1. Que fait le programme suivant ?
2. D'après la structure du programme et ces appels aux fonctions système, comment appel-t-on ce type de programme ?
3. A quoi sert l'entrée "install" du makefile ?
4. Décrivez le code et son fonctionement en détail.
5. Quel sera l'utilisateur, le groupe et les droits d'accès du fichier de log une fois celui-ci créé.
6. Quelle est la commande à taper depuis un terminal pour activer le handler du programme ?

somename.c

```c
#include <signal.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/sysinfo.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

#define PATH_LOG "/var/log"
#define NAME_LOG "myownlog.log"

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

/* ============ getNbProc ============
Cette fonction retourne le nombre de processus du systeme,
vous ne serez pas interroge sur cette fonction
sortie: le nombre de processus actifs du systeme
note: la fonction termine le processus d'appel en cas d'erreur
   ============ getNbProc ============*/
int getNbProc(void)
{
    struct sysinfo si;

    if(sysinfo(&si) != 0)
        exit_err("Connot get system info");

    return si.procs;
}


void handler(int sig)
{
```

```
39      static FILE* logFile;
40      int nbProc;
41
42      nbProc = getNbProc();
43      if((logFile = fopen(NAME_LOG,"a")) == NULL)
44          exit_err("Cannot open log file");
45
46      fprintf(logFile, "User call: %d\n", nbProc);
47
48      fclose(logFile);
49  }
50
51  int main(void)
52  {
53      pid_t pid;
54      int fd;
55      static FILE* logFile;
56
57      pid = fork();
58      if(pid < 0)
59          exit_err("Cannot fork");
60      else if(pid != 0)
61          return 0;
62
63      if(setsid() == -1)
64          exit_err("Cannot setsid");
65
66      if(chroot(PATH_LOG) != 0)
67          exit_err("Cannot change root");
68      if(chdir("/") != 0)
69          exit_err("Cannot change path");
70
71      umask(S_IXUSR | S_IXGRP | S_IROTH | S_IWOTH | S_IXOTH);
72
73      if((logFile = fopen(NAME_LOG,"w")) == NULL)
74          exit_err("Cannot open log file");
75      fclose(logFile);
76
77      chown(NAME_LOG, getuid(), getgid());
78
79      setuid(getuid());
80
81      if((fd = open(NAME_LOG, O_WRONLY | O_APPEND)) == -1)
82          exit_err("Cannot open log file for input / output redirection");
83      if( (dup2(fd, STDERR_FILENO) == -1) || (dup2(fd, STDOUT_FILENO) ==
            -1) )
84          exit_err("Cannot exchange file descriptors");
85      close(fd);
86
87      struct sigaction sa;
88      sa.sa_handler = handler;
89      sigfillset(&sa.sa_mask);
90      sa.sa_flags = 0;
91      sigaction(SIGUSR1, &sa, NULL);
92
```

```
93      while(1)
94          pause();
95
96      return 0;
97  }
```

makefile

```
1   all: somename.c
2           gcc somename.c -Wall -o somename
3
4   install:
5           @echo "L'installation necessite les droits root"
6           chown root ./somename
7           chmod u+s ./somename
8           @echo "Installation reussie"
```