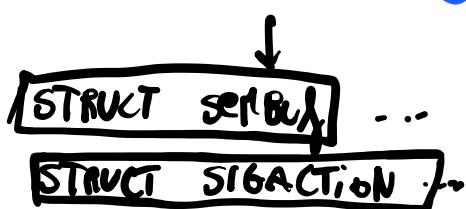


# SIGACTION

# handle = None



```

int main(int argc, char** argv)
{
    sigset(SIGINT, handle); // MASCHERA gli BIT
    struct sigaction act;

    sigfillset(&act); // PASSA LA MASK A ZERO I BIT
    act.sa_sigaction = handle; // Nome del gestore
    act.sa_mask = 0; // SET
    act.sa_flags = SA_SIGINFO; // SEMPRE NULL
    act.sa_restorer = NULL; // SEMPRE NULL
    sigaction(SIGINT, &act, NULL);
    sigprocmask(SIG_BLOCK, &act.sa_mask, NULL);
}

```

→ system("%s", "CAT more")  
SA\_NODEFER

↳ SIG\_IGN = SIG\_IGN  
SIGNAL(SIGINT, handle)

SETTAGGIO

```

----  

CTRL + C  

sigpending(&act);
if(sigismember(&act.sa_mask, SIGINT))
{ // solo se premo CTRL + C
    contatore_gestione_segnale =
    contatore_gestione_segnale + 1;
    sigemptyset(&act.sa_mask);
    sigaddset(&act.sa_mask, SIGINT);
    if(contatore_gestione_segnale == 1)
        sigprocmask(SIG_UNBLOCK, &act.sa_mask, NULL);
    sigprocmask(SIG_BLOCK, &act.sa_mask, NULL);
}

```

TUTTI quelli pendenti, c'è uno  
SIGINT?  
RISVUOTO la MASCHERA  
1 = SIGINT  
01000

