eviterSigInt-DurantTraitement.c  
#include<stdlib.h>  
#include<stdio.h>  
#include<unistd.h>  
#include<signal.h>  
  
void gere(int sig) {  
 for(int i=0; i<5; i++) {  
 printf("Itération %d pour la gestion du signal %d\n", i+1, sig);  
 if(i==2)  
 sleep(5);*//afin d'avoir le temps de placer un ctrl+c*  
}  
 exit(1);  
}  
  
int main(void) {  
 struct sigaction action;  
 sigemptyset(&action.sa\_mask);  
 sigaddset(&action.sa\_mask,SIGINT);*//pour ignorer SIGINT durant l'execution de la gérante*  
action.sa\_flags = 0;  
 action.sa\_handler = gere;  
 sigaction(SIGUSR1, &action, NULL);  
 pause();  
 return 0 ;  
}

shellFifo.c

#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <string.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <sys/wait.h>  
  
int main(int argc, char \* argv[]) {  
 int i, indice2;  
 char \* nom\_tube = "fifo";  
 for (i = 1; i < argc; i++) {  
 if (strcmp(argv[i], "{}") == 0) {  
 argv[i] = nom\_tube;  
 }  
 if (strcmp(argv[i], "<|") == 0) {  
 argv[i] = NULL;  
 indice2 = i + 1;  
 }  
 }  
  
 int descripteur;  
 pid\_t commande1, commande2;  
 if (mkfifo(nom\_tube, 0666) == -1) {  
 perror("mkfifo");  
 return 1;  
 }  
  
 commande1 = fork();  
 if (commande1 == 0) {  
 descripteur = open(nom\_tube, O\_WRONLY);  
 if (descripteur == -1) {  
 perror("open fichier fifo");  
 return 1;  
 }  
 dup2(descripteur, 1);  
 close(descripteur);  
 execvp(argv[indice2], argv + indice2);  
 perror(argv[indice2]);  
 return 1;  
 }  
  
 commande2 = fork();  
 if (commande2 == 0) {  
 execvp(argv[1], argv + 1);  
 perror(argv[1]);  
 return 1;  
 }  
  
 waitpid(commande1, NULL, 0);  
 waitpid(commande2, NULL, 0);  
 unlink(nom\_tube);  
 return 0;  
}

*gestionDirecte.c*  
#include<stdlib.h>  
#include<stdio.h>  
#include<unistd.h>  
#include<signal.h>  
void gere(int sig) {  
 printf("Réception du signal SIGINT\n");  
 exit(1);  
}  
int main(void) {  
 struct sigaction action;  
 sigemptyset(&action.sa\_mask);  
 action.sa\_flags = 0;  
 action.sa\_handler = gere;  
 sigaction(SIGINT, &action, NULL);  
 pause();  
 return 0 ;  
}  
*//En ignorant l’occurence du signal:*  
*//#include<signal.h>*  
*//#include<unistd.h>*  
*//*  
*//int main(void) {*  
*// struct sigaction action;*  
*// sigemptyset(&action.sa\_mask);*  
*// action.sa\_flags = 0;*  
*// action.sa\_handler = SIG\_IGN;*  
*// sigaction(SIGINT, &action, NULL);*  
*// pause();*  
*//*  
*// return 0 ;*

shellFifo-2.c

#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <string.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <sys/wait.h>  
int main(int argc, char \* argv[]) {  
 int i, indice1, indice2;  
 char chemin\_pseudo\_fichier[20];  
 strcpy(chemin\_pseudo\_fichier, "/proc/self/fd/");  
  
 for (i = 1; i < argc; i++) {  
 if (strcmp(argv[i], "{}") == 0) {  
 argv[i] = chemin\_pseudo\_fichier;  
 indice1 = i;  
 }  
 if (strcmp(argv[i], "<|") == 0) {  
 argv[i] = NULL;  
 indice2 = i + 1;  
 }  
 }  
  
 int p[2];  
 pid\_t commande1, commande2;  
 pipe(p);  
  
