

```

00001 #include <errno.h>
00002 #include <fcntl.h>
00003 #include <signal.h>
00004 #include <stdarg.h>
00005 #include <stdint.h>
00006 #include <stdio.h>
00007 #include <stdlib.h>
00008 #include <stdnoreturn.h>
00009 #include <string.h>
00010 #include <time.h>
00011 #include <unistd.h>
00012
00013 #define CHEMIN_MAX 512
00014 #define MAX_INT_LEN 20 // Le plus grand entier possède 20 caractères
00015
00016 // Vérifie les appels aux fonctions qui renvoient -1 en cas d'erreur
00017 #define CHK_PS(v) \
00018     do { \
00019         if ((v) == -1) \
00020             raler(1, #v); \
00021     } while (0)
00022
00023 // Vérifie les appels à la fonction 'signal'
00024 #define CHK_SG(v) \
00025     do { \
00026         if ((v) == SIG_ERR) \
00027             raler(1, #v); \
00028     } while (0)
00029
00030 // Rale
00031 noreturn void raler(int syserr, const char *fmt, ...) {
00032     va_list ap;
00033     va_start(ap, fmt);
00034     vfprintf(stderr, fmt, ap);
00035     fprintf(stderr, "\n");
00036     va_end(ap);
00037     if (syserr)
00038         perror("");
00039     exit(1);
00040 }
00041
00042 // Valeurs atomiques pour gérer les signaux
00043 volatile sig_atomic_t sig_alrm, sig_int, sig_term;
00044
00045 void fct_alrm(int sig) {
00046     (void)sig;
00047     sig_alrm = 1;
00048 }
00049
00050 void fct_int(int sig) {
00051     (void)sig;
00052     sig_int = 1;
00053 }
00054
00055 void fct_term(int sig) {
00056     (void)sig;
00057     sig_term = 1;
00058 }
00059
00060 int main(int argc, char *argv[]) {
00061     if (argc != 4)
00062         raler(0, "Nombre d'arguments incorrect");
00063
00064     if (strlen(argv[1]) > CHEMIN_MAX)
00065         raler(0, "Chemin trop long");
00066

```

Ah ? Vous en êtes sûr ?

=&gt; non portable

Faites un message utile : rappelez la syntaxe

```

00067     int tinc = atoi(argv[2]);
00068     int tstamp = atoi(argv[3]);
00069     if (tinc < 0 || tstamp <= 0)
00070         raler(0, "Temps invalide");
00071
00072     alarm(tstamp);
00073
00074     CHK_SG(signal(SIGALRM, fct_alrm));
00075     CHK_SG(signal(SIGINT, fct_int));
00076     CHK_SG(signal(SIGTERM, fct_term));
00077
00078     int fd;
00079     CHK_PS(fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0666));
00080
00081     time_t time_epoch;
00082     char *time_formatted;
00083
00084     char compteur_str[MAX_INT_LEN + 1];
00085     int write_size;
00086
00087     sig_alrm = sig_int = sig_term = 0;
00088     for (uintmax_t compteur = 0;; compteur++) {
00089         if (usleep(tinc * 1000) == -1) {
00090             if (EINTR)
00091                 fprintf(stderr, "\tSommeil interrompu\n");
00092             else if (EINVAL)
00093                 raler(1, "'tinc' trop grand pour ce système");
00094         }
00095
00096         if (sig_alrm) {
00097             if ((time_epoch = time(NULL)) == (time_t)-1)
00098                 raler(1, "Erreur de la fonction 'time'");
00099             time_formatted = ctime(&time_epoch);
00100             CHK_PS(write(fd, time_formatted, 25));
00101             sig_alrm = 0;
00102             alarm(tstamp);
00103         }
00104
00105         if (sig_int) {
00106             if ((write_size = snprintf(compteur_str, MAX_INT_LEN + 1, "%ju%c",
00107                                     compteur, '\n')) < 0)
00108                 raler(0, "Erreur de conversion du compteur en chaîne");
00109             CHK_PS(write(fd, compteur_str, write_size));
00110             sig_int = 0;
00111         }
00112
00113         if (sig_term) {
00114             CHK_PS(write(fd, "fin\n", 4));
00115             CHK_PS(close(fd));
00116             return 0; // Indique que le programme s'est arrêté comme on voulait
00117         }
00118     }
00119 }

```

L'énoncé autorise les fcts de bib (ex : fopen/fprintf) : ce serait

plus simple

Le nom "epoch" prête à confusion

dans ce cas là, il ne faudrait rien faire (rien afficher)