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

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)

1.3 I/O

This module offers facilities to communicate with the serial console and the malta screen. This include structure to wrap them.

1.4 Message

The aim of this module is to allow processes to communicate between each others asynchronously. It provides some functions to send and receive a message, and a set of functions to manage a list of messages.

This module is dependent of the module Process in the sense that it need that two processes are created and know the pid of the other to send/receive messages.

To manage the messages, we have two structures:

msg which represents a message. The structure is made of:

- int msg_id the message identifier
- int pid_send the process identifier of the sender
- int pid_recv the process identifier of the receiver
- int pri the priority of the message
- void *user_data some user data included into the message

msg_lst which represents the list of msg received in each process. It is made of:

- msg *msg a message
- msg_lst *msg_lst a pointer to the next element in the list

1.4.1 Message functions (kmsg.c)

1.4.1.1 create_msg

- Description

  Create a new message object with the given information and send it to the receiver (i.e. add it to the message list of the receiver).

- Parameters

  int msg_id the message identifier

int pid_send the process identifier of the sender

int pid_recv the process identifier of the receiver

int pri the priority of the message

void *user_data some user data included into the message

- Return

  int the error identifier in case of any failure

1.4.1.2 receive_msg

- Description

  Wait for a message with a priority equals to *pri* from any process. Delete the message after received it.

- Parameters

  int timeout the maximum time a process is waiting for a message. 0 means infinite time.

  int pri the priority of the message to receive

  msg* the received message. null value if no such message has been received.

- Return

  int the error identifier in case of any failure

1.4.1.3 delete_msg

- Description

  Delete the message identified by msg_id from the message list.

- Parameters

  int msg_id the identifier of the message

- Return

  int the error identifier in case of any failure

1.4.2 Message list functions (kmsg_lst.c)

The following list functions are dependent of the message structure and functions.

1.4.2.1 create_msg_lst

- Description

  Create a new message list with no element.

- Parameters

  msg_lst* the new empty list

- Return

  void

1.4.2.2 add_to_msg_lst

- Description

  Add the message identified by msg_id to the end of the list lst.

- Parameters

  msg_lst* lst the message list

  int msg_id the message identifier

  msg_lst* the initial list plus the new message

- Return

  int the error identifier in case of any failure

1.4.2.3 rm_from_msg_lst

- Description

  Remove the message identified by msg_id from the list lst.

- Parameters

  msg_lst* lst the message list

  int msg_id the message identifier

  msg_lst* the initial list minus the message specified in parameter

- Return

  int the error identifier in case of any failure

1.4.2.4 rm_msg_lst

- Description

  Remove all elements in the list and delete the list.

- Parameters

  msg_lst* lst the message list

- Return

  int the error identifier in case of any failure

1.4.2.5 lookup_into_msg_lst

- Description

  Lookup the message identified by msg_id into the list lst.

- Parameters

  msg_lst* lst the message list

  int msg_id the message identifier

- Return

  bool true if the message is found in the list, false otherwise

1.4.2.6 sort_msg_lst

- Description

  Sort the message list according to the priority (highest first).

- Parameters

  msg_lst* lst the message list

  msg_lst* lst the sorted message list

- Return

  int the error identifier in case of any failure

1.4.2.7 empty_space_msg_lst

- Description

  Find the fist empty space in the array representing the list.

- Parameters

  msg_lst* lst the message list

  msg_lst* lst the message list

- Return

  int the error identifier in case of any failure

1.4.2.8 is_empty_msg_lst

- Description

  Specifies if the given list is empty or not.

- Parameters

  msg_lst* lst the message list

- Return

  bool true if the list is empty, false otherwise

1.5 Error

The aim of this module is to help the developer to diagnostic what occured in case of any failure.

1.5.1 Functions (kerror.c)

1.5.1.1 print_error

- Description

  Print the specified err_msg followed by the description of the error according to the global variable error_no.

- Parameters

  string err_msg the message the user wants to add to the error message

- Return

  void

In this module, we also define:

int *err_no Value of the last error that occured

1.6 System library

This module is dependent of the kernel library module beacause the following functions only perform syscalls to the kernel functions.

1.6.1 String functions (string.c)

1.6.1.1 strcpy

- Description

  Copy the string src to dest.

- Parameters

  char *src the source string
  char *dest the destination string

- Return

  int the error identifier in case of any failure

1.6.1.2 strcpyn

- Description

  Copy the length first characters of the string src to dest.

