descripteursFichiers.c

#include <stdio.h>  
#include <unistd.h>  
#include <stdlib.h>  
#include<fcntl.h>  
  
void main(int argc, char \*\*argv) {  
 if(argc < 4){  
 printf("Usage : ./redir IN OUT commande [argument...]\n");  
 exit(EXIT\_FAILURE);  
 }  
 int fd\_in, fd\_out, retour\_dup\_0;  
 fd\_in = open(argv[1], O\_RDONLY);  
 if(fd\_in == -1){  
 perror(argv[1]);  
 }  
 fd\_out = creat(argv[2], 0666);  
 if(fd\_out == -1){  
 perror(argv[2]);  
 }  
 if(dup2(fd\_in, 0) == -1 || dup2(fd\_out, 1) == -1 || dup2(fd\_out, 2) == -1){  
 perror("echec dup2");  
 }  
 close(fd\_in);  
 close(fd\_out);  
 execvp (argv[3], argv+3);  
 perror("La commande ne peut pas être exécutée");  
 exit(EXIT\_FAILURE);  
}

mini-tail.c

#include <stdio.h>  
#include <stdlib.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <unistd.h>  
#include <fcntl.h>  
  
int main(int argc, char\* argv[]) {  
 if (argc != 2) {  
 return 1;  
 }  
 int file\_descriptor;  
 off\_t old\_size;  
 struct stat file\_stat;  
 file\_descriptor = open(argv[1], O\_RDONLY);  
 if (file\_descriptor == -1) {  
 perror("Erreur lors du open");  
 return 1;  
 }  
 if (fstat(file\_descriptor, &file\_stat) == -1) {  
 perror("Erreur lors du stat");  
 return 1;  
 }  
 old\_size = file\_stat.st\_size;  
  
 char\* buffer = NULL;  
 while (1) {  
 sleep(1);  
 if (fstat(file\_descriptor, &file\_stat) == -1) {  
 perror("Erreur lors du stat");  
 return 1;  
 }  
 if (file\_stat.st\_size > old\_size) {  
 int size = file\_stat.st\_size - old\_size;  
 buffer = malloc(size + 1);  
 if (buffer == NULL) {  
 perror("Erreur lors du malloc");  
 return 1;  
 }  
 buffer[size] = '\0';  
 lseek(file\_descriptor, old\_size, SEEK\_SET);  
 if (read(file\_descriptor, buffer, size) == -1){  
 perror("Erreur lors du read");  
 return 1;  
 }  
 printf("%s\n", buffer);  
 } else if (file\_stat.st\_size < old\_size) {  
 buffer = malloc(file\_stat.st\_size + 1);  
 if (buffer == NULL) {  
 perror("Erreur lors du malloc");  
 return 1;  
 }  
 buffer[file\_stat.st\_size] = '\0';  
 if (read(file\_descriptor, buffer, file\_stat.st\_size) == -1){  
 perror("Erreur lors du read");  
 return 1;  
 }  
 printf("--- Le fichier a été tronqué! ---\n%s\n", buffer);  
 }  
 old\_size = file\_stat.st\_size;  
 free(buffer);  
 buffer = NULL;  
 lseek(file\_descriptor, 0, SEEK\_SET);  
 }  
}

mini-tail-PlusieursFichiers.c

#include <stdio.h>  
#include <stdlib.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <unistd.h>  
#include <fcntl.h>  
int main(int argc, char\* argv[]) {  
 if (argc < 2) {  
 return 1;  
 }  
 int fds[argc - 1];  
 off\_t prev\_sizes[argc - 1];  
 struct stat statbuf;  
 int st;  
 for (int i = 1; i < argc; i++) {  
 fds[i-1] = open(argv[i], O\_RDONLY);  
 if (fds[i-1] == -1) {  
 perror("Erreur lors de l'ouverture");  
 return 1;  
 }  
 st = fstat(fds[i-1], &statbuf);  
 if (st == -1) {  
 perror("Erreur lors du stat");  
 return 1;  
 }  
 prev\_sizes[i-1] = statbuf.st\_size;  
 }  
 char\* buf = NULL;  
 while (1) {  
 sleep(1);  
 for (int i = 0; i < argc-1; i++) {  
 st = fstat(fds[i], &statbuf);  
 if (st == -1) {  
 perror("Erreur lors du stat");  
 return 1;  
 }  
 if (statbuf.st\_size > prev\_sizes[i]) {  
 int size = statbuf.st\_size - prev\_sizes[i];  
 buf = malloc(size + 1);  
 if (buf == NULL) {  
 perror("Erreur lors de l'allocation");  
 return 1;  
 }  
 buf[size] = '\0';  
 lseek(fds[i], prev\_sizes[i], SEEK\_SET);  
 int rd = read(fds[i], buf, size);  
 printf("%s:\n%s\n", argv[i+1], buf);  
 } else if (statbuf.st\_size < prev\_sizes[i]) {  
 buf = malloc(statbuf.st\_size + 1);  
 if (buf == NULL) {  
 perror("Erreur lors de l'allocation");  
 return 1;  
 }  
 buf[statbuf.st\_size] = '\0';  
 int rd = read(fds[i], buf, statbuf.st\_size);  
 printf("%s:---Fichier a été tronqué---\n%s\n", argv[i+1], buf);  
 }  
 prev\_sizes[i] = statbuf.st\_size;  
 free(buf);  
 buf = NULL;  
 lseek(fds[i], 0, SEEK\_SET);  
 }  
 }  
}

