Le programme ci-dessous implémente une version multi-processus du jeu de la vie. Dans ce jeu, les cellules d'un plateau à deux dimensions sont soit en vie soit morte. Le status des cellules évolue au court du temps. Les régles qui détermine si une cellule doit vivre ou mourir à l'étape suivante sont:

1. Une cellule morte possédant exactement trois voisines vivantes devient vivante (elle naît).
2. Une cellule vivante possédant deux ou trois voisines vivantes le reste, sinon elle meurt.

Questions:

1. Combien de processus ce programme génère-t-il ? Décrire le rôle et les actions de chaque processus.
2. Dans le cas ou le processus principal (parent) meurt qu'arrive-t-il aux enfants de ce processus ?
3. quelle est l'utilité de la fonction sched_yield (ligne 59) dans ce programme.
4. Y-a-t-il des conflits dans les ressources utilisées par les différents processus ? Expliquez.
5. Imaginons que ce programme tourne sur un processeur unique, que ce passerais-t-il si les lignes 85-87 n'étaient pas commentées.

PS: Attention cette implémentation ne correspond pas sctrictement au jeux de la vie

life.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/mman.h>
5  #include <sys/stat.h>
6  #include <fcntl.h>
7  #include <sched.h>
8
9  #define NB_LIGNES 10
10 #define NB_COLS 10
11 #define MEM_NAME "/plateau"
12
13 void exit_err(const char* str) {
14     perror(str);
15     exit(EXIT_FAILURE);
16 }
17
18 char val_tab(const char*plateau, int x , int y) {
19     return plateau[NB_COLS*x + y];
20 }
21
22
23
```

```c
/* Cette fonction affiche le plateau toujours au meme
endroit sur l'ecran, vous n'aurez pas de question sur
cette fonction */
void affiche_plateau(const char* plateau) {
    int i,j;
    for(i=0;i < NB_LIGNES; i++)
        for(j=0;j < NB_COLS; j++)
            printf("\033[%d;%dH%d", i+1, j+1, val_tab(plateau, i, j));
}

int compte(char* plateau, int x, int y) {
    int i,j,cpt=0;
    for(i=-1; i < 2 ;i++)
        for(j=-1; j < 2; j++)
            if( !((i == 0) && (j == 0)) && (x+i >= 0) && (y+j >= 0) && (
                x+i < NB_LIGNES) && (y+i < NB_COLS))
                cpt += val_tab(plateau, x+i, y+j);
    return cpt;
}

void cellule(char* plateau, int x, int y) {

    char *cell = ( plateau + NB_COLS*x + y );

    // % -> modulo (reste de la division)
    *cell = getpid() % 2;

    while(1) {
        int cpt = compte(plateau, x, y);
        if(*cell > 0) {
            if( (cpt < 2) || (cpt > 3) )
                *cell = 0;
        }
        else if(cpt == 3)
            *cell = 1;

        sched_yield();
    }

    exit(EXIT_SUCCESS);
}

int main() {
    char *plateau;
    int fd_mem;
    size_t mem_size = NB_LIGNES * NB_COLS * sizeof(char);

    fd_mem = shm_open(MEM_NAME, O_RDWR | O_CREAT | O_EXCL, 0600);
    if (fd_mem == -1)
        exit_err("main, shm_open");
    shm_unlink(MEM_NAME);

    if(ftruncate(fd_mem, mem_size) == -1)
        exit_err("main, ftruncate");

```

```
78      plateau = (char*) mmap(NULL, mem_size, PROT_READ | PROT_WRITE,
            MAP_SHARED, fd_mem, 0);
79      if (plateau == MAP_FAILED)
80          exit_err("main, mmap");
81      close(fd_mem);
82
83      //Le code commente ci-dessous est en lien avec une des question ci-
            dessus
84  /*
85      struct sched_param sp;
86      sp.sched_priority = 1;
87      sched_setscheduler(getpid(), SCHED_FIFO, &sp);
88  */
89      int i, j;
90      for(i=0;i < NB_LIGNES; i++) {
91          for(j=0;j < NB_COLS; j++) {
92              pid_t res = fork();
93              if(res == -1)
94                  exit_err("main, fork");
95              else if(res == 0)
96                  cellule(plateau, i, j);
97          }
98      }
99
100     while(1)
101         affiche_plateau(plateau);
102
103     if(munmap(plateau, mem_size) == -1)
104         exit_err("main, munmap");
105
106     return 0;
107 }
```

Makefile

```
1  all: life
2
3  life: life.c
4          gcc -Wall -g life.c -o life -l rt
```