

CHAPITRE 1 – PROCESSUS

Exercice 1.1 :

Le but de cet exercice est de créer un programme qui lancera un processus puis s'endormira juste après. De cette façon, on essaiera de constater que le processus fils devient un processus zombie.

Code source :

Header.h :

```
#ifndef HEADER_H
#define HEADER_H

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

#endif // HEADER_H
```

Main.c :

```
#include "headers/header.h"

int main(int argc, char** argv) {
    printf("Programme principal\n");
    int pid = fork();

    if (pid == -1) { printf("Fork impossible\n");
    } else if (pid == 0) {
        //Le processus fils désire mourir mais doit attendre que son père
        //récupérer son état pour se terminer.
        printf("Processus fils\n");
        exit(0);
    } else {
        //Le processus père ne pourra pas récupérer l'état du processus fils
        //pour que celui-ci puisse se terminer.
        printf("Processus père\n");
        pause();
    }
    return (EXIT_SUCCESS);
}
```

Trace d'exécution :

```
jeremie@Extensa-2511: ~/Developpement/NetBeansProjects/OS_Exercice_1_1
jeremie@Extensa-2511:~/Developpement/NetBeansProjects/OS_Exercice_1_1$ cd Developpement/NetBeansProjects/OS_Exercice_1_1/
jeremie@Extensa-2511:~/Developpement/NetBeansProjects/OS_Exercice_1_1$ ll
total 20K
drwxrwxrwx 3 jeremie jeremie 4,0K mai 27 22:15 build
drwxrwxrwx 3 jeremie jeremie 4,0K mai 27 22:15 dist
-rwxrwxrwx 1 jeremie jeremie 3,5K mai 24 14:36 Makefile
drwxrwxrwx 3 jeremie jeremie 4,0K mai 27 22:15 nbproject
drwxrwxrwx 3 jeremie jeremie 4,0K mai 27 22:15 src
jeremie@Extensa-2511:~/Developpement/NetBeansProjects/OS_Exercice_1_1$ ./dist/Debug/GNU-Linux/os_processus
Programme principal
Processus père
Processus fils
```

Une fois lancé, le programme crée bien deux processus, un père et un fils. En exécutant la commande « ps -e -o pid,ppid,stat,cmd » on obtient ceci :

```
jeremie@Extensa-2511: ~/Developpement/NetBeansProjects/OS_Exercice_1_1
1834 1086 Sl /usr/lib/gvfs/gvfsd-network --spawner :1.4 /org/gtk/gvfs/exec_spaw/2
1863 1086 Sl /usr/lib/gvfs/gvfsd-dnssd --spawner :1.4 /org/gtk/gvfs/exec_spaw/6
1909 1086 S /usr/lib/x86_64-linux-gnu/gconf/gconfd-2
4718 709 S /usr/sbin/dnsmasq --no-resolv --keep-in-foreground --no-hosts --bind-interfaces --pid-file=/v
5093 1086 Sl /usr/lib/gvfs/gvfsd-http --spawner :1.4 /org/gtk/gvfs/exec_spaw/8
8886 2 S [kworker/1:1]
9259 2 S [kworker/2:2]
9467 2 S [kworker/0:3]
9470 2 S [kworker/u16:46]
10940 2 S [irq/50-mei_me]
10997 709 S /sbin/dhclient -d -q -sf /usr/lib/NetworkManager/nm-dhcp-helper -pf /var/run/dhclient-wlp3s0.
11228 1086 Sl /usr/lib/libreoffice/program/oosplash --writer
11368 2 S [kworker/0:0]
11369 2 S [kworker/u16:0]
11376 2 S [kworker/1:2]
11377 2 S [kworker/3:1]
11380 2 S [kworker/2:0]
11432 11228 Sl /usr/lib/libreoffice/program/soffice.bin
11478 2 S [kworker/1:0]
11480 2 S [kworker/2:1]
11481 2 S [kworker/3:2]
11530 2 S [kworker/u16:1]
11536 2 S [kworker/0:1]
11547 1086 Sl /usr/lib/gnome-terminal/gnome-terminal-server
11554 11547 Ss bash
11573 11554 S+ ./dist/Debug/GNU-Linux/os_processus
11574 11573 Z+ [os_processus] <defunct>
11575 11547 Ss bash
11588 11575 R+ ps -e -o pid,ppid,stat,cmd
jeremie@Extensa-2511:~/Developpement/NetBeansProjects/OS_Exercice_1_1$
```

On constate alors que le processus 11574, lancé par le processus 11573 est en mode Zombie (Z+). En effet, le processus fils envoie un signal indiquant au père qu'il est arrivé au bout de son fil d'exécution, il cherche alors à mourir, mais le père est en mode sommeil et ne peut pas récupérer le signal envoyé par le fils. Le processus fils passe alors en mode zombie.