mini-tail-INOTIFY.c

#include <stdio.h>  
#include <stdlib.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <sys/inotify.h>  
#include <unistd.h>  
#include <fcntl.h>  
  
struct file\_watch {  
 char\* path;  
 int fd;  
 int wd;  
 off\_t size;  
};  
  
void handle\_mod(int inotif, struct file\_watch\* watched, int num\_wds) {  
 struct inotify\_event\* event;  
 char buf[sizeof(struct inotify\_event)];  
 int len = read(inotif, buf, sizeof(struct inotify\_event));  
 struct stat statbuf;  
 char\* content = NULL;  
 char\* msg = NULL;  
 while (len > 0) {  
 event = (struct inotify\_event \*)buf;  
 for (int i = 0; i < num\_wds; i++) {  
 if (watched[i].wd == event->wd) {  
 fstat(watched[i].fd, &statbuf);  
*// Si la taille a augmentée, on affiche le nouveau contenu*  
if (statbuf.st\_size > watched[i].size) {  
 msg = "";  
 int size = statbuf.st\_size - watched[i].size;  
 content = malloc(size + 1);  
 content[size] = '\0';  
 read(watched[i].fd, content, size);  
*// Si la taille a baissée, on affiche tout le contenu*  
} else if (statbuf.st\_size < watched[i].size) {  
 msg = " file truncated";  
 lseek(watched[i].fd, 0, SEEK\_SET);  
 content = malloc(statbuf.st\_size + 1);  
 content[statbuf.st\_size] = '\0';  
 read(watched[i].fd, content, statbuf.st\_size);  
 } else {  
 continue;  
 }  
 printf("File: %s%s\n%s\n", watched[i].path, msg, content);  
 free(content);  
 content = NULL;  
 watched[i].size = statbuf.st\_size;  
 }  
 }  
  
 len = read(inotif, buf, sizeof(struct inotify\_event));  
 }  
}  
  
int main(int argc, char\* argv[]) {  
 if (argc < 2) {  
 return 1;  
 }  
  
 int inotif = inotify\_init();  
 if (inotif == -1) {  
 perror(NULL);  
 return -1;  
 }  
 struct file\_watch\* watched = malloc(sizeof(struct file\_watch) \* argc-1);  
 struct stat statbuf;  
 char\* content = NULL;  
 *// pour chaque fichier, on ajoute un watcher inotify, on remplit le struct et on affiche le contenu initial*  
for (int i = 1; i < argc; i++) {  
 watched[i-1].path = argv[i];  
 watched[i-1].wd = inotify\_add\_watch(inotif, argv[i], IN\_MODIFY);  
 if (watched[i-1].wd == -1) {  
 perror("Add watch");  
 free(watched);  
 return -1;  
 }  
  
 watched[i-1].fd = open(watched[i-1].path, O\_RDONLY);  
 if (watched[i-1].fd == -1) {  
 perror("Open");  
 free(watched);  
 return -1;  
 }  
 int st = fstat(watched[i-1].fd, &statbuf);  
 if (st == -1) {  
 perror("Stat");  
 free(watched);  
 return -1;  
 }  
 watched[i-1].size = statbuf.st\_size;  
 content = realloc(content, watched[i-1].size + 1);  
 if (content == NULL) {  
 perror("Realloc");  
 free(watched);  
 return -1;  
 }  
 content[watched[i-1].size] = '\0';  
 int rd = read(watched[i-1].fd, content, watched[i-1].size);  
 if (rd == -1) {  
 perror("Read");  
 free(watched);  
 return -1;  
 }  
 printf("File: %s\n%s\n", watched[i-1].path, content);  
 }  
 free(content);  
  
 while (1) {  
 handle\_mod(inotif, watched, argc-1);  
 }  
 return 0;  
}