- Parameters

  char *src the source string
  char *dest the destination string
  int length the number of characters to copy

- Return

  int the error identifier in case of any failure

1.6.1.3 strcmp

- Description

  Compare the two string str1 and str2 to specify if str1 = str2 of which one of them is the first alphabetically.

- Parameters

  char *str1 the first string
  char *str2 the second string

- Return

  int 0 means str1 = str2, -1 means str1 < str2 and 1 means that str1 > str2

1.6.1.4 strcmpn

- Description

  Compare the first n characters of the two string str1 and str2 to specify if str1 = str2 of which one of them is the first alphabetically.

- Parameters

  char *str1 the first string

  char *str2 the second string

  int n the number of characters to compare

- Return

  int 0 means str1 = str2, -1 means str1 < str2 and 1 means that str1 > str2

1.6.1.5 strlen

- Description

  Specify the number of characters of the string str

- Parameters

  char *str the string

- Return

  int the length of the string. -1 if str invalid.

1.6.1.6 strchr

- Description

  res is a pointer to the first occurrence of character c in the string str.

- Parameters

  char *str the string

  char c the character to find

  char *res the substring (result)

- Return

  int the error identifier in case of any failure

1.6.1.7 isspace

- Description

  Checks if parameter c is a white-space character (SPC, TAB, LF, VT, FF, CR).

- Parameters

  char c the character to evaluate

- Return

  bool true means that c is a space character, false otherwise

1.6.2 Display functions (stdio.c)

1.6.2.1 printf

- Description

  Print the string str to the standard output.

- Parameters

  char *str the string to print

- Return

  int the error identifier in case of any failure

1.6.2.2 fprintf

- Description

  Print the string str to the specified output.

- Parameters

  int out the output where to print (0 = console, 1 = Malta)
  char *str the string to print

- Return

  int the error identifier in case of any failure

1.6.2.3 getc

- Description

  Returns the character currently pointed by the internal file position indicator of the input stream (keyboard).

- Parameters

  void

- Return

  char the character read

1.6.2.4 fgets

- Description

  Reads characters from stream and stores them as a string into str until (num-1) characters have been read or either a newline or a the End-of-File is reached, whichever comes first. stream (keyboard). A null character is added to the end.

- Parameters

  char *str the string read from the input stream (keyboard)
  int num the number of charaters to be read

- Return

  int the error identifier in case of any failure

1.6.3 Error codes (error.h)

SUCCESS No error occured

OUTOMEM Out of memory

UNKNPID Unknown pid (process identifier)

UNKNMID Unknown mid (message identifier)

INVPRI Invalid priority

OUTOPID Out of pid (number of processes is limited)

OUTOMID Out of mid (number of messages is limited)

NULLPTR Null pointer error

INVEID Invalid eid (error identifier)

1.7 Kernel library

1.7.1 Functions (klib.c)

1.7.1.1 kprintf

- Description

  Print the string str to the standard output.

- Parameters

  char *str the string to print

- Return

  int the error identifier in case of any failure

1.7.1.2 kfprintf

- Description

  Print the string str to the specified output.

- Parameters

  int out the output where to print (0 = console, 1 = Malta)
  char *str the string to print

- Return

  int the error identifier in case of any failure

1.7.1.3 kgetc

- Description

  Returns the character currently pointed by the internal file position indicator of the input stream (keyboard).

- Parameters

  void

- Return

  char the character read

1.7.1.4 kfgets

- Description

  Reads characters from stream and stores them as a string into str until (num-1) characters have been read or either a newline or a the End-of-File is reached, whichever comes first. stream (keyboard). A null character is added to the end.

- Parameters

  char *str the string read from the input stream (keyboard)

  int num the number of charaters to be read

- Return

  int the error identifier in case of any failure

1.7.1.5 kisspace

- Description

  Checks if parameter c is a white-space character (SPC, TAB, LF, VT, FF, CR).

- Parameters

  char c the character to evaluate

- Return

  bool true means that c is a space character, false otherwise

1.7.1.6 kstrchr

- Description

  Update res which is a pointer to the first occurrence of character c in the string str.

- Parameters

  char *str the string

  char c the character to find

  char *res the substring (result)

- Return

  int the error identifier in case of any failure

2. OS API

The API (Application Programming Interface) for user programs in your operating systems.

