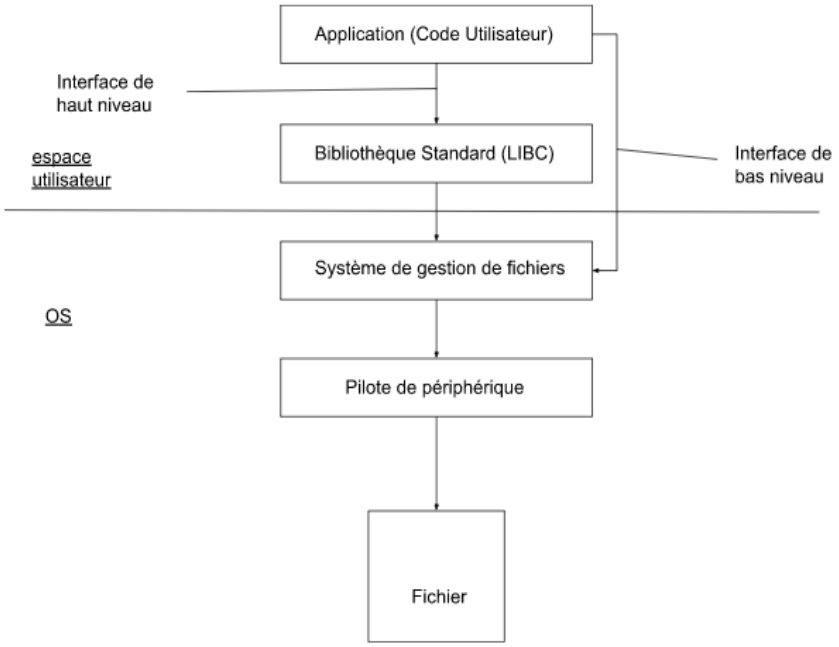
Exercice 1

```
int main(int argc, char *argv[]) {
    int status;
    int save_out = dup(fileno(stdout));
    char * cmd1[argc];
    char * cmd2[argc];
    int cpt = 0;
    for (int i = 0; i < argc; i++){
        if (strcmp(argv[i+1],"--") == 0){ cmd1[i+1] = NULL; break; }
        cmd1[i] = argv[i+1];
        cpt++;
    }
    int j = 0;
    for (int i = cpt+1; i < argc-1; i++){
        if (i+1 ==  argc){ cmd2[i+1] = NULL; }
        cmd2[j] = argv[i+1];
        j++;
    }
    pid_t exec1 = fork();
    if (exec1 == -1) {
        perror("fork1");
        return EXIT_FAILURE;
    }
    if (exec1 == 0) {
        int output1 = open("output1.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
        dup2(output1, fileno(stdout));
        execvp(cmd1[0], cmd1);
        close(output1);
    }
    waitpid(exec1, &status, 0);
    pid_t exec2 = fork();
    if (exec2 == -1) {
        perror("fork2");
        return EXIT_FAILURE;
    }
    if (exec2 == 0) {
        int output2 = open("output2.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
        dup2(output2, fileno(stdout));
        execvp(cmd2[0], cmd2);
        close(output2);
    }
    waitpid(exec2, &status, 0);
    pid_t exec3 = fork();
    if (exec3 == -1) {
        perror("fork3");
        return EXIT_FAILURE;œ
    }
    if (exec3 == 0) {
        dup2(save_out, fileno(stdout));
        execlp("diff", "diff", "-u", "output1.txt", "output2.txt", NULL);
    }
    else {
        waitpid(exec3, &status, 0);
        if (WIFEXITED(status)) {
            if (WEXITSTATUS(status) == 0) { printf("Les fichiers sont identiques\n"); }
            else { printf("Les fichiers sont différents\n"); }
        }
        remove("output1.txt");
        remove("output2.txt");
        return 0;
    }
    return 0; }
```

```
void writeLine(int offset, void *addr, int len) {
    int i;
    char line[17];
    unsigned char *car = addr;
    for (i = 0; i < len; i++) {
        if ((i % 16) == 0) {
            if (i != 0)
                printf(" %s\n", line);
            printf("%08x: ", offset);
            offset += (i % 16 == 0) ? 16 : i % 16;
        }
        printf("%02x", car[i]);
        if ((i % 2) == 1) printf(" ");
        if (isprint(car[i])) line[i % 16] = car[i];
        else line[i % 16] = '.';
        line[(i % 16) + 1] = '\0';
    }
    while ((i % 16) != 0) {
        printf("  ");
        if (i % 2 == 1) putchar(' ');
        i++;
    }
    printf(" %s\n", line);
}

int main(int argc, char *argv[]) {
    if (argc == 2) {
        FILE *FILE = fopen(argv[1], "rb");
            fprintf(stderr, "%s: failed to open file '%s' for reading\n", argv[0], argv[1]);
            exit(EXIT_FAILURE);
        }
        char buffer[SIZE];
        int n = fread(buffer, 1, SIZE, FILE);
        if (n > SIZE) {
            fprintf(stderr, "%s: file '%s' is too large, maximum %d character\n", argv[0], argv[1], SIZE);
            exit(EXIT_FAILURE);
        }
        if (n == 0) {
            fprintf(stderr, "%s: failed to read file '%s'\n", argv[0], argv[1]);
            exit(EXIT_FAILURE);
        }
        int offset = 0;
        writeLine(offset, buffer, n);
        fclose(FILE);
    }
    if (argc == 1) {
        char buffer[SIZE];
        int i = scanf("%[^\n]", buffer);
        if (i == 0) {
            fprintf(stderr, "%s: failed to read input\n", argv[0]);
            exit(EXIT_FAILURE);
        }
        int n = strlen(buffer);
        if (n > SIZE) {
            fprintf(stderr, "%s: text is too large, maximum %d character\n", argv[0], SIZE);
            exit(EXIT_FAILURE);
        }
        int offset = 0;
        writeLine(offset, buffer, n);
    }
    return 0; }
```