Hue Pierrick

En interrompant le processus, on constate la disparition du processus père et du processus zombie :

```
jeremie@Extensa-2511: ~/Developpement/NetBeansProjects/OS_Exercice_1_1
1764 1086 SL /usr/lib/x86_64-linux-gnu/unity-lens-files/unity-files-daemon
1818 1310 SL /usr/lib/x86_64-linux-gnu/deja-dup/deja-dup-monitor
1834 1086 SL /usr/lib/gvfs/gvfsd-network --spawner :1.4 /org/gtk/gvfs/exec_spaw/2
1863 1086 SL /usr/lib/gvfs/gvfsd-dnssd --spawner :1.4 /org/gtk/gvfs/exec_spaw/6
1909 1086 S /usr/lib/x86_64-linux-gnu/gconf/gconfd-2
4718 709 S /usr/sbin/dnsmasq --no-resolv --keep-in-foreground --no-hosts --bind-interfaces --pid-file=/v
5093 1086 SL /usr/lib/gvfs/gvfsd-http --spawner :1.4 /org/gtk/gvfs/exec_spaw/8
8886 2 S [kworker/1:1]
9470 2 S [kworker/u16:46]
10940 2 S [irq/50-mei_me]
10997 709 S /sbin/dhclient -d -q -sf /usr/lib/NetworkManager/nm-dhcp-helper -pf /var/run/dhclient-wlp3s0.
11228 1086 SL /usr/lib/libreoffice/program/oosplash --writer
11368 2 S [kworker/0:0]
11432 11228 SL /usr/lib/libreoffice/program/soffice.bin
11478 2 S [kworker/1:0]
11480 2 S [kworker/2:1]
11530 2 S [kworker/u16:1]
11547 1086 SL /usr/lib/gnome-terminal/gnome-terminal-server
11554 11547 Ss bash
11627 2 S [kworker/u16:2]
11638 2 S [kworker/2:0]
11640 2 S [kworker/3:1]
11684 2 S [kworker/0:1]
11709 2 S [kworker/2:2]
11713 2 S [kworker/3:2]
11719 2 S [kworker/u16:0]
11740 2 S [kworker/0:2]
11750 2 S [kworker/1:2]
11753 11554 R+ ps -e -o pid,ppid,stat,cmd
jeremie@Extensa-2511:~/Developpement/NetBeansProjects/OS_Exercice_1_1$
```

Exercice 1.2 :

Le but de cet exercice est de nous faire créer deux processus concurrents qui vont partager des descripteurs de fichier. De cette façon, on pourra rediriger les flux d'entrée sortie (dans notre cas, de sortie) vers un fichier.

Code source :

Header.h

```
#ifndef HEADER_H
#define HEADER_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#endif // HEADER_H
```

Main.c

```
#include "headers/header.h"

int main(int argc, char** argv) {
    int pid_filsDate = 0;
    int pid_filsAttente = 0;

    FILE* canal;
    canal = fopen("message.log", "wt");
    if (canal == NULL) {
        printf("%s\n", "Impossible d'ouvrir le fichier message.log");
        exit(0);
    }

    pid_filsDate = fork();
    if (pid_filsDate == 0) {
        for (int i = 0; i < 10; i++) {
            sleep(3);
            pid_filsDate = fork();

            dup2(canal->_fileno, 1);
            //Redirection de la sortie standard vers message.log.

            if (pid_filsDate == 0) {
                execl("/bin/date", "date", "-u", NULL);
                //execl ecrase le processus qui l'a lancé.
                //Les petits fils ne créeront pas d'autres processus.
            }
        }
    }
}
```

```
    }

    close(canal->_fileno);
    exit(1);
}

pid_filsAttente = fork();
if (pid_filsAttente == 0) {
    for (int i = 0; i < 30; i++) {
        sleep(2);
        pid_filsAttente = fork();

        dup2(canal->_fileno, 1);
        //Redirection de la sortie standard vers message.log.

        if (pid_filsAttente == 0) {
            printf("Attendre\n");
            break; //Seul le fils parcourt la boucle, pas les petits fils.
        }
    }

    close(canal->_fileno);
    exit(2);
}

while (wait(NULL) > -1) {

}

dup2(canal->_fileno, 1);
printf("%s\n", "C'est terminé !");
close(canal->_fileno);
fclose(canal);
}
```

En exécutant le code ci-dessus, on remplit au fur et à mesure un fichier message.log qui contient les sorties du programme « date -u » et la sortie de notre programme. En effet, en utilisant les descripteurs de fichier, il nous est possible de rediriger le flux de sortie de printf vers un fichier (en l'occurrence, message.log).

Trace d'exécution :

Contenu de message.log :

[illegible]

On constate bien que l'écriture des sorties se fait dans le fichier `message.log`. De plus, on peut remarquer qu'à chaque appel de la fonction `sleep()`, l'ordonnanceur passe la main à un autre processus, d'où l'enchevêtrement de l'affichage. Le `timecode` nous montre toutefois que les timings sont bien respectés.

Une fois tous les fils terminés, le processus père quitte en affichant « C'est terminé ! ».

