Manufacturing Cell Monitoring Process

Carlos Martins *CCP* and Rui Rocha *CCP*

***Abstract* - This paper describes all the manufacturing cell monitoring process particularities, and its several features. It is also done a brief presentation of the data structure used to monitor the manufacturing cell status, and it is explained its achievement and its philosophy by a complete flowchart.**

1.Introduction and brief description

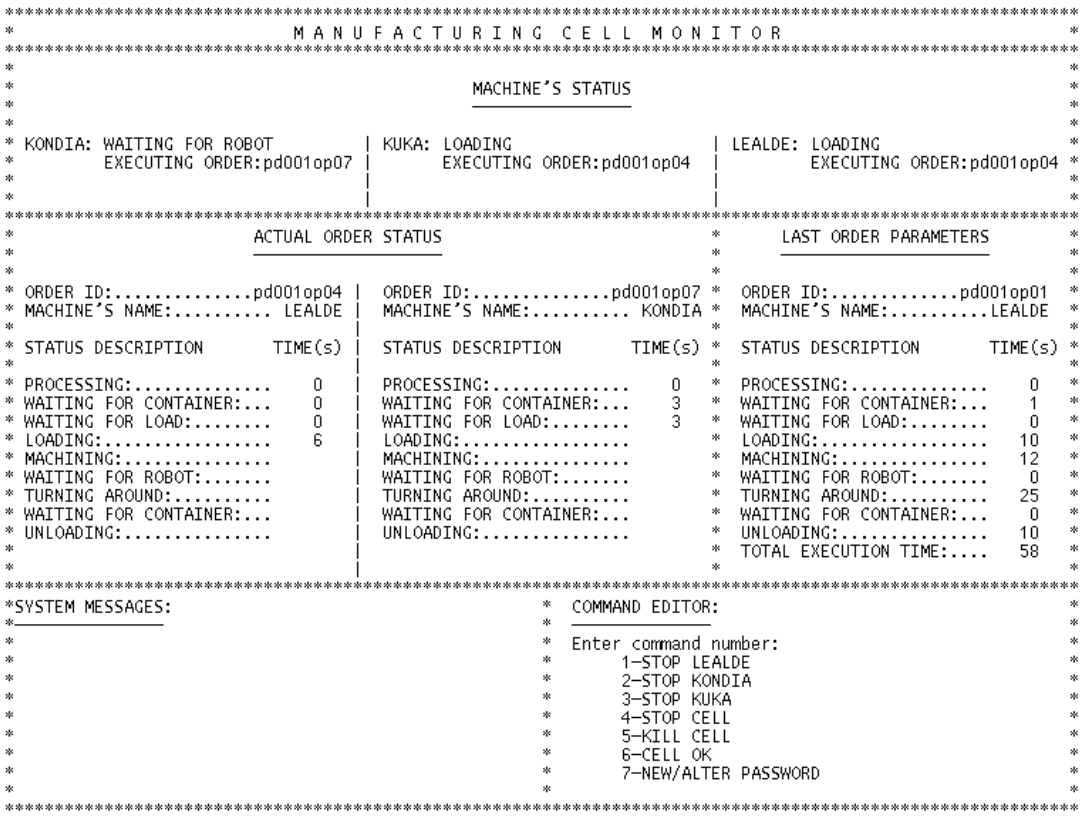


Figure 1 - Monitor appearance

The monitoring process of the manufacturing cell deals with the visualisation in real time of the manufacturing orders evolution, which are running at the moment, as well as the current state of the several machines (Kondia, Lealde and Kuka).

For each manufacturing order we can see the time spent in the last states, as well as the time spent in the current state. In a moment, we can have a maximum of 2 orders (see figure 1). When an order finishes, all the status times are added, and it is shown the total execution time. At that moment, all the information about that order are transferred to a display zone called "Last Order Parameters", liberating the display zone that was occupied by that ended order. Those information are the order's identification, the machine associated with that order, the time spent in the several states (processing, waiting for container, waiting for robot, loading, machining, piece turning around, waiting for robot to make the unload or the turning around operation, waiting for container to unload and unloading) and the total execution time.

