Le programme ci-dessous est implémenté dans une machine à café dont les boutons envoies les signaux CAFE, CREME et ANUL au programme principal (c.f. code).

1. Combien de processus ce programme génère-t-il ? Décrire le rôle et les actions de chaque processus.

2. Que ce passe-t-il si le bouton CAFE est pressé plusieurs fois de suite ?

3. Que ce passe-t-il si l'on appuye sur SIG_CREME sans qu'un café ne soit lancé ? Que ce passe-t-il si l'on appuye sur SIG_CREME lorsqu'un café est lancé ?

4. Dans quel cas le programme affiche-t-il l'erreur "Erreure survenue, cafe interrompu" ?

5. Pourquoi utiliser la fonction "write" au lieu de "printf" dans les handlers de signaux ?

cafe.c

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <errno.h>

#define SIG_CAFE SIGUSR1
#define SIG_CREME SIGUSR2
#define SIG_ANUL SIGINT

const char * const cremeStr = "Ajout de creme sur le cafe\n";
const char * const anulStr = "Cafe interrompu par l'utilisateur\n";
const char * const errStr = "Erreure survenue, cafe interrompu\n";

pid_t pidService = 0;

void exit_err(const char* str)
{
    perror(str);
    exit(EXIT_FAILURE);
}

void set_sigaction_handler(int signum, void (*sig_handler)(int))
//Le parametre sig_handler est une fonction "handler" ou SIG_IGN
{
    struct sigaction action;

    if(sigemptyset(&action.sa_mask) == -1)
        exit_err("set_sigaction_handler, sigemptyset");

    action.sa_flags = 0;
    action.sa_handler = sig_handler;
    if(sigaction(signum, &action, NULL) == -1)
```

```
37          exit_err("set_sigaction_handler, sigaction");
38  }
39
40  void service_handler(int sig)
41  {
42      switch(sig)
43      {
44          //SIG_CREME recu
45          case SIG_CREME:
46              write(STDIN_FILENO, cremeStr, strlen(cremeStr));
47              break;
48
49          //SIG_ANUL recu
50          case SIG_ANUL:
51              write(STDIN_FILENO, anulStr, strlen(anulStr));
52              exit(EXIT_SUCCESS);
53              break;
54
55          default:
56              fprintf(stderr, "Signal %d non attendu\n", sig);
57              exit(EXIT_FAILURE);
58      }
59  }
60
61  void service()
62  {
63      set_sigaction_handler(SIG_CREME, service_handler);
64      set_sigaction_handler(SIG_ANUL, service_handler);
65
66      int err;
67      sigset_t set, oldset;
68      err = sigemptyset(&set);
69      err |= sigaddset(&set, SIG_CREME);
70      if(err != 0)
71          exit_err("service, construction d'ensemble de signaux");
72      if(sigprocmask(SIG_BLOCK, &set, &oldset) == -1)
73          exit_err("service, sigprocmask 1");
74
75      printf("Je commence a preparer un cafe\n");
76      sleep(5);
77
78      if(sigprocmask(SIG_SETMASK, &oldset, NULL) == -1)
79          exit_err("service, sigprocmask 2");
80
81      printf("Cafe termine\n");
82      exit(EXIT_SUCCESS);
83  }
84
85  void btn_handler(int sig)
86  {
87      switch(sig)
88      {
89          //SIG_CAFE recu
90          case SIG_CAFE:
91              pidService = fork();
```

```
92              if(pidService == -1)
93                  exit_err("btn_handler, fork");
94              else if (pidService > 0)
95              {
96                  set_sigaction_handler(SIG_CREME, btn_handler);
97                  set_sigaction_handler(SIG_ANUL, btn_handler);
98
99                  int status = 0;
100                 while( (status = waitpid(pidService, NULL, 0)) == -1)
101                     if(errno != EINTR)
102                         exit_err("btn_handler, waitpid");
103                 if(WIFEXITED(status))
104                     if(WEXITSTATUS(status) != 0)
105                         write(STDIN_FILENO, errStr, strlen(errStr));
106             }
107             else
108                 service();
109             break;
110
111         //SIG_CREME recu
112         case SIG_CREME:
113             if(kill(pidService, SIG_CREME) == 1)
114                 exit_err("btn_handler, kill");
115             break;
116
117         //SIG_ANUL recu
118         case SIG_ANUL:
119             if(kill(pidService, SIG_ANUL) == 1)
120                 exit_err("btn_handler, kill");
121             break;
122
123         default:
124             fprintf(stderr, "Le signal %d non attendu\n", sig);
125             exit(EXIT_FAILURE);
126     }
127 }
128
129 int main(void)
130 {
131     set_sigaction_handler(SIG_CAFE, btn_handler);
132     set_sigaction_handler(SIG_CREME, SIG_IGN);
133     set_sigaction_handler(SIG_ANUL, SIG_IGN);
134
135     while(1)
136         pause();
137
138     return 0;
139 }
```

Makefile

```
1  all: cafe
2
3  cafe: cafe.c
4          gcc cafe.c -g -Wall -o cafe
```