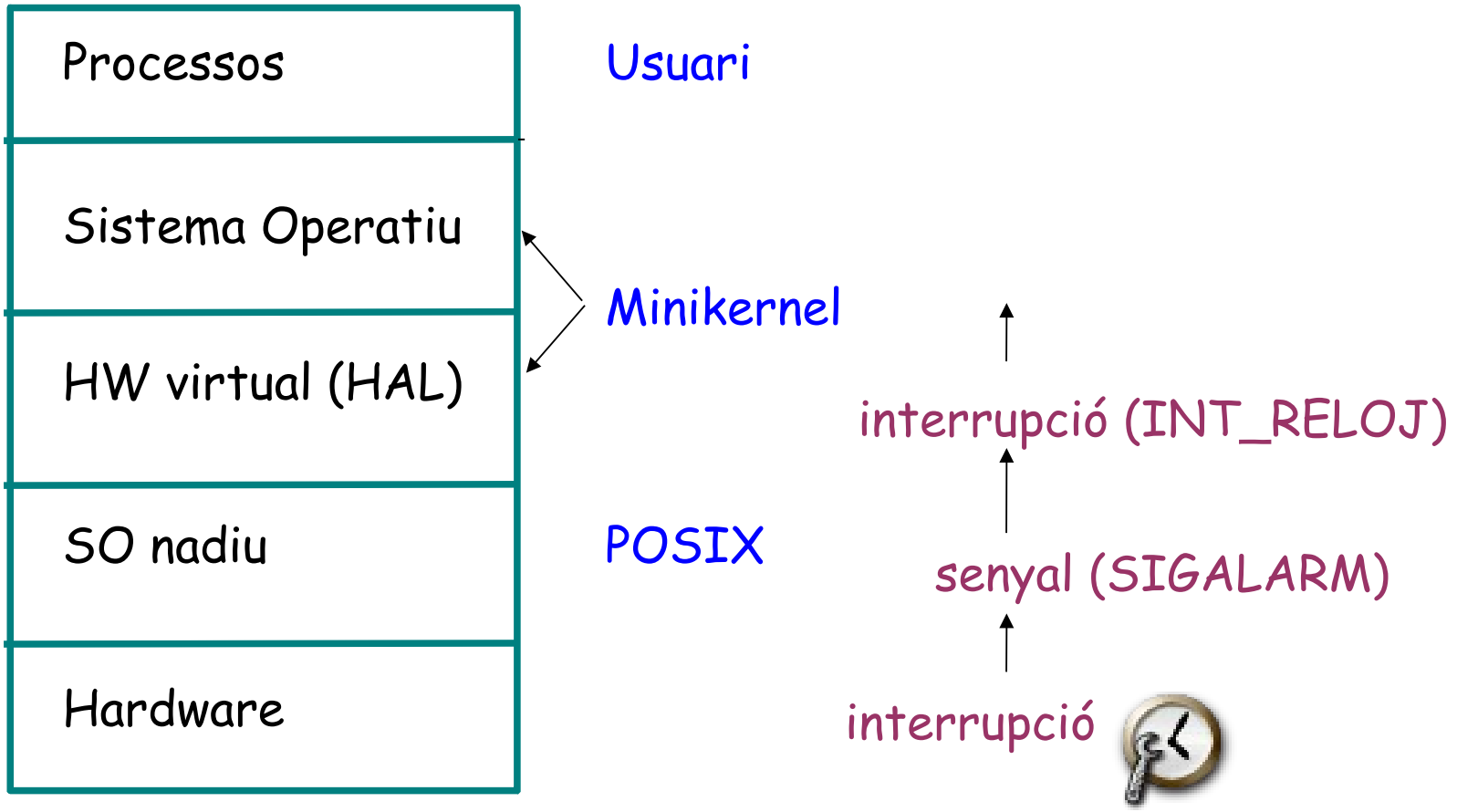
Arquitectura del minikernel



HAL (Hardware Abstraction Layer)

Modes:

privilegiat

usuari

↓

Registre d'estat

Nivells de prioritat:

Nivell 3: rellotge

Nivell 2: terminal

Nivell 1: crida al sistema o int. software

Nivell 0: mode usuari

Interrupcions	tabla_vectores
Rellotge	INT_RELOJ
Terminal	INT_TERMINAL
Excep. aritm	EXC_ARITM
Excep. memo	EXC_MEM
Crida sistema	LLAM_SIS
Software	INT_SW