Synchronised with the order's evolution, it is possible to observe the machine´s status and associate them with the orders. For instance (see figure 1): while a piece is being loaded in the Lealde by the Kuka robot, we can see in the "Loading" state of an hipotetic order "pd001op04", the time running, and at the same time, observe that the Lealde's state is "Loading" and that the Kuka's state is also "Loading", and observe on the bottom of these messages "Executing Order: pd001op04".

Simultaneously, it exists a display's zone called "System Messages", where are shown all the error and warning messages, associated with possible manufacturing cell controller abnormal situations, as file opening errors, MMS errors, password incorrect introducing, and so on. This zone has the capacity to show the eight most recent messages.

Another feature of this monitoring process is the "Command Editor", which makes possible to send commands by the user to the main controller (in a mailbox), after the introduction of the security password. This editor permits, for instance, stop a machine, stop the manufacturing cell, make a setup or alter the security password.

When the user strikes any key, the monitoring process creates automatically a child process, that begins by verifying if the command is valid. If it is not valid, it is generated a related error message, and the child process kills itself. If it is valid, the user must insert a password. If it doesn't exist any password selected, the user must insert one before the execution of any other command, otherwise it is generated an error, and the child process kills itself. If the user takes more than 20 seconds on entering the password, it is generated an error, and the child process kills itself. After the introduction of the correct password, it is sent a message in a mailbox to the main controller, except if the command is an introduction of the first password, or a simply alteration of the current. The child kills itself in both situations.

While the child process of the command editor is executing its own tasks, the monitoring process continues its job, altering its variables, which are a virtual image of the manufacturing cell status. As soon as the child process finishes, the monitoring process refreshes all the display.

If the child process doesn't exist, any alteration in the manufacturing cell can be observed in the display, in real time.

2. Programmation strategiy used in the monitoring process

2.1 Messages types changed between main controller and monitoring processes

The messages sent by the main controller to the monitoring process may be grouped into three classes, according to their semantics:

1. an order that is running on the manufacturing cell changed its actual status;
2. an system error or warning occurred (problem on opening a file, for example);
3. the manufacturing cell controller is going down.

The message is composed of a structure of type *t_mensagem*, declared as follows:

```
typedef struct t_mensagem{
    int num1;
    int num2;
    int num3;
    int label_erro;
    char frase[50];
} t_mensagem;
```

and a variable *"mtype"* of type *long.*

The resulting message is a structure of type *msgbuf1*:.

```
typedef struct msgbuf1{
    long mtype;
    t_mensagem mensagem;
} msgbuf1;

msgbuf1 buf;
```

The field *"mtype"* is used to distinguish the messages according to its semantics. So:

- mtype=FIM - the message's class is 3;

- mtype!=FIM - the message's class is either 1 or 2. In this case it's a message, that corresponds to the normal mode of the controller status.

If mtype!=FIM the message may be of class 1 or 2:

- class 1 : - The information is distributed by the structure buffer fields as bellow:

  buf.mensagem.label_erro=0;

  buf.mensagem.num2="*idmaq*";

  buf.mensagem.num3=*"actual_status"*

  buf.mensagem.frase="*idordem"*;

- class 2 : - The field *buf.mensagem.label_erro* contains a label, whose values are declared in "*visual.h"* and identifies the system error, or warning that has occurred. All these label values differ from zero.

2.2 Monitoring process description

The monitoring process starts by initialisation the monitoring window, as shown in figure 1 (in the introduction).

After that, it jumps into a loop to receive the messages from the mailbox *´msq_cp_monitor'*. This loop is broken if a class 3 message is received, indicating that all the cell controller's processes and mailbox will be destroyed.

In each loop, one attempt to read the mailbox will be made. If there was no message to be read the process will just actualise the time of both actual order's status, and print them in the correct place of the screen. In order to execute this request, the function *visualiza_ordem(0,buf.mensagem.frase,buf.mensagem.num2,buf.mensagem.num3)* will be called. If it was found a class 2 message in the mailbox, the function *processa_erros(buf.mensagem.label_erro)* will be called, in order to print a message in the "SYSTEM MESSAGES" box of the screen. Finally, the function *visualiza_ordem(1, bufmensagem.frase,buf.mensagem.num2,buf.mensagem.num3)* will be called, if there was a class 1 message.