CHAPITRE 2 - COMMUNICATION INTER-PROCESSUS (IPC)

Exercice 2.1 :

Le but de ce programme est de permettre d'intercepter les signaux émis aux processus pour associer des actions à ces signaux. Dans notre cas, on interceptera les signaux SIGTERM (15 - envoyé par défaut grâce à la commande « kill ») pour incrémenter une variable.

Code source :

Header.h :

```
#ifndef HEADER_H
#define HEADER_H

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

void handler(int);

#endif // HEADER_H
```

Main.c :

```
#include "headers/header.h"

int N = 0; //Attention, variable globale !

int main(int argc, char** argv) {
    signal(SIGTERM, handler); //Assignment d'un handler
    while (1) {
        printf("Toc toc %d\n", N);
        sleep(2);
    }
    return (EXIT_SUCCESS);
}

void handler(int sig) {
    if (sig == SIGTERM) {
        N += 1;
        signal(SIGTERM, handler); //Sans cela le programme s'arrête au prochain
                                   //SIGTERM
    } else if (sig == SIGQUIT) { exit(0); }
}
```

Exercice 2.2 :

Le but de cet exercice est de nous faire ouvrir un canal de communication nommé entre deux processus sans lien de parenté. Le premier processus (le producteur) va enfiler des caractères aléatoires dans le pipe. Le second processus (le consommateur) va les lire et lorsque deux caractères identiques successifs sont lus, on quitte les deux programmes.

Code source :

Header.h :

```
#ifndef HEADER_H
#define HEADER_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <time.h>
#include <sys/signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>

#endif // HEADER_H
```

Producer.c :

```
#include "headers/header.h"

int main(int argc, char** argv) {
    char* pipeName = argv[1];
    printf("Producer\n");
    char* character = (char*) malloc(1 * sizeof(char));
    srand(time(NULL));

    int fileno = open(pipeName, O_WRONLY);
    int fifo = mkfifo(pipeName, 0666);
    if (fileno == -1) {
        printf("Erreur à l'ouverture du pipe.\n");
        exit(-1);
    }

    while (1) {
        *character = 'A' + rand() % 26;
        int wrote;
        do {
            wrote = write(fileno, character, 1);
        } while (wrote == -1);

        *character = 'A';
        sleep(1);
    }
}
```

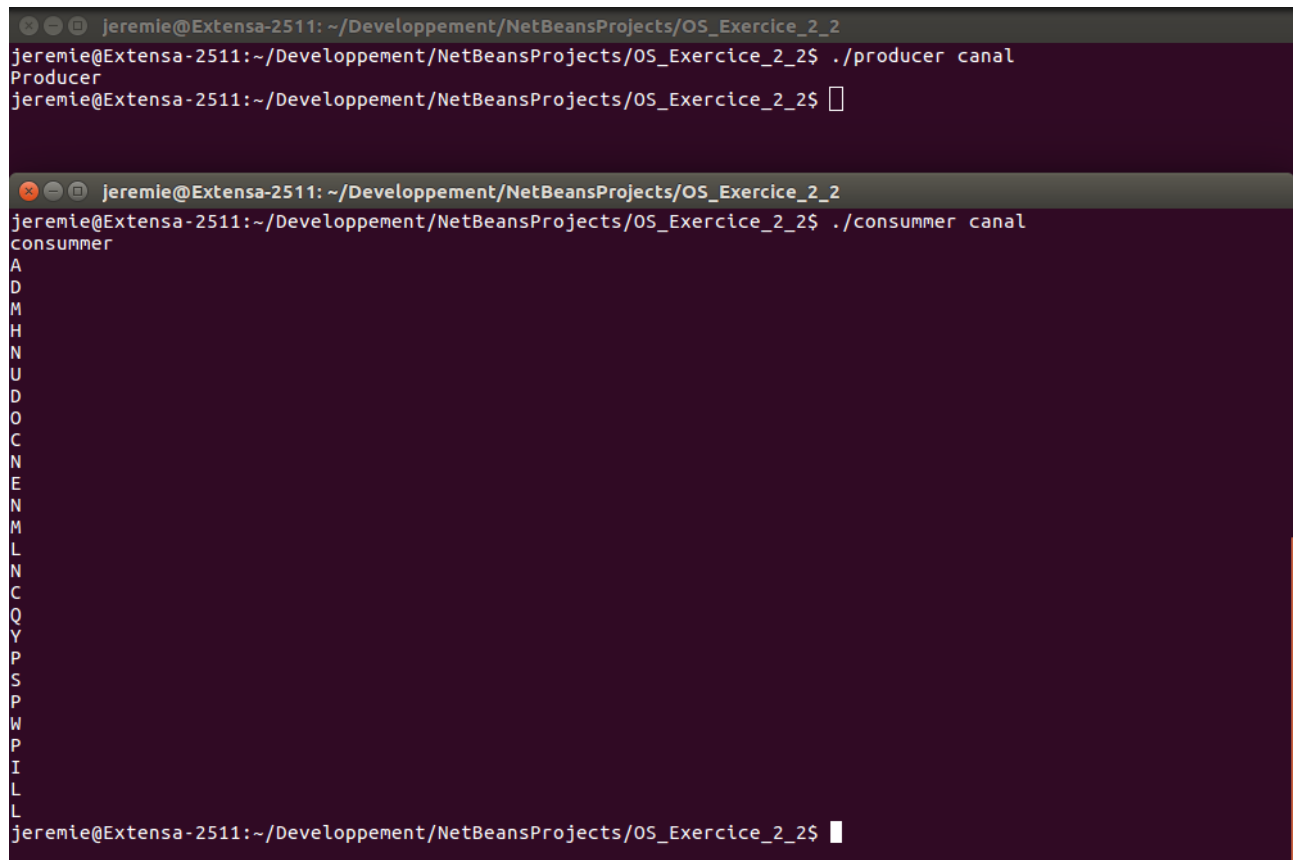
```
    }  
  
    free(character);  
    close(fileno);  
  
    return EXIT_SUCCESS;  
}
```