Estructura de directoris de l'entorn

- Makefile
- boot -> boot
- minikernel -> Makefile
  - kernel.c, HAL.o
  - kernel
  - include -> HAL.h, const.h, llamsis.h, kernel.h
- usuari -> Makefile
  - init.c, *.c
  - include -> servicios.h
  - lib -> Makefile
    - serv.c
    - misc.o
    - libserv.a (serv.o, misc.o)

kernel.c

```
int main(){
    /* s'arriba amb les interrupcions prohibides */
    iniciar_tabla_proc();
    instal_man_int(EXC_ARITM, exc_arit);
    instal_man_int(EXC_MEM, exc_mem);
    instal_man_int(INT_RELOJ, int_reloj);
    instal_man_int(INT_TERMINAL, int_terminal);
    instal_man_int(LLAM_SIS, tratar_llamsis);
    instal_man_int(INT_SW, int_sw);
```

→ Instalació de les funcions de tractament de les interrupcions

```
    iniciar_cont_int();
    iniciar_cont_reloj(TICK);
    iniciar_cont_teclado();
```

Inicialització dels controladors de dispositius

```
    if (crear_tarea((void *)"init")<0)
        panico("no encontrado el proceso inicial");
```

CREACIÓ PROCÉS INICIAL

```
    p_proc_actual=planificador();
```

SCHEDULER DE PROCESSOS

```
    cambio_contexto(NULL, &(p_proc_actual->contexto_regs));
    panico("S.O. reactivado inesperadamente"); return 0;
}
```

EXECUCIÓ PROCÉS ACTUAL

kernel.c

- Exemple de funció de tractament d'interrupció del terminal:

```
static void int_terminal(){

        printk("-> TRATANDO INT. DE TERMINAL %c\n",
                                        leer_puerto(DIR_TERMINAL));

        return;
}
```

- Funció de tractament d'interrupció de crides al sistema:
  - Les possibles crides al sistema estan guardades al vector tabla_servicios.
  - Els paràmetres es passen via registres (en el 0 es posa quina crida de la tabla_servicios es vol executar) i el retorn es torna en el registre 0.

```
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;
}
```

kernel.h

```
typedef struct{
        int (*fservicio)();
} servicio;

/*
 * Prototipos de las rutinas que realizan cada llamada al sistema
 */
int sis_crear_proceso();
int sis_terminar_proceso();
int sis_escribir();

/*
 * Variable global que contiene las rutinas que realizan cada llamada
 */
servicio tabla_servicios[NSERVICIOS]={  {sis_crear_proceso},
                                        {sis_terminar_proceso},
                                        {sis_escribir}};
```

llamsis.h

```
#ifndef _LLAMSIS_H
#define _LLAMSIS_H

/* Numero de llamadas disponibles */
#define NSERVICIOS 3

/* Identificador de la crida a sistema  */
#define CREAR_PROCESO 0
#define TERMINAR_PROCESO 1
#define ESCRIBIR 2


#endif /* _LLAMSIS_H */
```

kernel.h

Informació dels procesos: BCP

```
typedef struct BCP_t *BCPptr;

typedef struct BCP_t {
        int id;                         identificador del procés
        int estado;                     TERMINADO|LISTO|EJECUCION|BLOQUEADO
        contexto_t contexto_regs;       còpia de regs. de UCP
        void * pila;                    dir. inicial de la pila
        BCPptr siguiente;               apuntador a altre BCP
        void *info_mem;                 descriptor del mapa de memòria
}BCP;
```

Informació dels procesos: llista de BCP's

Aquest tipus s'utilitza en diferents llistes: processos a punt (listos), processos bloquejats per semàfors, etc.