The function *visualiza_ordem* uses, among other ones, the variables listed bellow:

```
typedef struct {
  char idordem[50];
  int idmaq;
  int estado_actual;
  unsigned long t_estados[NUM_ORD_ESTADO];
  unsigned long n_seg;
```

```
} estr_estado;

typedef struct {
  int estado_maq;
  char idordem[50];
} estr_maq;

estr_estado ord_buffer[3];
estr_maq maq_buffer[NUM_MAQ];
```

where:

- NUM_ORD_ESTADO: - is the number of status of an order. For the present implementation, this value is 12;
- NUM_MAQ: - is the number of machines available at the manufacturing cell. In the present application, this value is 3.

2.3 Table of order's status and times

The array of three positions *ord_buffer* contains in positions 0 and 1 the information related with the orders that are running in the cell, and in position 2 the information related with the most recent order that has finished, according to the table in figure 2.

array index	idordem	idmaq	estado_actual	n_seg	t_estados[]										
0	pd00op2	LEALDE	MACHINING												
1	pd00op3	KONDIA	LOADING												
2	pd00op1	LEALDE	COMPLETE												

Figure 2 - The "ord_buffer" array

Meaning of the several structure fields:

- idordem: - identifier of the order;
- idmaq: - the machine related with the manufacturing order;
- estado_actual: - actual status of the order;

- n_seg: - stores the moment (specified by the system function *time(0)*) in which the time associated with the status *estado_actual* was incremented most recently. This variable is used to decide if one second has elapsed since the last turn, in which the time was displayed for the actual order status. In affirmative case, the time shown is actualised on the screen;
- t_estados[]: - stores the time that the order spent in each state. If there is a state by which the order didn't pass, the related field on the array will be filled with the value -1;

  t_estados[0]- time elapsed while the order was in state PROCESSING;

  t_estados[1]- time elapsed while the order was in state WAITING FOR CONTAINER;

  .....

  .....

  t_estados[NUM_ORD_ESTADO-1]- time elapsed while the order was in state UNLOADING;

2.4 Table of machine's status and related orders

The array *maq_buffer* (see figure 3) stores the machine's status (idle / occupied) for each machine in the manufacturing cell, as well as the associated order identifier.

Machine	*estado_maq*	*idordem*
KONDIA	idle	.....
LEALDE	occupied	pd00op1
KUKA	occupied	pd00op1

Figure 3 - The "maq_buffer" array

2.5 Function "visualiza_ordem"

This function (and some other ones called inside) is used to update the information contained in the previous two tables, to display the machine's status and order's status times in the screen and to produce the desired result discussed in the introduction point.

2.6 Command editor process

When the command editor process is created, and as well as it exists, the controller evolution cannot be seen in the screen, because the cursor cannot be at two different positions on the window. In this case, the evolution is only registered in the tables shown in figures 2 and 3.

When, finally, the process is killed, the screen will be actualised with the information contained in the tables.

To implement the mechanism that avoids the cursor motion by the two concurrent processes at the same time, the follow strategy was taken:

- Two semaphores were created at the beginning of the monitoring process: semaphore 0 and semaphore 1.

  Semaphore 0 is used to control the access to the screen for write operations. Its status, according to the circumstances, is the following:

