*
- Proporciona operaciones para:
  - Manejo de controladores de dispositivos
    - P.ej. `iniciar_cont_reloj`
  - Gestión de interrupciones
    - P.ej. `instal_man_int`
  - Gestión de contextos de procesos
    - P.ej. `cambio_contexto`
  - Gestión de memoria
    - P.ej. `crear_imagen`

Módulo kernel: el S.O.

- Se proporciona una versión inicial que hay que modificar
- Algunos aspectos destacables:
  - 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:
    - BCP, tabla de procesos, tipo cola de procesos, cola de listos

Iniciación del S.O.

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

```
int main(){
        instal_man_int(EXC_ARITM, exc_arit);
        ..... // Instalar todos los manejadores
        iniciar_cont_int();
        iniciar_cont_reloj(TICK);
        iniciar_cont_teclado();
        iniciar_tabla_proc();
        crear_tarea((void *)"init");
        p_proc_actual=planificador();
        cambio_contexto(NULL, &(p_proc_actual->
                                        contexto_regs));}
```


Tratamiento de excepciones

- Ejemplo: tratamiento de excepciones aritméticas

```
static void exc_arit(){
    if (!viene_de_modo_usuario())
            panico("excepcion aritmetica cuando estaba
                    dentro del kernel");
    liberar_proceso();
    return; /* no debería llegar aqui */
}
```

Tratamiento de llamadas (1/2)

```
#define NSERVICIOS 3
#define CREAR_PROCESO 0
#define TERMINAR_PROCESO 1
#define ESCRIBIR 2
servicio tabla_servicios[NSERVICIOS]={
        {sis_crear_proceso},
        {sis_terminar_proceso},
        {sis_escribir}};
```


Tratamiento de llamadas (2/2)

- Usa registros para todo (como Linux)

```
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

```
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

- S.O. es sólo otro programa: Usa definiciones convencionales

```
typedef struct BCP_t {
    int id; /* pid */
    int estado;
    contexto_t contexto_regs; /* copia de regs. */
    void * pila; /* dir. inicial de la pila */
    BCPptr siguiente; /* puntero a otro BCP */
    void *info_mem; /* descriptor mapa memoria */
} BCP;
BCP * p_proc_actual;
BCP tabla_procs[MAX_PROC];
```

Colas de procesos

- 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;
```

- Cola de listos (incluye proceso actual)

```
lista_BCPs lista_listos;
```


Cambio de contexto

```
..... Sentencias previas .....
p_proc_actual->estado=LISTO o BLOQUEADO;
p_anterior=p_proc_actual;
nivel=fijar_nivel_int(NIVEL_3);
eliminar_primero(&lista_listos);
insertar_ultimo(&lista_destino, p_anterior);
fijar_nivel_int(nivel);
p_proc_actual=planificador();
cambio_contexto(&(p_anterior->contexto_regs),
        &(p_proc_actual->contexto_regs));
..... Sentencias posteriores .....
```


Creación de un proceso

```
proc=buscar_BCP_libre();
p_proc=&(tabla_procs[proc]);
/* crea la imagen de memoria leyendo ejecutable */
imagen=crear_imagen(prog, &pc_inicial);
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);
```


Terminación de un proceso

```
liberar_imagen(p_proc_actual->info_mem);
p_proc_actual->estado=TERMINADO;
eliminar_primero(&lista_listos);
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 `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 pedida

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