```c
#include <signal.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdnoreturn.h>
#include <string.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>

#define CHEMIN_MAX 512
#define TAILLE_BLOC 4096

/* Vérifie les appels aux fonctions qui renvoient -1 en cas d'erreur */
#define CHK_PS(v)          \
    do {                   \
        if ((v) == -1)     \
            raler(1, #v); \
    } while (0)

/* Pour facilement râler auprès du parent */
#define CHK_CD(v)                    \
    do {                             \
        if ((v) == -1)               \
            raler_au_parent(1, #v); \
    } while (0)

/* Vérifie les appels à la fonction 'signal' */
#define CHK_SG(v)            \
    do {                     \
        if ((v) == SIG_ERR) \
            raler(1, #v);    \
    } while (0)

/* Vérifie les appels à la fonction 'fflush' */
#define CHK_FL(v)                    \
    do {                             \
        if ((v) == EOF)              \
            raler_au_parent(1, #v); \
    } while (0)

/* Râle */
noreturn void raler(int syserr, const char *fmt, ...) {
    va_list ap;
    va_start(ap, fmt);
    vfprintf(stderr, fmt, ap);
    fprintf(stderr, "\n");
    va_end(ap);
    if (syserr)
        perror("");
    exit(1);
}

/* Râler spéciale du fils auprès du père */
void raler_au_parent(int syserr, char *message) {
    fprintf(stderr, "Process PID %d\n%s\n", getpid(), message);
    if (syserr)
        perror("");
    fflush(stderr);
    kill(getppid(), SIGUSR2);  // Réveille le parent
    kill(getppid(), SIGHUP);   // Indique au parent de tuer tous les autres
    exit(1);
}

/* Explication de l'usage du programme */
void usage(char *prog) {
```

le test6 ne passe pas, dommage !

sinon, je trouve que vous faites toujours très compliqué...


```c
    raler(0, "usage: %s n", prog);
}

// Permission au fils de lire un octet dans le tube
volatile sig_atomic_t allow_read;
// Indication au père de continuer a écrire dans le tube
volatile sig_atomic_t cont_write;
// Au cas ou tout ne se passe pas bien dans un fils
volatile sig_atomic_t child_has_issues;
// Le fils reçoit un signal extérieur de terminaison
volatile sig_atomic_t child_term;

/* Fonction pour gérer ces signaux */
void fct(int sig_num) {
    switch (sig_num) {
        case SIGUSR1:
            allow_read = 1;
            break;
        case SIGUSR2:
            cont_write = 1;
            break;
        case SIGHUP:
            child_has_issues = 1;
            break;
        case SIGTERM:
            child_term = 1;
            break;
        default:
            break;
    };
}

/* Pour facilement initialiser les 'sigaction' */
void init_sigaction(struct sigaction *s, int sig, void (*f)(int)) {
    s->sa_handler = f;
    s->sa_flags = 0;
    CHK_PS(sigemptyset(&s->sa_mask));
    CHK_PS(sigaddset(&s->sa_mask, sig));
    CHK_PS(sigaction(sig, s, NULL));
}

/* Pour terminer proprement tous les fils */
void kill_all_childs(pid_t *pid_fils, int n_proc) {
    for (int k = 0; k < n_proc; k++)
        CHK_PS(kill(pid_fils[k], SIGHUP));

    for (int k = 0; k < n_proc; k++)
        wait(NULL);
}

/* Lecture d'un octet dans le tube par le fils */
void lecture_fils(int id_fils, int tube_0) {
    unsigned char c;
    int nread;

    sigset_t masque, vieux, vide;
    CHK_CD(sigemptyset(&vide));
    CHK_CD(sigemptyset(&masque));
    CHK_CD(sigaddset(&masque, SIGUSR1));

    allow_read = 0;
    for (;;) {
        CHK_CD(sigprocmask(SIG_BLOCK, &masque, &vieux)); /* Section critique */
        while (!allow_read)
            sigsuspend(&vide);  // Attend l'ordre de lecture
        allow_read = 0;
```

inutile : le signal courant est toujours automatiquement masqué quand on est dans la fonction