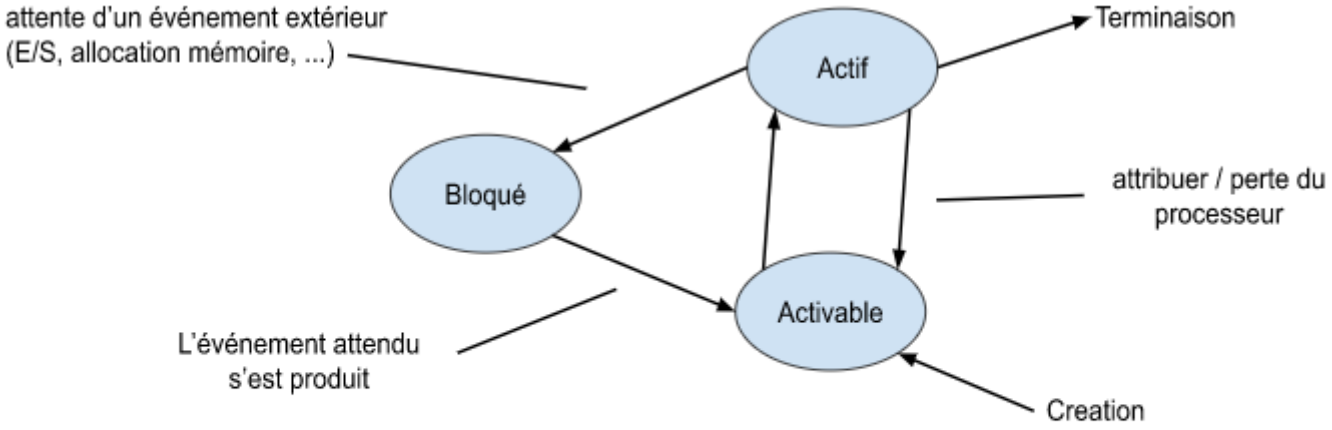


Interface de haut niveau

- Fournie par la bibliothèque standard LIBC
- Indépendante de l'OS (Windows, Linux)
- Utilisation en C : #include <stdio.h>
- Descripteurs de fichiers : type file* et int en haut niveau
- flux standard : stdin, stdout, stderr

Interface de bas niveau

- Fournie par l'OS
- Supporte plus de fonctionnalités (permissions, tubes, …)
- Descripteur de fichiers : type int
- flux standard : 0, 1, 2



- Actif : un processeur a été attribué au processus et il exécute une partie de son code
- Bloqué : un événement extérieur ou une ressource est nécessaire pour poursuivre l'exécution (exemple : entrée / sortie)
- Activable : processus prêt à être exécuté, il dispose de toutes les ressources nécessaires sauf le processeur.

WIFEXITED(status) : Vrai si le processus s'est terminé normalement
=>WEXITSTATUS(status) donne le code de retour

WIFSIGNALED(status) : Vrai si le processus s'est terminé par un signal (donc anormalement)
=>WTERMSIG(status) donne le numéro du signal qui a tué le processus

Les descripteurs (haut-niveau) et des descripteurs (bas-niveau):

stdin	0	ouvert au lancement du programme (associé au terminal en tant qu'entrée)
stdout	1	ouvert au lancement du programme (associé au terminal en tant que sortie)
stderr	2	ouvert au lancement du programme (associé au terminal en tant que sortie)
fd	3	ouverture du fichier
	4	
	5	

1) ouverture du fichier => fd = open(...)
2) fermer la sortie standard => close(1)
3) dupliquer fd => dup(fd)
4) fermer fd => close(fd)

Signaux

→ Un signal est une information élémentaire reçue par un processus pour lui signaler un événement, par exemple:

- fin d'un processus
- erreur arithmétique
- demande de terminaison
- demande de suspension ou reprise

Il est possible de modifier le comportement d'un processus à réception d'un signal (sauf pour SIGKILL et SIGSTOP)
On peut choisir :

- d'ignorer le signal (SIG_IGN)
- retrouver le comportement par défaut (SIG_DFL)
- exécuter une fonction utilisateur

Pour modifier le comportement du processus à réception d'un signal, on peut utiliser soit **signal()** soit **sigaction()**
Avec signal:

```
typedef void(*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

→ signal dont le comportement à changer

handler peut valoir : SIG_IGN
SIG_DFL
[pointeur vers une fct (int) → void]

NB: On évitera d'utiliser signal() avec une fct: préférer sigaction()