Un petit test de boucle

```
#include <stdio.h>
#include <stdlib.h>
int main() {
	int i; //ma variable compteur

	for(i=0;i<10;i++)
		printf("et voici %i\n",i); //ma boucle

	exit(0); //je sors proprement
}
```

Un petit exec !!!

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

int main() {
	printf("exec de ps avec system\n");
	execlp("ps","ps",NULL);
	exit(0); //inutile, vous verrez pourquoi...
}
```

Par fainéantise

```
#include <stdlib.h>
#include <stdio.h>
int main() {
	printf("exec de ps avec system\n");
	system("ps -ax");
	exit(0); //je sors proprement
}
```

Récupérer les variables d'environnement

```
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char **argv) {

	 char * env1; /* pour accueillir les deux valeurs */
	 char * env2;


	 env1=getenv("ENV1\0");  /* je vais chercher les deux variables
```

```
d'environnement */
        env2=getenv("ENV2\0");

        printf("ENV1 vaut %s\n",env1);
        printf("ENV2 vaut %s\n",env2);


}
```

Test du fork

```
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main() {
        pid_t pid;  //pour recevoir la réponse de fork()
        int n;

        printf("les forks\n");
        pid = fork();  // je fork !!

        printf("la fonction m'a répondu %i\n",pid); //j'affiche la valeur récupérée
        printf("termine\n");
        exit(0);
}
```

Papa et fiston se disputent la sortie standard

```
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main() {
        pid_t pid;
        int n;
        printf("les forks\n");
        pid = fork();
        if (pid==0) {
                for(n=0;n<5;n++) {
                        printf("je suis le fils\n");
                        sleep(3);
                }
        } else {
                for(n=0;n<5;n++) {
                        printf("je suis le pere de %i\n",pid);
                        sleep(1);
                }
        }
        printf("termine\n");
```

```
	exit(0);
}
```

Papa commande, et les fistons obéissent..

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
int main(int argc, char **argv) {

	int attente;
	int i;

	//gestion du paramétrage
	attente=atoi(getenv("ATTENTE\0"));

	//traitement des arguments
	for (i=1;i<argc;i++) { //Attention, on commence à 1 !!!
		char * cde=argv[i]; // on stocke la commande voulue
		pid_t p=fork();
		if(p==0) {
			//je suis le fils
			printf("je suis le fils chargé de faire %s\n",cde);
			printf("j'attends %i cycles\n",attente);
			sleep(attente);
			execlp(cde,cde,NULL);
			exit(0); //inutile, mais on est jamais trop prudent
		} else {
			//on est le père
			printf("%i se charge du boulot\n",p);

		}
	}


}
```

Les fils parlent au père

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
int main(int argc, char **argv) {

	int i;

	for (i=1;i<argc;i++) { //Attention, on commence à 1 !!!
```

```
                char * cde=argv[i]; // on stocke la commande voulue
                pid_t p=fork();
                if(p==0) {
                        //je suis le fils
                        printf("je suis le fils chargé de faire %s\n",cde);
                        sleep(1);
                        execlp(cde,cde,NULL);
                        exit(0); //inutile, mais on est jamais trop prudent
                } else {
                        //on est le père
                        printf("%i se charge du boulot\n",p);
                        //wait(); //on attends le fils
                        pid_t retour;
                        int status;
                        retour=waitpid(-1,&status,0);
                        printf("on a eu le retour de %i\n",retour);
                }
        }


}
```

Aux armes

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>

int nbVies=5; //compteur de vie

void wounded(int s) {
                printf("Aie ! tu m'as fait %i\n",s);
                nbVies--;
}

void rebirth(int s) {
                puts("et hop, ça repart");
                nbVies+=5;
}



int main(int argc, char * argv[]) {

        if(signal(SIGINT,wounded)==SIG_ERR) {
                perror("plantage de la gestion du signal CTRL-C");
                exit(1);
        }
```

```
	if(signal(SIGUSR1,rebirth)==SIG_ERR) {
		perror("plantage de la gestion du signal SIGUSR1");
		exit(1);
	}

	while(nbVies>0) {
		sleep(1);
		printf("il me reste %i vies\n",nbVies);
	}
	exit(0);
}
```

Premier exo des pipes : un fils qui met le clavier en majuscules pour le père

```
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <ctype.h>

int p[2]; //les files descriptors sont en global, pour être visibles partout

void parent() { //ici, le père qui lit dans le pipe pour afficher....
	//quelques messages pour agrémenter le fonctionnement
	char msg_wait[] = "Waiting...\n";
	char msg_read[] = "Just read : " ;
	char msg_done[] = "End of file. Bye!\n";
	int readlen,buflen = 10 ;
	char buf[buflen] ;

	close(p[1]); //fermeture du file descriptor inutile
	while (1) {
	  if ( write(1,msg_wait,strlen(msg_wait)) == -1 ) {
	    perror("write");
	    exit(-1);
	  }
	  readlen = read(p[0],buf,buflen); //On lit vraiment ici
	  if ( readlen == -1 ) {
	    perror("read");
	    exit(-1);
	  }
	  if ( ! readlen ) break;
	  if ( write(1, msg_read, strlen(msg_read)) == -1 ) {
	    perror("write");
	    exit(-1);
	  }
```

```
          if ( write(1, buf, readlen) == -1 ) {
            perror("write");
            exit(-1);
          }
          if ( write(1, "\n", 1) == -1 ) {
            perror("write");
            exit(-1);
          }
        }
        if ( write(1, msg_done, strlen(msg_done) ) == -1 ) {
          perror("write");
          exit(-1);
        }
}

void child() { //ici, le fils qui va lire le clavier, puis envoyer au père)

  close(p[0]);

  char c; //le char qui va accueillir la saisie
  while (1==read(0,&c,1)) { //1 car. par car., notez le pointeur
        c=toupper(c); //conversion
    if ( write(p[1], &c, 1 ) == -1 ) {
      perror("write");
      exit(-1);
    }
  }
}

int main(int argc,char* argv[]) {

  if ( pipe(p) == -1 ) {
    perror("pipe");
    exit(-1);
  }
  switch ( fork() ) { //gestion standard père fils
  case -1:
    perror("fork");
    exit(-1);
  case 0:
    child();
    exit(0);
  default:

        parent();
    exit(0);
  }
  return 0; /* never reached */
}
```