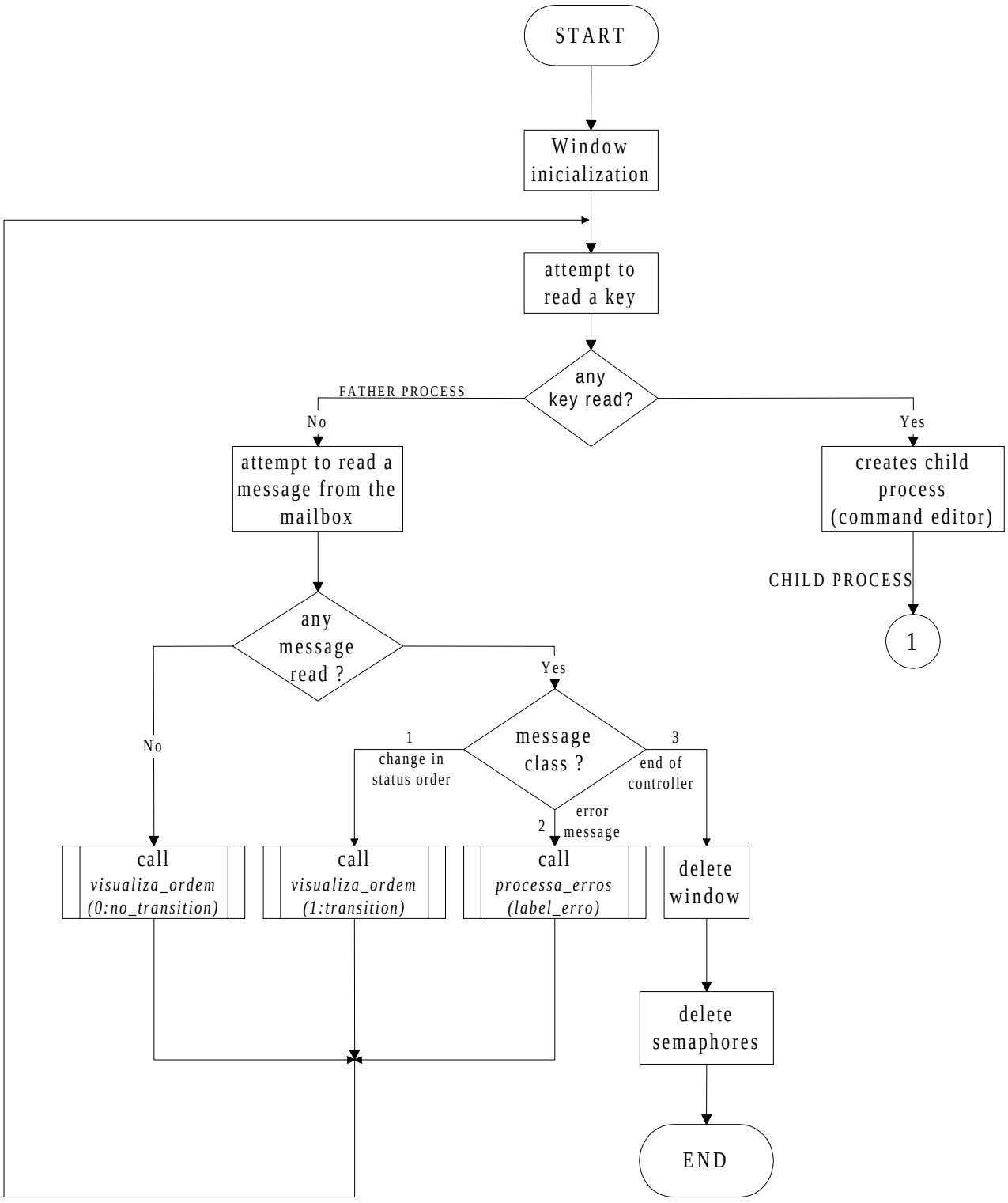
  Marked: - when the command editor process does not exist, or an instruction to write in the screen is not reached;