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

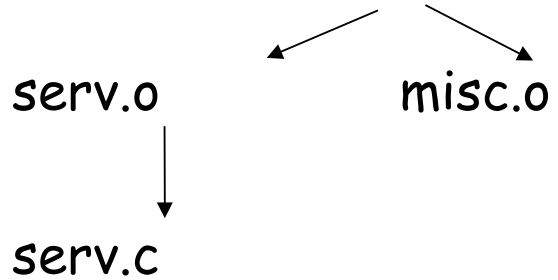
Variables globals del kernel:

```
BCP * p_proc_actual=NULL;

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

Procesos d'usuari

- Procés bàsic: init
- Processos que es llencen: simplon, excep_arit, excep_memo
- Llibreria utilitzada: libserv.a



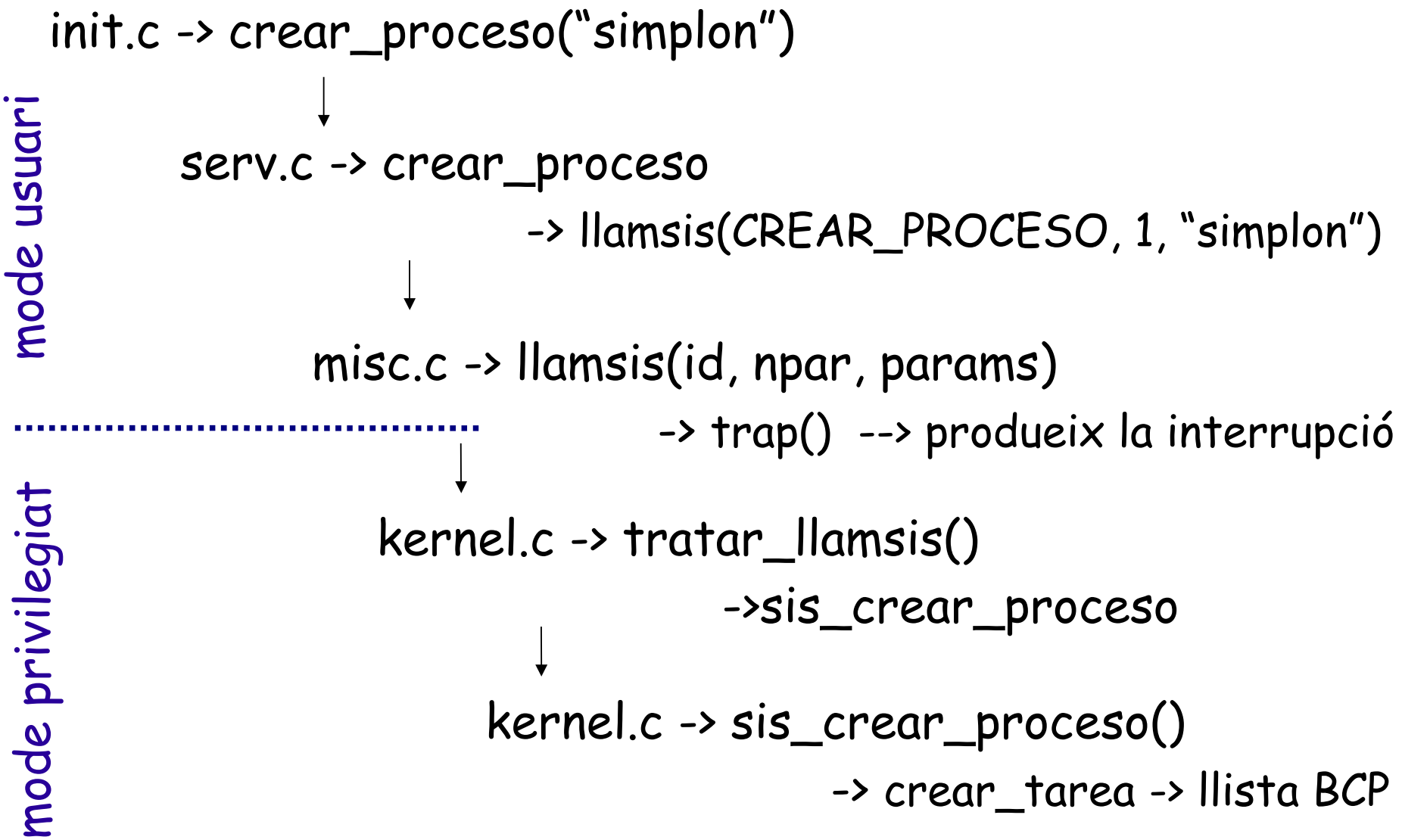
conté la capa API
de crides al sistema

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

```
/*
 * Tractament de crida al sistema crear_proceso. Crida a la
 * funció auxiliar crear_tarea
 */
int sis_crear_proceso() {
        char *prog;
        int res;

        printk("-> PROC %d: CREAR PROCESO\n", p_proc_actual->id);
        prog=(char *)leer_registro(1);
        res=crear_tarea(prog);
        return res;
}
```

Execució d'una crida al sistema



Exercicis proposats

- Baixar el tar del minikernel i explorar tot el sistema
- Arrencar el kernel: boot/boot minikernel/kernel >a.a
- Per què surt dues vegades l'execució del simplon?
- Crear un altre procés d'usuari des del init.c amb un dels programes C desenvolupats en les sessions anteriors que només tingui sortida (fibo.c, per exemple).
- Heu de subtituir #include <stdio.h> per #include "servicios.h" per a que s'utilitzi el printf del minikernel.
- Heu d'editar el Makefile per a que el recompili.

Introducció d'una nova crida al sistema

- kernel:
  - kernel.c :
    - implementar la nova funció que implementa la nova crida al sistema (sis_nova)
  - include: kernel.h :
    - la nova capçalera de la crida (sis_nova)
    - incloure-la a la tabla_servicios
  - include: llamsis.h :
    - incrementar en 1 la constant NSERVICIOS
    - definir l'identificador de la nova crida com una constant (NOVA)
- Usuari:
  - lib: serv.c : afegir la funció d'API de sis_nova (nova)
  - include: servicios.h : Afegir la capçalera de la nova funció

Exercicis proposats

- Afegir la nova crida a sistema get_pid() i un procés d'usuari nou que la utilitzi. (seguir l'enllaç del campus virtual)