Teste - 12/6/15 - SO

II

```
#define SIZE 1024
char buffer[SIZE];
int time = 0, mRead = 0;

void hAlarm(){
    write(1, &buffer, mRead);
    mRead = 0;
    buffer[0] = '\0';
}

void hAumenta(){
    time++;
}

void hDiminui(){
    time--;
    if(time < 0){
        time = 0;
    }
}

void atrasa (int tempo){
    int r;
    time = tempo;
    mRead = 0;

    signal(SIGALRM, hAlarm);
    signal(SIGUSR1, hAumenta);
    signal(SIGUSR2, hDiminui);
    alarm(time);

    while((r = read(0, buffer+mRead, 1)) > 0){
        mRead++;
        if(buffer[mRead-1] == '\n'){
            alarm(time);
            pause();
        }
    }
    pause();
}
```

```
int main (int argc, char **argv){
    if (argc == 1)
        Return -1;
    atnasa ( atoi (argv[1]);
    Return 0;
}
```

III

CRIA PIPE

```
int main(){
  int i= mkfifo ("ordenar", 0666);
  if(i<0){
    perror("\nERRO");
    return -1;
  }
  Return 0;
```

SORT

```
int main (int argc, char **argv) {
    int fd-in = open ("ordenar", O_RDONLY);
    int fd-file , fd-out;
    char buffer [128] , file_name [128];
    int R;
    if ( fd_in > 0 ){
        R = Read ( fd-in , buffer , 124);
        if ( R>0 ) {
            buffer [R-1] = '\0';
            fd-file = open(buffer,O_RDONLY);
            strcpy (file_name, buffer);
            strcat (file_name, ".sorted");
            fd-out = open ( file_name , O_CREAT | O_WRONL
        }
```

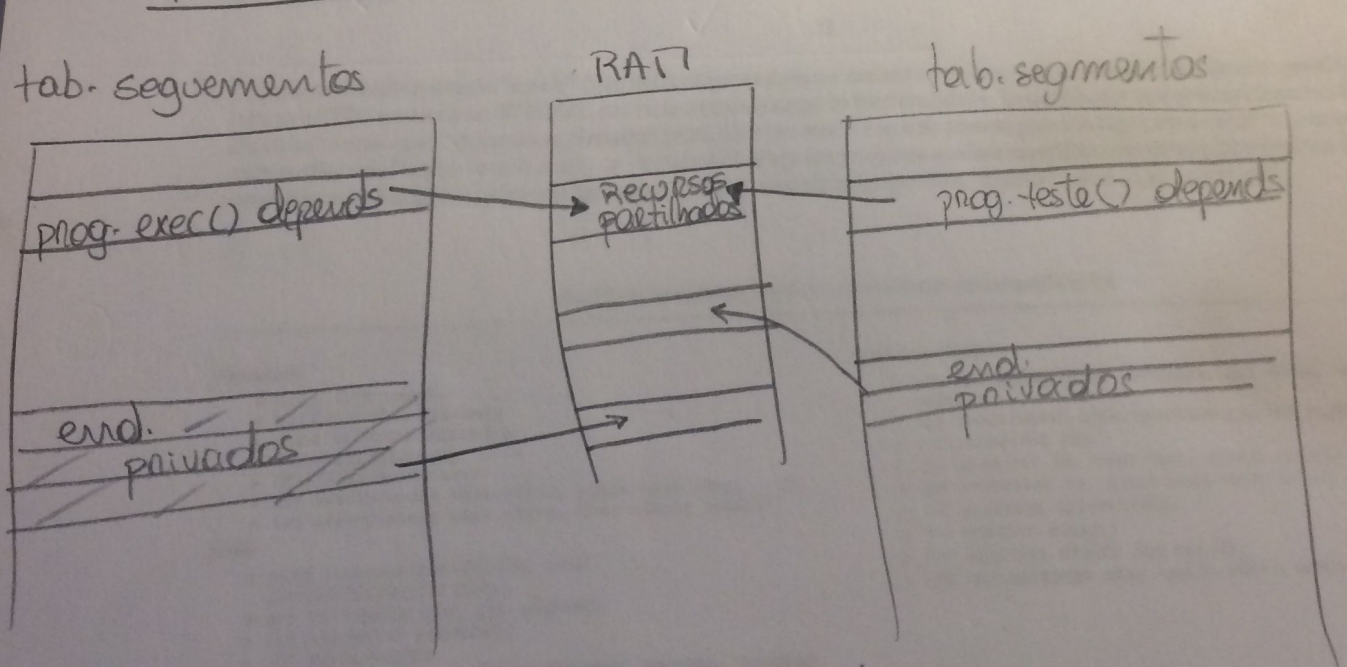

```
if (fd_out > 0) {
        if (!fork()) {
            dup2(fd_file,0);
            close(fd_file);
            dup2(fd_out, 1);
            close(fd_out);
            execlp("sort", "sort", NULL);
            exit(-1);
        }
    close(fd_out);
    close(fd_file);
}
close(fd_in);
}
```

!. fork() == fork()

Teoria: ① Partilhar um conjunto de end. memória por vários processos e simultaneamente garantir q cada processo mantém as suas próprias vars. locais e vars. globais.

② Quando criamos um novo processo c/ a system call fork(), o filho e o pai, no início vão possuir a mesma memória, visto q a quando da criação do filho a memoria do pai é copiada p/ o filho. E a partir daí cada um vai possuir as suas próprias variáveis.

R: Também conseguimos garantir isso através da implementação da memória virtual. O espaço de endereçamento q 1 processo pensa q tem, e as endereços são emitidos pelos CPU ao executar instruções, são trocados antes de serem colocados no barramento de endereços de memória Real. Como? Procurando numa tabela de segmentos q contém o modo de acesso permitido e o endereço real em memória, se estiver carregado em RAM.



O processo 1 e 2 ambos precisam de aceder ao mesmo endereço de memória, pois possuem endereços partilhados, mas todas as outras variáveis são independentes pois cada processo faz coisas diferentes.