Consumer.c :

```
#include "headers/header.h"  
  
int main (int argc, char** argv) {  
    char* pipeName = argv[1];  
    printf("Consumer\n");  
    char* buffer = (char*) malloc(1 * sizeof(char));  
    char* lastChar = (char*) malloc(1 * sizeof(char));  
    srand(time(NULL));  
  
    int fileno = open(pipeName, O_RDONLY);  
    int fifo = mkfifo(pipeName, 0666);  
    if (fileno == -1) {  
        printf("Erreur à l'ouverture du pipe.\n");  
        exit(-1);  
    }  
  
    lastChar[0] = 'a';  
    int readed, sameChar = 0;  
    do {  
        readed = read(fileno, buffer, 1);  
        if (readed != -1) {  
            printf("%s\n", buffer);  
        }  
  
        if (strcmp(lastChar, buffer) == 0) {  
            sameChar = 1;  
        } else {  
            strcpy(lastChar, buffer);  
        }  
    } while (sameChar != 1);  
  
    free(buffer);  
    close(fileno);  
  
    return EXIT_SUCCESS;  
}
```

Trace d'exécution :

On lance, dans deux consoles différentes, les programmes producer et consumer en spécifiant un même nom de canal en argument. Voici le résultat observé :



```
jeremie@Extensa-2511: ~/Developpement/NetBeansProjects/OS_Exercice_2_2
jeremie@Extensa-2511:~/Developpement/NetBeansProjects/OS_Exercice_2_2$ ./producer canal
Producer
jeremie@Extensa-2511:~/Developpement/NetBeansProjects/OS_Exercice_2_2$ █

jeremie@Extensa-2511: ~/Developpement/NetBeansProjects/OS_Exercice_2_2
jeremie@Extensa-2511:~/Developpement/NetBeansProjects/OS_Exercice_2_2$ ./consumer canal
consumer
A
D
M
H
N
U
D
O
C
N
E
N
M
L
N
C
Q
Y
P
S
P
W
P
I
L
L
jeremie@Extensa-2511:~/Developpement/NetBeansProjects/OS_Exercice_2_2$ █
```

Comme on peut le constater, le programme producer remplit le canal avec des caractères et le programme consumer les lit. Lorsque deux caractères identiques successifs sont repérés, le programme consumer ferme le pipe et quitte, faisant quitter le programme producer par la même occasion.

Exercice 2.3 :

Le but de ce programme est de permettre de faire des échanges entre le processus père et le processus fils via des pipes. Ainsi, le processus père s'occupera de récupérer la chaîne de caractère et de l'afficher après transformation quant au processus fils, il devra transformer cette chaîne en majuscule.

Code source :

Header.h :

```
#ifndef HEADER_H
#define HEADER_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <time.h>
#include <ctype.h>
#include <sys/signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>

#define BUFF_SIZE 32
void clearBuffer(char*);

#endif // HEADER_H
```