 commande1 = fork();  
 if (commande1 == 0) {  
 dup2(p[1], 1);  
 close(p[1]);  
 execvp(argv[indice2], argv + indice2);  
 perror(argv[indice2]);  
 return 1;  
 }  
  
 commande2 = fork();  
 if (commande2 == 0) {  
 sprintf(argv[indice1], "%s%d", argv[indice1], p[0]);  
 close(p[1]);  
 execvp(argv[1], argv + 1);  
 perror(argv[1]);  
 return 1;  
 }  
  
 close(p[0]);  
 close(p[1]);  
 waitpid(commande1, NULL, 0);  
 waitpid(commande2, NULL, 0);  
 return 0;  
}

shellPipe.c

#include <stdio.h>  
#include <unistd.h>  
#include <string.h>  
#include <sys/types.h>  
#include <sys/wait.h>  
int main(int argc, char \*\* argv) {  
 */\*on fera confiance aux arguments entrés par l'utilisateur(pour cette fois-ci)\*/*  
int indice\_debut = 0;  
 for (int i = 1; i != argc; i++) {  
 if (strcmp(argv[i], "+") == 0) {  
 argv[i] = NULL;  
 indice\_debut = i + 1;  
 break;  
 }  
 }  
 int p[2];  
 pipe(p);  
 pid\_t commande1 = fork();  
 if (commande1 == 0) {  
 dup2(p[1], 1);  
 close(p[0]);  
 close(p[1]);  
 execvp(argv[1], argv + 1);  
 perror(argv[1]);  
 return 1;  
 }  
 pid\_t commande2 = fork();  
 if (commande2 == 0) {  
 dup2(p[0], 0);  
 close(p[0]);  
 close(p[1]);  
 execvp(argv[indice\_debut], argv + indice\_debut);  
 perror(argv[indice\_debut]);  
 return 1;  
 }  
 close(p[0]);  
 close(p[1]);  
 int wstatus;  
 waitpid(commande1, NULL, 0);  
 waitpid(commande2, &wstatus, 0);  
 if (WIFEXITED(wstatus)) {  
 return WEXITSTATUS(wstatus);  
 } else {  
 return 1;  
 }  
 return 0;  
}

traiterDeuxFoisSignal-RevenirDefault.c  
#include<stdlib.h>  
#include<stdio.h>  
#include<unistd.h>  
#include<signal.h>  
  
void gere(int sig) {  
 printf("Réception du signal SIGINT\n");  
}  
  
int main(void) {  
 struct sigaction action;  
 sigemptyset(&action.sa\_mask);  
 action.sa\_flags = 0;  
 action.sa\_handler = gere;  
 sigaction(SIGINT, &action, NULL);  
 pause();  
 pause();  
 action.sa\_handler = SIG\_DFL;  
 sigaction(SIGINT, &action, NULL);  
 pause();  
 return 0 ;  
}

traiterUneFois-RevenirTraitementDefault.c

#include<stdlib.h>  
#include<stdio.h>  
#include<unistd.h>  
#include<signal.h>  
  
void gere(int sig) {  
 printf("Réception du signal SIGINT\n");  
}  
  
int main(void) {  
 struct sigaction action;  
 sigemptyset(&action.sa\_mask);  
 action.sa\_flags = 0;  
 action.sa\_handler = gere;  
 sigaction(SIGINT, &action, NULL);  
 pause();  
 action.sa\_handler = SIG\_DFL;  
 sigaction(SIGINT, &action, NULL);  
 pause();  
 return 0 ;  
}

verifierReception-RetarderCrtl-C.c

#include<stdlib.h>  
#include<stdio.h>  
#include<unistd.h>  
#include<signal.h>  
  
void gere(int sig) {  
 for(int i=0; i<5; i++) {  
 printf("Itération %d pour la gestion du signal %d\n", i+1, sig);  
 }  
 exit(1);  
}  
  
int main(void) {  
 struct sigaction action;  
 sigemptyset(&action.sa\_mask);  
 action.sa\_flags = 0;  
 action.sa\_handler = gere;  
 sigaction(SIGUSR1, &action, NULL);  
 pause();  
 return 0 ;  
}