Adrien Forest
fjxokt@gmail.com
Christophe Carasco
carasco.christophe@gmail.com
Matthieu Maury
mayeu.tik@gmail.com

UPPSALA
UNIVERSITET
Department of Information Technology
Master Student

Operating System Project 10hp

SSIK : Simply & Stupidly Implemented Kernel

System Design

April 16, 2010

University year 2009-2010

Written with LaTeX

Contents

1. Kernel Structure

1.1 Kernel

This module hold all the global variable for the kernel, sure as:

- process list (`pready`, `prunning`, `pwaiting`, `pterminate`)
- `int kerror` the last kernel error
- `int *error` a global pointer to the last error. If you are in a user process, the error will be in the PCB, in the kernel this point to te `kerror` variable
- the list of existing program

This module also start the system and manage exception.

1.1.1 kernel.c

1.1.1.1 Declaration

This file hold the process list:

```
pls pready
pls prunning
pls pwaiting
pls pterminate
```

and the kernel error variable: `int kerror`

1.1.1.2 Functions

<++>: <+>+
parameter:<++>
return:<++>
job:<++>

kinit: `void kinit ( void )`
parameter: void
return: void
job: first function launched. Print informations, initialize some global variable, and spawn the init process

init: `void init ( void )`
parameter: void
return: void
job: finalize the initialization. Spawn the malta process and a shell

1.1.2 kexception.c

Handle the exception and interruption. They are all traped here, and this determine what to do.

1.1.3 kprogram.c

1.1.3.1 Declaration

A program is a structure like this :

```
typedef struct {
  char name[20] ;
  int  adress ;
  char desc[1024] ;
} prgm ;
```

The `name` is what you give to the `fourchette` syscall to spawn a process running this program. `adress` is the adresse of the first instruction, and `desc` is a description of the program (print by the help command).

You also will found the static list of program.

1.1.3.2 Functions

search: `prgm* search ( char *name )`
parameter: `*name` a pointer to a string, wich represent a possible programm name
return: a pointer to the program, or `NULL` if not found
job: this function will search a program into the program list

print_programs: `void print_programs ( void )`
parameter: void
return: void
job: print all the program with description

1.2 Process module

This module will manage all the process related functions.

1.2.1 kprocess.c

This file manage process individualy.

1.2.1.1 Declaration

A process is reprented by it's PCB :

```
typedef struct {
  int  pid
  char name[20]
  int  pri
  int  supervise[NSUPERVISE]
  int  superviser[NSUPERVISE]
  save (pc, registre ...)
  int  error
} pcb ;
```

We also need a structure to safelly pass some info without passing a pointer to the pcb:

```
typedef struct {
  int  pid
  char name[20]
  int  pri
  int  supervise[NSUPERVISE]
  int  superviser[NSUPERVISE]
  int  error
} pcbinfo
```

1.2.1.2 Functions

create_p: `int create\_proc (char *name, pcb *p )`
parameter: `name` the name of the program to launch, `p` the pointer to the pcb
return: the pid (>0), or an error (<0)
job: initialize a pcb with all the needed value, add it to the ready queue, and ask for a long term scheduling.

rm_p: `int rm_p ( pcb *p )`
parameter: `p` the process to delete
return: an error code
job: deallocate a pcb

chg_pri: `int chg_ppri ( pcb *p, int pri )`
parameter: `p` the pcb, `pri` la nouvelle priorité
return: an error code
job: change the priority of a process

get_pinfo: `int get\_pinfo ( pcb *p, pcbinfo *pi )`
parameter: `p` a pointer to the pcb that we need information, `pi` a pointer to a pcbinfo struct
return: an error code
job: this function copy and give the information of a pcb

copy_p: `int copy\_p ( pcb *psrc, pcb *pdest )`
parameter: the source pcb and the destination pcb
return: an error code
job: copy a pcb inside an other

add_psupervise: `int add_psupervise ( pcb *p, int pid )`
parameter: `p` a pointer to the process, the pid to add
return: an error code
job: add a pid to the supervise list of a process

add_psuperviser: `int add_psuperviser ( pcb *p, int pid )`
parameter: `p` a pointer to the process, the pid to add
return: an error code
job: add a pid to the superviser list of a process

rm_psupervise: `int rm_psupervise ( pcb *p, int pid )`
parameter: `p` a pointer to the process, the pid to add
return: an error code
job: remove a pid from the supervise list of a process

rm_psuperviser: `int rm_psuperviser ( pcb *p, int pid )`
parameter: `p` a pointer to the process, the pid to add
return: an error code
job: remove a pid from the superviser list of a process

1.2.2 kprocess_list.c

Manage a list of process

1.2.2.1 Declaration

```
struct pls {
  pcb ls[defined\_size]
  pcb *current
}
```

1.2.2.2 Functions

create_pls: `int create_pls ( pls *ls )`
parameter: a pointer to a list
return: an error code
job: initialize a list of pcb

rm_ls: `int rm_ls ( pls *ls )`
parameter: a pointer to a list
return: an error
job: delete a list of pcb

rm_from_ls: `int rm\_from\_ls ( pcb *p, pls *ls)`
parameter: the pcb to remove, and the list where he is
return: an error code
job: delete a pcb from a list and reorder the list

empty_space: `pcb* empty_space ( pls *ls )`
parameter: a pointer to a list of pcb
return: the first empty pcb
job: return the first empty space in a process list

is_empty: `bool is_empty ( pcb *p )`
parameter: a pcb
return: a boolean
job: is this pcb empty

seach: `pcb* search ( int pid, pls *ls )`
parameter: a pid and a process list
return: a pcb

job: search for a process in a list

seachall: `pcb* searchall ( int pid )`
parameter: a pid
return: a pcb
job: search for a process in all the lists

move: `int move ( int pid, pls *src, pls *dest )`
parameter: the pid we want to move, the source and dest list
return: an error code
job: move a process from a list to another (will search to ensure that the pcb is in the list)

sort: `int sort ( pls *ls )`
parameter: a process list
return: an error code
job: sort a process list by priority (highest to lowest)