SEGN 1  
>>  
SEGN

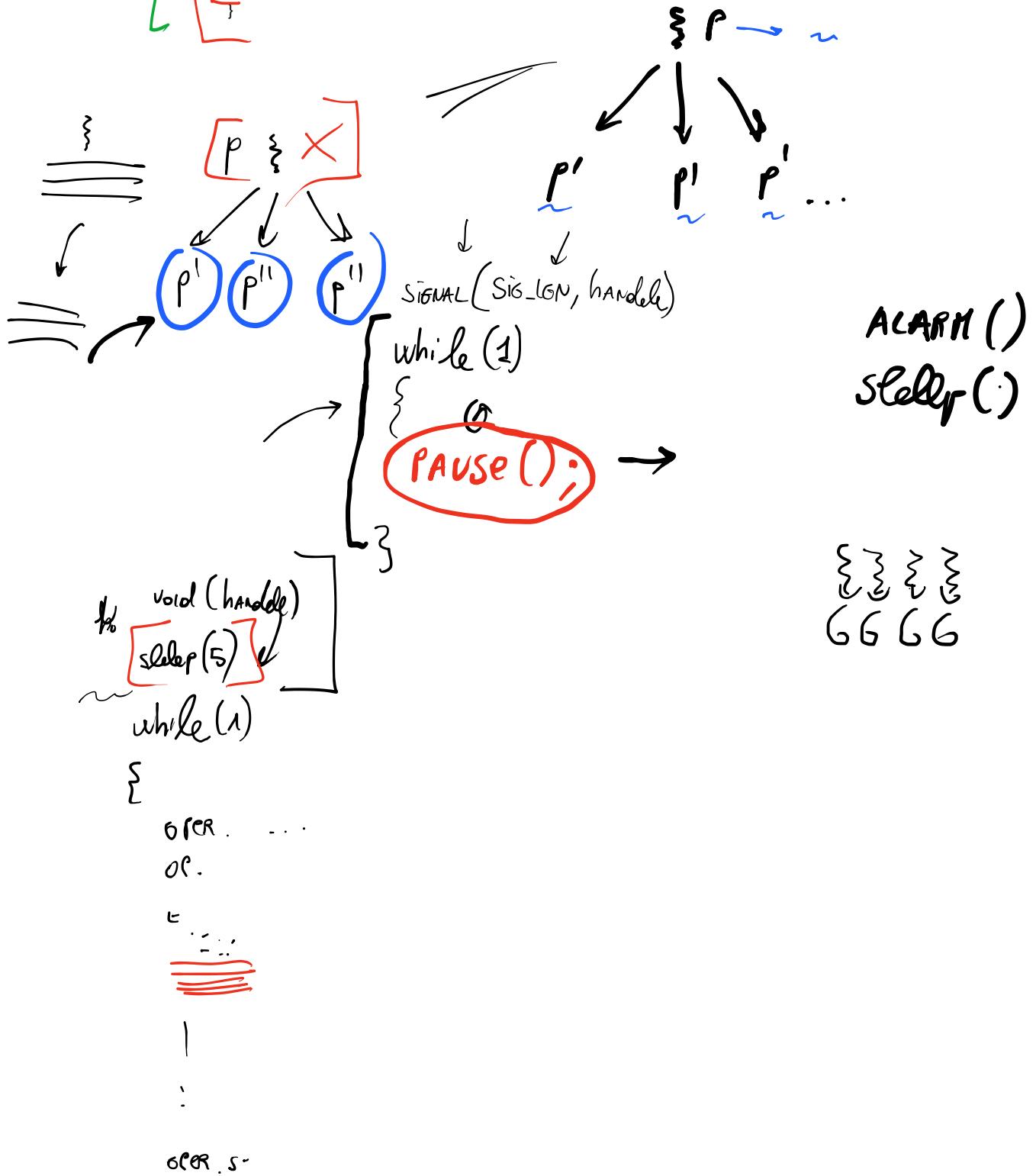
FINI

... più tardi

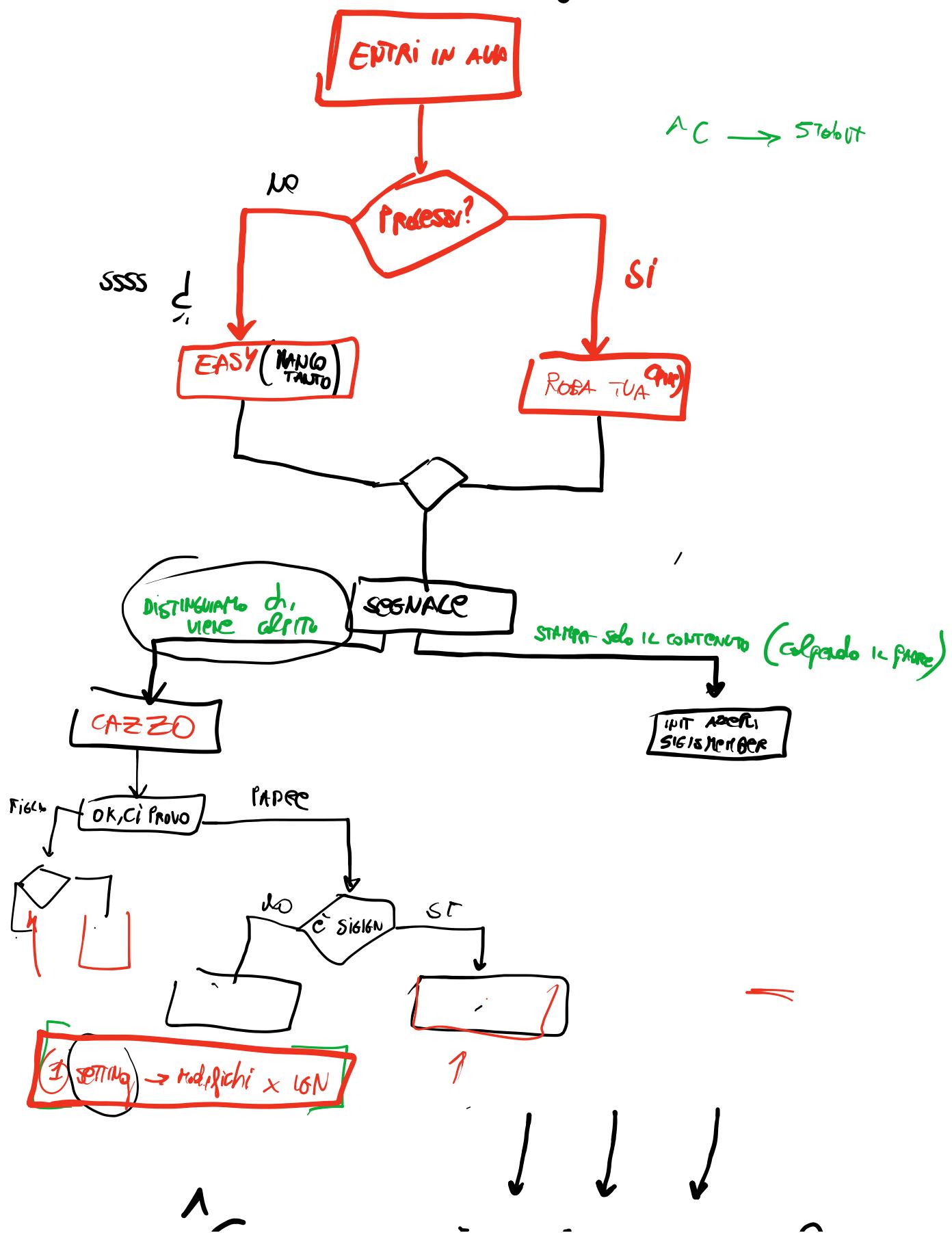
```

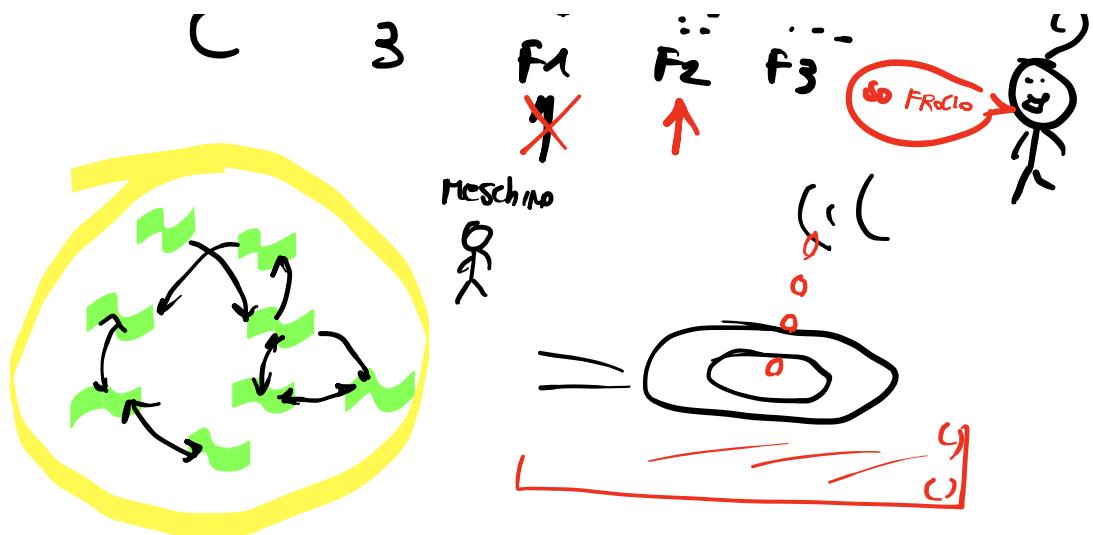
sigpending(&set); // ILLUMINA
if(sigismember(&set,SIGINT)) // TRA TUTTI I SIGNAL PENDENTI, c'è anche SIGINT?
{
    printf("\n");
    sprintf(buffer, "cat %s", file_names[pid+(pid+1)]);
    printf("\n-----\n");
    printf("Contenuto del %s che correntemente è\n"
           "gestito : [%s]\n", file_names[pid+(pid+1)]);
    system(buffer);
    sigemptyset(&set);
    sigaddset(&set,SIGINT);
    sigprocmask(SIG_UNBLOCK,&set,NULL);
    sigprocmask(SIG_BLOCK,&set,NULL);
}