2.1 Functions

2.1.1 fourchette

- parameters

  char *name program name that will be executed

  int prio program name that will be executed

- return

  int pid pid

- description

  fourchette() creates a child process that differs from the parent process only in its PID and PPID. If success, the PID of the child process is returned in the parent's thread of execution, and a 0 is returned in the child's thread of execution.

2.1.2 get_proc_info

- parameters

  int pid process pid

  pcb_info *res pcb_info structure

- return

  int error error returned if something went wrong

- description

  get_proc_info() fill the pcb_info structure given in parameter with the pcb information. Only not critical information is given to the user.

- structure used : pcb_info

  This structure is made of:

  - pid process id
  - name process/program name
  - pri process priority
  - ls_supervise list of supervised processes
  - ls_superviser list of superviser processes

2.1.3 chgpri

- parameters

  int pid process pid

  int prio process's new priority

- return

  int error error returned if something goes wrong

- description

  chgpri changes the priority of the process from the old one to the new priority 'prio'.

2.1.4 sleep

- parameters

  int time sleep for the specified number of milliseconds

- return

  void

- description

  sleep() makes the current process sleep until 'time' milliseconds seconds have elapsed.

2.1.5 wait

- parameters

  int *status sleep for the specified number of milliseconds

- return

  void

- description

  The wait() function suspends execution of its calling process until status information is available for a terminated child process.

2.1.6 sendmsg

- parameters

  int msg_id message identifier

  int pid_sender pid of the sender

  int pid_receiver pid of the receiver

  int priority message priority

  void *user_data user data

- return

  int error error returned if something went wrong

- description

  The sendmsg() sends a message from a process to another. User data can be attached to the message using the user_data pointer.

2.1.7 recvmsg

- parameters

  int timeout the maximum time a process is waiting for a message. 0 means infinite time.

  int pri message priority

  msg *message message to be send

- return

  int error error returned if something went wrong

- description

  Wait for a message with a priority equals to pri.

2.1.8 perror

- parameters

  char *msg error message to print

- description

  the perror() function writes the last error that occured followed by a newline, to the standard output. If the argument string is non-NULL, this string is prepended to the message string and separated from it by a colon and space (": "); otherwise, only the error message string is printed.

2.2 How is the OS API invoked?

The OS API is invoked by system call. Implementing system calls requires a control transfer which involves some sort of architecture-specific feature. A typical way to implement this is to use trap. Interrupts transfer control to the OS so software simply needs to set up some register with the system call number they want and execute the software interrupt.

2.3 How are programs represented?

The programs are represented by their address located in the program table. The program table is the table which associated the program name with the code location in memory.

2.4 How can programs that need to communicate locate each other?

To communicate, programs can send messages between each other. To send a message, a program has to know the pid of it's target.

3. User programs

This chapter will describe all the user programs available in our Operating System.

3.1 Increment

Print the sequence 1, 2, 3, ..., n to the console, each number on a new line.

3.2 Fibonacci

Prints the Fibonacci number series, each number on a new line. Each term of the Fibonacci number series is the sum of the two previous ones. It starts with the first two numbers 0 and 1 and goes like this: 0, 1, 1, 2, 3, 5 and so on.

3.3 Command shell

Shell that allows user to do some operations on the processes. It will be able to do at least:

- Start processes
- Change priority of processes
- Obtain information about present processes
- Terminate processes
- Ouput to the Malta LCD display

3.4 Text Scroller

Scrolls text on the Malta board display. If the text that must be printed doesn't fit in the Malta LCD display, it will be scrolled until it has been completely printed, and then the scrolling will restart at the beginning of the text.

- The process provide for smooth scrolling even on a highly loaded system
- The scroller is a regular user process with high priority
- The scroller sleep between updates to the display
- The scroller start when the operating system starts

If the number of characters to be printed is higher than the number of characters the display can print, the program will start by printing the first characters, and then will start again, starting from the second character, and so one until the last charactere of the text is printed first.

3.5 Ring

User program that demonstrate that message passing communication is working

- A program should start a set of other processes, P_1 to P_n
- The processes should be set up in a communications ring, where P_1 sends messages to P_2, etc. on to P_n
- The demo will send some messages around the ring and show that they visit all processes along the way

3.6 Dining philosophers

User program that demonstrate that process synchronisation is working The purpose of this program is for the pilosophers to eat, using shared forks. A philosopher can only eat when he has two forks. Otherwise he will have to wait for another philosopher to release one of his to start eating.

3.7 Process supervision

User program that demonstrate that process supervision is working.