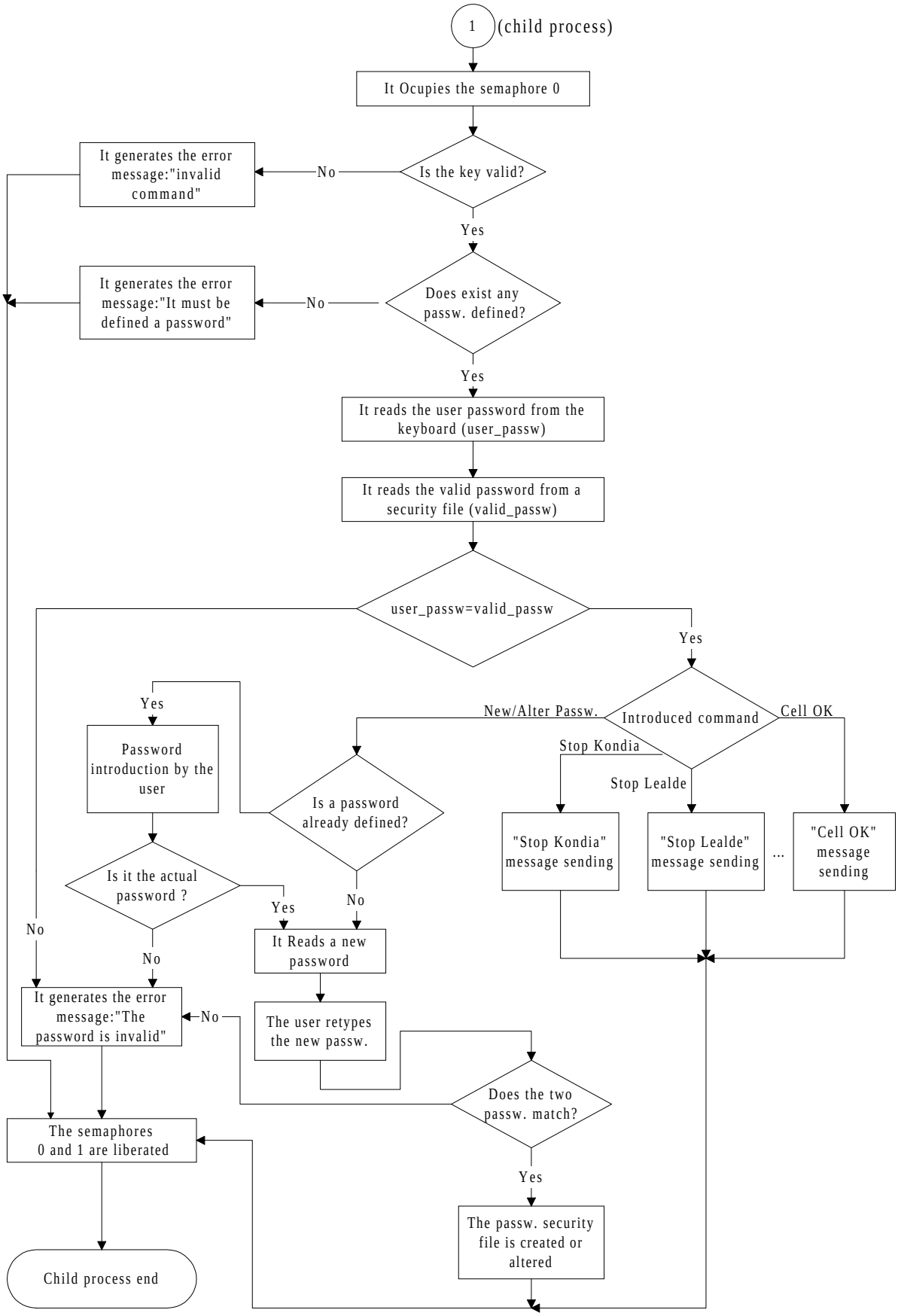
  Mark removed: - when the user strikes a key or the program reaches an instruction to write in the screen;

  Mark attributed: - when the command editor process is killed or the write instruction ends;

  Semaphore 1 is used to "tell" the monitoring process that the child process was killed and the screen must be refreshed with the actual information, using function *actualiza_ecran*. By default, this semaphore is not marked. The mark will be put back when the command editor process is killed, and will be removed when the function "*actualiza_ecran"* finishes.

3. Monitoring process flowchart




1
(child process)
It Ocupies the semaphore 0
Is the key valid?
No
It generates the error message:"invalid command"
Yes
Does exist any passw. defined?
No
It generates the error message:"It must be defined a password"
Yes
It reads the user password from the keyboard (user_passw)
It reads the valid password from a security file (valid_passw)
user_passw=valid_passw
No
Yes
Introduced command
New/Alter Passw.
Cell OK
Stop Kondia
Stop Lealde
"Stop Kondia" message sending
"Stop Lealde" message sending
...
"Cell OK" message sending
Yes
Password introduction by the user
Is a password already defined?
No
Is it the actual password ?
Yes
No
It Reads a new password
The user retypes the new passw.
Does the two passw. match?
No
Yes
It generates the error message:"The password is invalid"
No
The semaphores 0 and 1 are liberated
The passw. security file is created or altered
Child process end

4 - Conclusions

With the previous monitoring strategy, it is possible to display in “on-line” the most relevant activities occurring in the cell. Since all the orders are registered in a text file, the activities may also be checked later.

Data related with the time of execution of an order may be used to create statistics about the global performance of the cell.