Main.c :

```
#include "headers/header.h"

int main(void) {
    int tube_pere_fils[2];
    int tube_fils_pere[2];
    if (pipe(tube_pere_fils) == 1 || pipe(tube_fils_pere)) {
        perror("Pipes invalides\n");
        exit(0);
    }

    char* output = (char*) malloc(BUFF_SIZE * sizeof (char));
    char* input = (char*) malloc(BUFF_SIZE * sizeof (char));
    clearBuffer(input);
    clearBuffer(output);

    pid_t pid = fork();

    if (pid == -1) {
        perror("Fork impossible\n");
    }
```

```
    } else if (pid == 0) {
        close(tube_fils_pere[0]);
        close(tube_pere_fils[1]);

        do {
            read(tube_pere_fils[0], input, BUFF_SIZE);

            int i = 0;
            for (i = 0; i < strlen(input); i++) {
                output[i] = toupper(input[i]);
            }
            output[i] = '\0';

            write(tube_fils_pere[1], output, strlen(input) + 1);
        } while (input[0] != 'q' && input[1] != 'u' && input[2] != 'i' &&
input[3] != 't'); //Detection de "quit"
    } else {
        close(tube_pere_fils[0]);
        close(tube_fils_pere[1]);

        do {
            printf("String to transform : ");
            fgets(output, BUFF_SIZE, stdin);

            if (output[0] == 'q' && output[1] == 'u' && output[2] == 'i' &&
output[3] == 't') { //Detection de "quit"
                write(tube_pere_fils[1], output, strlen(output) + 1);
                break;
            }

            write(tube_pere_fils[1], output, strlen(output) + 1);
            read(tube_fils_pere[0], input, strlen(output) + 1);

            printf("-> %s\n", input);
        } while (strcmp(output, "quit") != 0);
    }

    return EXIT_SUCCESS;
}

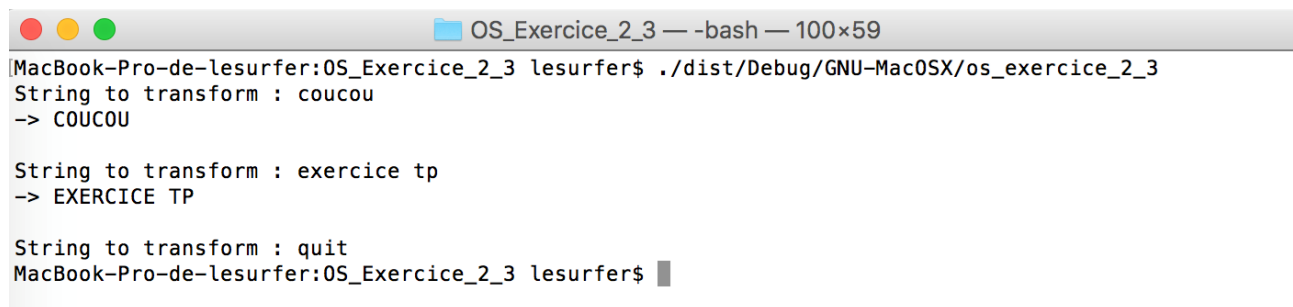
void clearBuffer(char* buff_out) {
    for (int i = 0; i < BUFF_SIZE; i++) {
        buff_out[i] = '\0';
    }
}
```

Trace d'exécution :

Au lancement du programme :

- Le processus père demande la saisi d'une chaîne de caractère.
- Le père écrit la chaîne de caractère dans le pipe pour la transmettre à son fils.
- Le fils va lire le pipe afin de réceptionner la chaîne pour la transformer en majuscule.
- Une fois transformé, le processus fils va écrire la chaîne dans le pipe.
- Le processus père va lire le pipe afin de récupérer le résultat retourné par le processus fils pour l'afficher.
- Si la chaîne saisi est 'quit' alors le programme s'arrête.

Voici le résultat observé :



```
OS_Exercice_2_3 — -bash — 100x59
MacBook-Pro-de-lesurfer:OS_Exercice_2_3 lesurfer$ ./dist/Debug/GNU-MacOSX/os_exercice_2_3
String to transform : coucou
-> COUCOU

String to transform : exercice tp
-> EXERCICE TP

String to transform : quit
MacBook-Pro-de-lesurfer:OS_Exercice_2_3 lesurfer$
```

Comme on peut le constater ci-dessus, la chaîne de caractère prise en entrée par le processus père est en minuscule et celle-ci est transformé en majuscule via le processus fils. On peut constater, que dès la saisi du mot 'quit', on quitte l'application.

Exercice 2.4 :

Le but de ce programme est de nous permettre d'effectuer une opération mathématique en utilisant la commande bc (Sous UNIX, cette commande permet d'implémenter une calculatrice en mode ligne). Cette commande sera appelée dans le processus fils permettant d'effectuer l'opération et le processus père s'occupera simplement de récupérer la chaîne entrée et d'afficher le résultat de l'opération.

Code source :

Header.h :

```
#ifndef HEADER_H
#define HEADER_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <time.h>
#include <ctype.h>
#include <sys/signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>

#define BUFF_SIZE 42

void clearBuffer(char*);

#endif // HEADER_H
```