- Processes can be appointed as supervisors of one or more other processes
- When a supervised process terminates, the supervisor is notified
- It is possible to differentiate between controlled and uncontrolled termination, i.e. it is possible for the supervisor to see if a subordinate process has crashed or if it terminated in good order

The demo include a supervisor that restarts its subordinates if they crash.

3.8 ps

User program that displays the currently-running processes. It will look inside the running list.

3.9 malta echo

User program that print a predefined text on the Malta LDC display.

3.10 help

User program that print the different programms that a user can execute.

3.11 Memory management (optional)

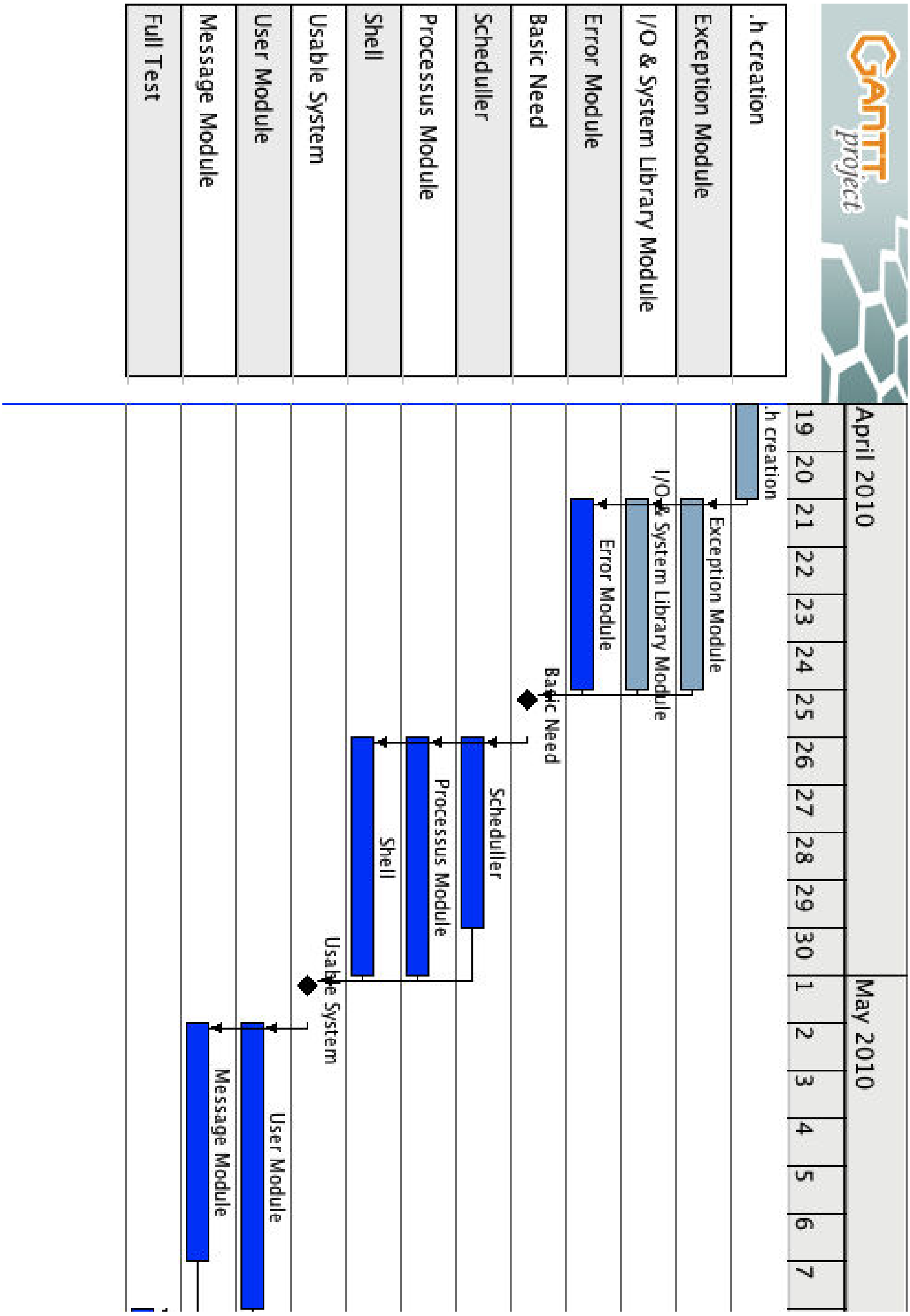
User program that demonstrate that memory management is working.