```



# COME CAZZO LO FA?





The diagram illustrates the memory layout for the `fork()` operation. It consists of two columns of memory blocks. The left column contains four blocks, each with a size of 3 and a starting address of  $i$ . The right column also contains four blocks, each with a size of 3 and a starting address of  $i$ . A vertical brace on the right side groups the first three blocks from both columns, while the fourth block from the left column stands alone.

$O(m)$ ,  $O(\log m)$   $\subseteq O(m)$

↓

Ricorsivo  $\times$   
Individuato ;

\* PCTR =  $f(a)$

PUNTATORI

The diagram illustrates the state of memory after the execution of the following C code:

```

[ INT * PTR = &a; ]    MALLOC
INT PTR = MALLOC( sizeof(INT*) )
FILE PTR = MALLOC( sizeof(FILE*) )

```

Two boxes at the top left show the state of memory before and after the first allocation:

- Before Allocation:** A box containing `INT * PTR = &a;` with a bracket under `PTR = &a;` and an oval labeled `MALLOC` to its right.
- After Allocation:** A box containing `INT PTR = MALLOC( sizeof(INT*) )` with a red bracket under `MALLOC( sizeof(INT*) )`.

Below these, two more boxes show the state of memory after the second allocation:

- After First Allocation:** A box containing `FILE PTR = MALLOC( sizeof(FILE*) )` with a green bracket under `MALLOC( sizeof(FILE*) )`.
- Final State:** A box containing `FILE PTR = MALLOC( sizeof(FILE*) )` with a green bracket under `MALLOC( sizeof(FILE*) )`.

A large green arrow points from the bottom left towards the final state, indicating the flow of control. The word `Allocn` is written in a green box at the bottom center. To the right, there is a partial view of another diagram labeled `B`.