Main.c

```
#include "headers/header.h"

int main(void) {
    int tube_pere_fils[2];
    int tube_fils_pere[2];
    if (pipe(tube_pere_fils) == 1 || pipe(tube_fils_pere) == 1) {
        perror("Pipes invalides\n");
    }

    char* output = (char*) malloc(BUFF_SIZE * sizeof (char));
    char* input = (char*) malloc(BUFF_SIZE * sizeof (char));
    clearBuffer(input);
    clearBuffer(output);

    pid_t pid = fork();

    if (pid == -1) {
        perror("Fork impossible\n");
    } else if (pid == 0) {
        close(tube_pere_fils[1]);
        close(tube_fils_pere[0]);

        dup2(tube_pere_fils[0], STDIN_FILENO);
        dup2(tube_fils_pere[1], STDOUT_FILENO);
```

```
        execl("/usr/bin/bc", "bc", NULL, NULL);
    } else {
        close(tube_pere_fils[0]);
        close(tube_fils_pere[1]);

        do {
            printf("String to compute : ");
            fgets(output, BUFF_SIZE, stdin);

            if (output[0] == 'q' && output[1] == 'u' && output[2] == 'i' &&
                output[3] == 't') { //Detection de "quit"
                write(tube_pere_fils[1], output, strlen(output));
                break;
            }

            write(tube_pere_fils[1], output, strlen(output));
            read(tube_fils_pere[0], input, strlen(output));

            printf("-> %s\n", input);

            clearBuffer(input);
            clearBuffer(output);
        } while (strcmp(output, "quit") != 0);
    }

    return EXIT_SUCCESS;
}

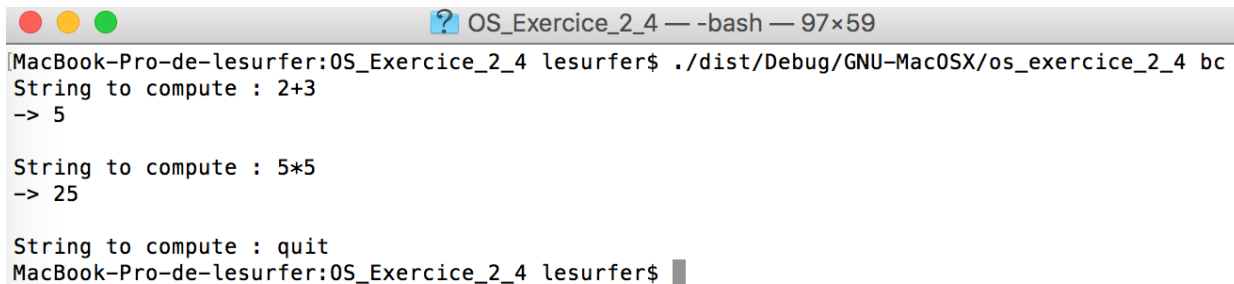
void clearBuffer(char* buff_out) {
    for (int i = 0; i < BUFF_SIZE; i++) {
        buff_out[i] = '\0';
    }
}
```

Trace d'exécution :

Au lancement du programme :

- Le processus père demande la saisi de l'opération.
- Opération est écrite dans le pipe par le processus père.
- Le processus fils, lit l'opération et effectue l'opération.
- Le processus fils écrit le résultat dans le pipe.
- Le processus père lit le pipe et affiche le résultat de l'opération transmit par le processus fils.

Voici le résultat observé :



```
OS_Exercice_2_4 — -bash — 97x59
MacBook-Pro-de-lesurfer:OS_Exercice_2_4 lesurfer$ ./dist/Debug/GNU-MacOSX/os_exercice_2_4 bc
String to compute : 2+3
-> 5

String to compute : 5*5
-> 25

String to compute : quit
MacBook-Pro-de-lesurfer:OS_Exercice_2_4 lesurfer$
```

On constate que le processus père a comme valeur, la chaîne de caractère saisie par l'utilisateur. Ainsi le processus fils s'occupe quant à lui d'effectuer l'opération et le processus père récupère la chaîne et l'affiche une fois la transformation effectuée. Si le mot 'quit' est saisi alors on quitte l'application.

Exercice 2.5 :

Le but de ce programme est d'interpréter des commandes pour faire un programme de dessin. Ainsi le programme board et l'interpréteur devront communiquer via des tubes.

Code source :

Header.h :

```
#ifndef HEADER_H
#define HEADER_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <time.h>
#include <ctype.h>
#include <sys/signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>

#define BUFF_SIZE 42

void clearBuffer(char*);
struct DrawingCommand* parseCommand(char*);

#endif // HEADER_H
```

DrawingCommands.h :

```
#ifndef _DRAWING_COMMANDS_H_
#define _DRAWING_COMMANDS_H_

struct LineArgs {
    int x1;           /**< Colonne du premier point */
    int y1;           /**< Ligne du premier point */
    int x2;           /**< Colonne du second point */
    int y2;           /**< Ligne du second point */
};

struct CircleArgs {
    int x;             /**< Colonne du centre du cercle */
    int y;             /**< Ligne du centre du cercle */
    int radius;        /**< Rayon du cercle */
};

struct ColorArgs {
    int red;           /**< Composante rouge de la couleur */
    int green;         /**< Composante verte de la couleur */
    int blue;          /**< Composante bleue de la couleur */
};
```

```
};

enum CommandType {
    Pen = 0,          /**< Changer la couleur de trait */
    Fill,             /**< Changer la couleur de remplissage */
    Line,             /**< Dessiner une ligne droite */
    Circle,           /**< Dessiner un cercle */
    Quit              /**< Fermer le programme de dessin */
};

/**
 * Structure de commandes de dessin. Un champs (type) donne
 * le type de la commande et un champ de type union donne les
 * argument de la commande.
 */
struct DrawingCommand {
    enum CommandType type;      /**< Type de commande */
    union {
        struct LineArgs line;   /**< Arguments pour une ligne */
        struct CircleArgs circle; /**< Arguments pour un cercle */
        struct ColorArgs color; /**< Arguments pour une couleur
                                   (trait/remplissage) */
    } args;                    /**< Arguments de la commande */
};

#endif /* !defined _DRAWING_COMMANDS_H_ */
```

