

## Systemes 2 — TD

---

Semestre 4



---

# Table des matières

---

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Exercice 1 . . . . .	4
1.2	Exercice 2 . . . . .	4
<b>2</b>	<b>Processus</b>	<b>6</b>
2.1	Exercice 3 . . . . .	6
2.2	Exercice 4 . . . . .	6
2.3	Exercice 5 . . . . .	7
2.4	Exercice 6 . . . . .	7
2.5	Exercice 7 . . . . .	7
<b>3</b>	<b>Gestion de la mémoire : mémoire virtuelle et allocation non contiguë</b>	<b>8</b>
<b>4</b>	<b>Structure interne du système de fichier d'Unix</b>	<b>9</b>
<b>5</b>	<b>Primitives Unix (POSIX.1) de manipulation des fichiers</b>	<b>10</b>

---

# Introduction

---

## 1.1 Exercice 1

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int i;
5     for(i=1; i < argc; ++i) {
6         printf("%s\n", argv[i]);
7     }
8
9     return 0;
10 }
```

Listing 1.1 – Exercice 1 – Version portable

```
1 #define _POSIX_C_SOURCE 1
2
3 #include <stdio.h>
4
5 int main(int argc, char** argv, char** envp) {
6     int i;
7     printf("Argument :\n");
8     for(i=1; i < argc ; ++i) {
9         printf("argv [%d]=%s\n", i, argv[i]);
10    }
11    i=0;
12    while(envp[i] != NULL) {
13        printf("%s\n", envp[i++]);
14    }
15
16    return 0;
17 }
```

Listing 1.2 – Exercice 1 – Version Unix

## 1.2 Exercice 2

```
1 #define _POSIX_C_SOURCE 1
2
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(int argc, char** argv) {
7     if(argv != 1 || !(strlen(argv[1]))) {
8         return 1;
9     }
10
11     printf("%s = %s\n", argv[1], getenv(argv[1]));
```

12 | }

## Listing 1.3 – Exercice 2

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char** argv) {
5      char *valeur;
6
7      if(argc != 2) {
8          fprintf(stderr, "Usage : %s variable\n", argv[0]);
9          return (1);
10     }
11
12     valeur = getenv(argv[1]);
13
14     if(valeur == NULL) {
15         fprintf(stderr, "Variable %s inconnue \n", argv[1]);
16         return (2);
17     }
18
19     printf("%s=%s", argv[1], valeur);
20
21     return 0;
22 }
```

## Listing 1.4 – Exercice 2 – Correction

# Processus

## 2.1 Exercice 3

```
1 #define _POSIX_C_SOURCE 1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6
7 int main(int argc, char** argv) {
8     pid_t pid;
9     switch(pid = fork()) {
10         case -1:
11             perror("fork");
12             exit(1);
13         case 0: //fils
14             printf("Exécuté par le fils\n");
15             printf("PID du père: %d\n", (int)getppid());
16             printf("PID du fils %d\n\n", (int)getpid());
17             break;
18         default: //père
19             printf("Exécuté par le père\n");
20             printf("PID du père: %d\n", (int)getpid());
21             break;
22     }
23
24     return 0;
25 }
```

Listing 2.1 – Exercice 3

## 2.2 Exercice 4

```
1 #define _POSIX_C_SOURCE 1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <sys/types.h>
5 #include <unistd.h>
6 #include <sys/wait.h>
7
8 #define NBPROC 4 //nombre de processus fils
9 #define N 5 // nombre d'affichage
10
11 void affichage(pid_t pid);
12
13 int main(int argc, char** argv) {
14     pid_t fils, fils_termine;
15     pid_t successeur;
16     int i;
```

```

17  int circonstance;
18
19  successeur = getpid();
20
21  for(i = NBPROC; i > 0 ; --i) {
22      switch(fils = fork()) {
23          case -1:
24              perror("Erreur fork");
25              exit(EXIT_FAILURE);
26          case 0:
27              printf("Je suis le fils %d, je m'appelle %d mon successeur est %d\n",
28                  i, (int)getpid(), successeur);
29              affichage(getpid());
30              printf("[%d] fin\n", (int)getpid());
31              exit(i);
32          default:
33              successeur = fils;
34      }
35  }
36
37  printf("Je suis le père, je m'appelle %d mon successeur est %d",
38      (int)getpid(), (int)successeur);
39  while(fils_termine = wait(&circonstance) != -1) {
40      printf("[%d] : Mon fils %d est terminé avec le code %d\n",
41          (int)fils_termine, WEXITSTATUS(circonstance));
42  }
43  printf("Tous les fils sont terminés");
44
45  return EXIT_SUCCESS;
46 }
47
48 void affichage(pid_t pid) {
49     int k;
50     for (k=0 ; k < N ; ++k) {
51         printf("[%d] : c'est moi\n", (int) pid);
52         sleep(2); // processus pas de l'état élu à l'état bloqué
53     }
54 }

```

Listing 2.2 – Exercice 4

## 2.3 Exercice 5

```
| execlp("date", "date", NULL);
```

```
| Maintenant:
| vendredi 8 février 2013, 15:13:33 (UTC+0100)
```

## 2.4 Exercice 6

Il va afficher deux fois le pid du programme courant.

## 2.5 Exercice 7

---

# Gestion de la mémoire : mémoire virtuelle et allocation non contiguë

---

CHAPITRE

**3**



# Structure interne du système de fichier d'Unix

---

4

---

# Primitives Unix (POSIX.1) de manipulation des fichiers

---