en cas de forte charge (test6), le signal SIGUSR1 peut arriver entre le fork et cette ligne. Dans ce cas, allow_read passe à 1, puis repasse ici à 0 => le signal SIGUSR1 est donc perdu => le système se bloque.

```
00133         CHK_CD(sigprocmask(SIG_SETMASK, &vieux, NULL));
00134
00135         CHK_CD(nread = read(tube_0, &c, 1));  // Lit
00136         printf("%d: %c\n", id_fils, c);
00137         CHK_FL(fflush(stdout));
00138
00139         CHK_CD(kill(getppid(), SIGUSR2));  // Notifie le père
00140
00141         if (child_term)  // Le fils reçoit SIGTERM
00142             raler_au_parent(0, "Terminaison inattendue");
00143     }
00144 }
00145
00146 /* Main */
00147 int main(int argc, char *argv[]) {
00148     int n;
00149
00150     if (argc != 2)
00151         usage(argv[0]);
00152
00153     if ((n = atoi(argv[1])) <= 0)
00154         raler(0, "n > 0");
00155
00156     // Le signal SIGUSR1 indique au fils que c'est son tour de lire
00157     struct sigaction usr1;
00158     init_sigaction(&usr1, SIGUSR1, fct);
00159
00160     // Le signal SIGUSR2 indique au père qu'il peut continuer a écrire
00161     struct sigaction usr2;
00162     init_sigaction(&usr2, SIGUSR2, fct);
00163
00164     // Si le fils se termine par SIGTERM (ou autre erreur), il notifie le père
00165     struct sigaction term;
00166     init_sigaction(&term, SIGTERM, fct);
00167
00168     int tube[2];
00169     CHK_PS(pipe(tube));
00170
00171     pid_t pid;
00172     pid_t pid_fils[n];  // Les identités des fils
00173     for (int k = 0; k < n; k++) {
00174         switch (pid = fork()) {
00175             case -1:
00176                 raler(1, "Fork");
00177                 break;
00178             case 0:
00179                 CHK_CD(close(tube[1]));
00180                 lecture_fils(k, tube[0]);
00181                 CHK_CD(close(tube[0]));
00182                 exit(0);
00183                 break;
00184             default:
00185                 pid_fils[k] = pid;
00186                 break;
00187         }
00188     }
00189     CHK_PS(close(tube[0]));
00190
00191     // Le signal SIGHUP indique a tous les fils de s'arrêter (le père aussi)
00192     struct sigaction hup;
00193     init_sigaction(&hup, SIGHUP, fct);
00194
00195     init_sigaction(&term, SIGTERM, SIG_DFL);  // Retour a la normale
00196
00197     sigset_t masque, vieux, vide;
00198     CHK_PS(sigemptyset(&vide));
```



```
00199     CHK_PS(sigemptyset(&masque));
00200     CHK_PS(sigaddset(&masque, SIGUSR2));
00201
00202     unsigned char buffer[TAILLE_BLOC];
00203     ssize_t n_read;
00204     off_t octet_num = 0;  // Indice global de l'octet en cours
00205     int id_fils;
00206
00207     cont_write = 0;
00208     child_has_issues = 0;
00209
00210     while ((n_read = read(0, buffer, TAILLE_BLOC)) > 0) {
00211         for (int i = 0; i < n_read; i++) {
00212             CHK_PS(write(tube[1], &buffer[i], 1));  // Écrire l'octet
00213
00214             id_fils = octet_num % n;
00215             CHK_PS(kill(pid_fils[id_fils], SIGUSR1));  // Notifie le fils
00216             octet_num++;
00217
00218             CHK_PS(sigprocmask(SIG_BLOCK, &masque, &vieux)); /* Critique */
00219             while (!cont_write)
00220                 sigsuspend(&vide);  // Attend la confirmation de lecture
00221             cont_write = 0;
00222             CHK_PS(sigprocmask(SIG_SETMASK, &vieux, NULL));
00223
00224             if (child_has_issues) {  // Si un problème est survenu
00225                 CHK_PS(close(tube[1]));
00226                 kill_all_childs(pid_fils, n);
00227                 exit(1);
00228             }
00229         }
00230     }
00231
00232     CHK_PS(close(tube[1]));
00233     kill_all_childs(pid_fils, n);
00234     return 0;  // Indique que le fils s'est normalement terminé par un SIGHUP
00235 }
```