Main.c :

```
#include "headers/header.h"
#include "Board_full/Board/DrawingCommands.h"

int main(void) {
    int father_son_pipe[2];
    int son_father_pipe[2];
    if (pipe(father_son_pipe) == 1 || pipe(son_father_pipe) == 1) {
        perror("Pipes invalides\n");
    }

    char* output = (char*) malloc(BUFF_SIZE * sizeof (char));
    char* input = (char*) malloc(BUFF_SIZE * sizeof (char));
    clearBuffer(input);
    clearBuffer(output);

    pid_t pid = fork();

    if (pid == -1) {
        perror("Fork impossible\n");
    } else if (pid == 0) {
        close(father_son_pipe[1]);
        close(son_father_pipe[0]);

        dup2(father_son_pipe[0], STDIN_FILENO);
```

```
dup2(son_father_pipe[1], STDOUT_FILENO);

execl("/home/jeremie/Developpement/NetBeansProjects/OS_Exercice_2_5/src/Board_fu
ll/Board/board", "board", "--", NULL);
} else {
    close(father_son_pipe[0]);
    close(son_father_pipe[1]);

    do {
        printf("shell_board : ");
        fgets(output, BUFF_SIZE, stdin);

        struct DrawingCommand* dCom = parseCommand(output);

        if (output[0] == 'q' && output[1] == 'u' && output[2] == 'i' &&
            output[3] == 't') { //Detection de "quit"
            write(father_son_pipe[1], dCom, sizeof (dCom));
            read(son_father_pipe[0], input, BUFF_SIZE);

            free(dCom);
            break;
        }

        write(father_son_pipe[1], dCom, sizeof (dCom));
        read(son_father_pipe[0], input, BUFF_SIZE);

        printf("Board says : %s\n", input);

        clearBuffer(input);
        clearBuffer(output);
    } while (strcmp(output, "quit") != 0);
}
return EXIT_SUCCESS;
}

void clearBuffer(char* buff_out) {
    for (int i = 0; i < BUFF_SIZE; i++) {
        buff_out[i] = '\0';
    }
}

struct DrawingCommand* parseCommand(char* input) {
    char buff[10];
    int arg1, arg2, arg3, arg4;

    sscanf(input, "%s %d %d %d %d", buff, &arg1, &arg2, &arg3, &arg4);

    struct DrawingCommand* dCom = (struct DrawingCommand*) malloc (1 * sizeof(struct
DrawingCommand));

    if (strcmp(buff, "line") == 0) {
        dCom->type = Line;
```

```
    } else if (strcmp(buff, "pen") == 0) {
        dCom->type = Pen;
    } else if (strcmp(buff, "fill") == 0) {
        dCom->type = Fill;
    } else if (strcmp(buff, "circle") == 0) {
        dCom->type = Circle;
    } else if (strcmp(buff, "quit") == 0) {
        dCom->type = Quit;
    }

    struct LineArgs lArgs;
    struct ColorArgs coArgs;
    struct CircleArgs ciArgs;

    switch (dCom->type) {
        case Line:
            lArgs.x1 = arg1;
            lArgs.y1 = arg2;
            lArgs.x2 = arg3;
            lArgs.y2 = arg4;

            dCom->args.line = lArgs;
            break;
        case Pen:
            coArgs.red = arg1;
            coArgs.green = arg2;
            coArgs.blue = arg3;

            dCom->args.color = coArgs;
            break;
        case Fill:
            coArgs.red = arg1;
            coArgs.green = arg2;
            coArgs.blue = arg3;

            dCom->args.color = coArgs;
            break;
        case Circle:
            ciArgs.x = arg1;
            ciArgs.y = arg2;
            ciArgs.radius = arg3;

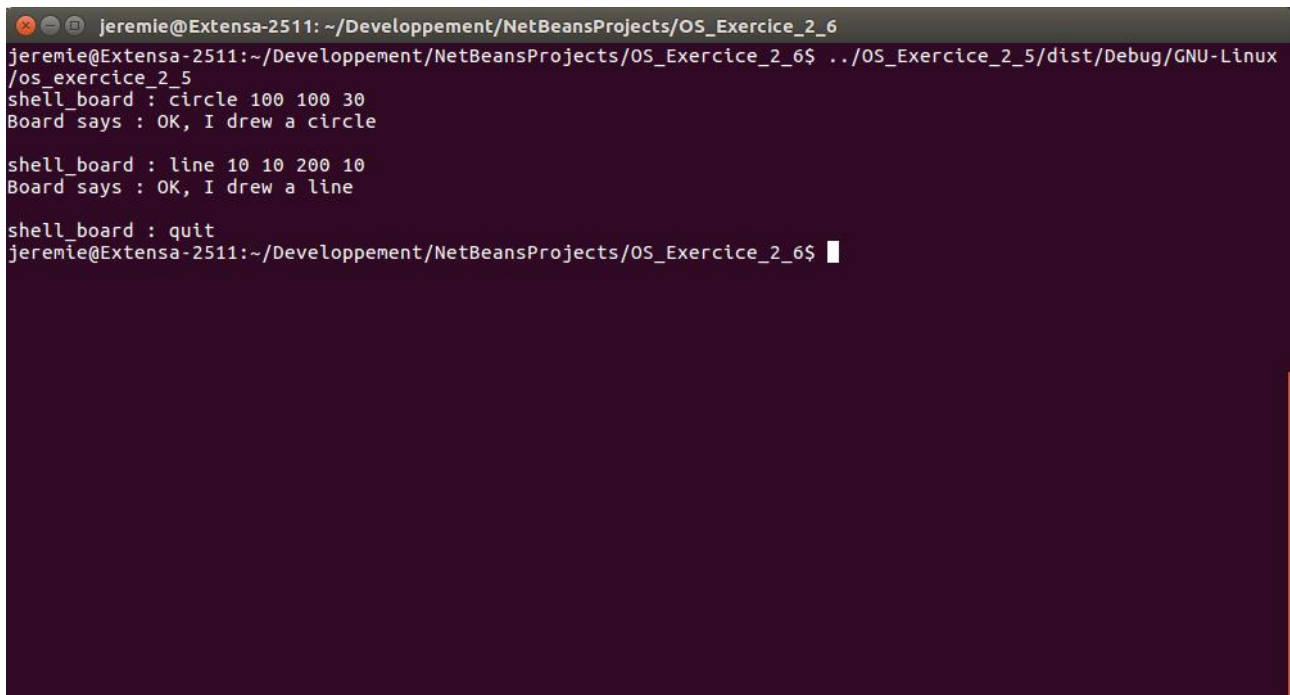
            dCom->args.circle = ciArgs;
            break;
        case Quit:
            break;
        default:
            printf("Unknowm command\n");
            break;
    }
    return dCom;
}
```

