

# 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>9</b>  |
| 3.1      | Exercice 8 . . . . .  | 9         |
| 3.1.1    | . . . . .   | 9         |
| 3.1.2    | . . . . .   | 9         |
| 3.2      | Exercice 9 . . . . .  | 9         |
| 3.2.1    | Temps d'accès moyen . . . . .   | 9         |
| 3.2.2    | Taux maximal de défaut de page . . . . .                                    | 10        |
| 3.3      | Exercice 10 . . . . .   | 10        |
| 3.3.1    | . . . . .   | 10        |
| <b>4</b> | <b>Structure interne du système de fichier d'Unix</b>                       | <b>11</b> |
| <b>5</b> | <b>Primitives Unix (POSIX.1) de manipulation des fichiers</b>               | <b>12</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

```

1 | #define _POSIX_C_SOURCE 1

```

```
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/wait.h>
7
8 int main(void) {
9     pid_t fils;
10    switch ( fils = fork() ) {
11        case -1 :
12            perror("Erreur fork");
13            exit(EXIT_FAILURE);
14        case 0 :
15            execlp("ls", "ls", "-l", NULL);
16            perror("Erreur execlp");
17            exit(EXIT_FAILURE);
18        default :
19            wait(NULL);
20            break;
21    }
22
23    switch ( fils=fork() ) {
24        case -1 :
25            perror("Erreur fork");
26            exit(EXIT_FAILURE);
27        case 0 :
28            execlp("pwd", "pwd", NULL);
29            perror("Erreur execlp");
30            exit(EXIT_FAILURE);
31        default :
32            wait(NULL);
33            break;
34    }
35
36    return EXIT_SUCCESS;
37 }
```

Listing 2.3 – Exercice 7



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

## 3.1 Exercice 8

### 3.1.1

- **Pages 2 et 3**  $[5 \times 1024; 6 \times 1024 - 1] = [2048; 4095]$
- **Page 5**  $[7 \times 1024; 8 \times 1024 - 1] = [5120; 6143]$
- **Page 7**  $[2 \times 1024; 4 \times 1024 - 1] = [7168; 8191]$

### 3.1.2

**R** Numéro de cadre  $\times 1024 + \text{décalage}$

| Adresse virtuelle | Page virtuelle | Page physique  | Adresse physique              |
|-------------------|----------------|----------------|-------------------------------|
| 0                 | 0              | 3              | $3 \times 1024 = 3072$        |
| 3728              | 3              | défaut de page | défaut de page                |
| 1023              | 0              | 3              | $3 \times 1024 + 1023 = 4095$ |
| 1024              | 1              | 1              | 1024                          |
| 1025              | 1              | 1              | 1025                          |
| 7800              | 7              | défaut de page | défaut de page                |
| 4096              | 4              | 2              | 2048                          |

## 3.2 Exercice 9

$d$  : taux de défaut de page.

### 3.2.1 Temps d'accès moyen

$$(1 - d) \times 2 \times 0.5 + d \times (20 + 2 \times 0.5) = (1 - d) + 21d = 1 + 20d$$

### 3.2.2 Taux maximal de défaut de page

$$\begin{aligned}1 + 20d &< 1.2 \\ d &< \frac{0.2}{20} \\ d &< 0.01\end{aligned}$$

## 3.3 Exercice 10

### 3.3.1

Chaque processus provoque 5 défauts de page correspondants aux changements initiaux des 5 pages.

Soit  $n$  le nombre de page référencancés par un processus. Le taux de défaut de page est  $\frac{5}{n}$ .

# Structure interne du système de fichier d'Unix

---

4

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

```

1  #define POSIX_C_SOURCE 1
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <unistd.h>
5  #include <sys/types.h>
6  #include <sys/stat.h>
7  #include <fcntl.h>
8
9  #define TAILLE_BLOC 512
10
11  /* code de retour */
12  #define TOUT_VA_BIEN      0
13  #define ERR_NB_PARAM      1
14  #define ERR_ACCES_FICHER  2
15  #define ERR_CREATION_FICHER 3
16
17  int main(int argc, char** argv) {
18      int src, dest;
19      int nbLus;
20      char bloc[TAILLE_BLOC];
21
22      if(argc != 3) {
23          fprintf(stderr, "Usage: %s fichiersource fichierdest", argv3);
24          exit(ERR_NB_PARAM);
25      }
26
27      src = open(argv[1], O_RDONLY);
28      if(src == -1) {
29          perror(argv[1]);
30          exit(ERR_ACCES_FICHER);
31      }
32
33      dest = open(argv[2], O_WRONLY|O_TRUNC|O_CREAT, S_IRUSR|S_IWUSR|S_IRGRP);
34      if(dest == -1) {
35          perror(argv[2]);
36          close(src);
37          exit(ERR_CREATION_FICHER);
38      }
39
40      while((nbLus = read(src, bloc, TAILLE_BLOC)) > 0) {
41          write(dest, bloc, nbLus);
42      }
43      close(src);
44      close(dest);
45
46      return (TOUT_VA_BIEN);
47  }

```

Listing 5.1 – Exercice 16