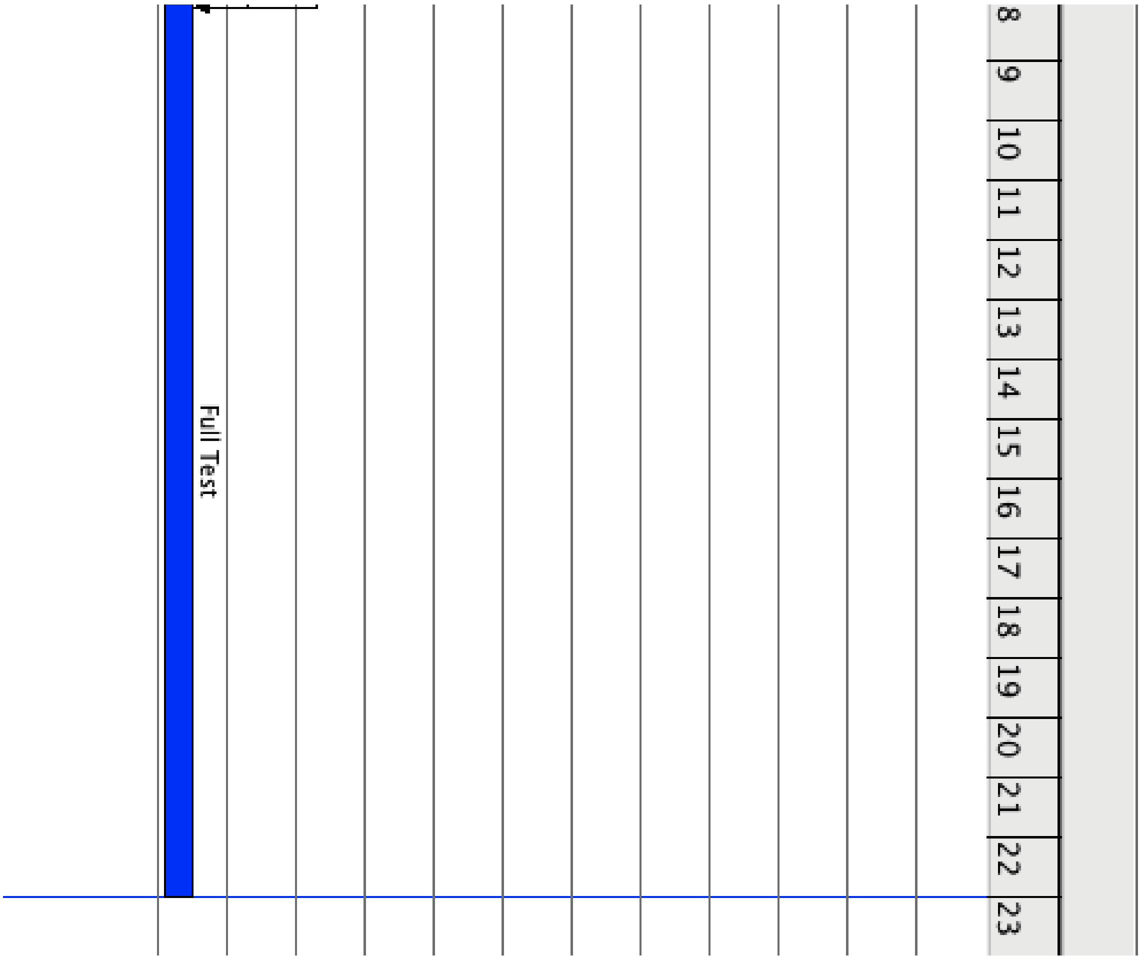
4. Schedule

We separate the code task as :

- Adrien Grand : Error, Shell
- Christophe Carasco : Exception, Process
- Matthieu Maury : I/O & System Library, Scheduler
- all three : user software, message

To illustrate the order of implementation, dependence and time, we have done a Gantt Diagram :


GANTT project
.h creation
Exception Module
I/O & System Library Module
Error Module
Basic Need
Scheduller
Processus Module
Shell
Usable System
User Module
Message Module
Full Test
April 2010
May 2010
19
20
21
22
23
24
25
26
27
28
29
30
1
2
3
4
5
6
7
.h creation
Exception Module
I/O & System Library Module
Error Module
Basic Need
Scheduller
Processus Module
Shell
Usable System
User Module
Message Module


Full Test
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23