Trace d'exécution :

Au lancement du programme :

- Le processus fils ouvre une fenêtre graphique.
- Le processus père récupère la commande saisie par l'utilisateur.
- Celle-ci est ensuite écrite dans le pipe par le processus père.
- Le processus fils traite la commande et met à jour la fenêtre graphique.
- Le processus fils écrit le résultat dans le pipe.
- Le processus père lit le pipe et affiche le résultat de la commande.

Voici le résultat observé :



```
jeremie@Extensa-2511:~/Developpement/NetBeansProjects/OS_Exercice_2_6
jeremie@Extensa-2511:~/Developpement/NetBeansProjects/OS_Exercice_2_6$ ../OS_Exercice_2_5/dist/Debug/GNU-Linux
/os_exercice_2_5
shell_board : circle 100 100 30
Board says : OK, I drew a circle

shell_board : line 10 10 200 10
Board says : OK, I drew a line

shell_board : quit
jeremie@Extensa-2511:~/Developpement/NetBeansProjects/OS_Exercice_2_6$
```

On constate que les commandes sont bien prises en compte mais dans notre cas, le résultat de celles-ci ne s'affiche pas dans la fenêtre graphique. Il est possible que le dessin dans la fenêtre graphique se fasse en blanc et apparaisse sur le fond blanc de l'image, mais à cause d'un bug de l'application board, il ne nous est pas possible de changer la couleur du dessin.

Exercice 2.6 :

Le but de ce programme est de garantir l'alternance stricte entre le processus père et le processus fils. Par conséquent pour effectuer cette opération d'alternance, deux sémaphores doivent être créés.

Code source :

Header.h :

```
#ifndef HEADER_H
#define HEADER_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <time.h>
#include <ctype.h>
#include <sys/signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#define BUFF_SIZE 42

void clearBuffer(char*);
struct DrawingCommand* parseCommand(char*);

#endif // HEADER_H
```

Main.c :

```
#include "headers/header.h"

void p(int, int);
void v(int, int);

int main(int argc, char** argv) {
    key_t key = ftok(argv[0], 'E');
    int semid = semget(key, 2, IPC_CREAT | IPC_EXCL | 0600);
    if (semid == -1) {
        printf("%s\n", "Creation de semaphore impossible.");
        exit(0);
    }

    semctl(semid, 0, SETVAL, 1);
    semctl(semid, 1, SETVAL, 0);

    int count = 10;
```

```
pid_t pid = fork();

if (pid == -1) {
    printf("%s\n", "Fork impossible.");
} else if (pid == 0) {
    while (count > 0) {
        p(semid, 1);
        printf("Je suis le processus fils de PID %d\n", getpid());
        count -= 1;
        v(semid, 0);
    }
} else {
    while (count > 0) {
        p(semid, 0);
        printf("Je suis le processus pere de PID %d\n", getpid());
        count -= 1;
        v(semid, 1);
    }
}

semctl(semid, 0, IPC_RMID, 0);
return (EXIT_SUCCESS);
}

void p(int semid, int num) {
    struct sembuf op;
    op.sem_flg = 0;
    op.sem_num = num;
    op.sem_op = -1;

    semop(semid, &op, 1);
}

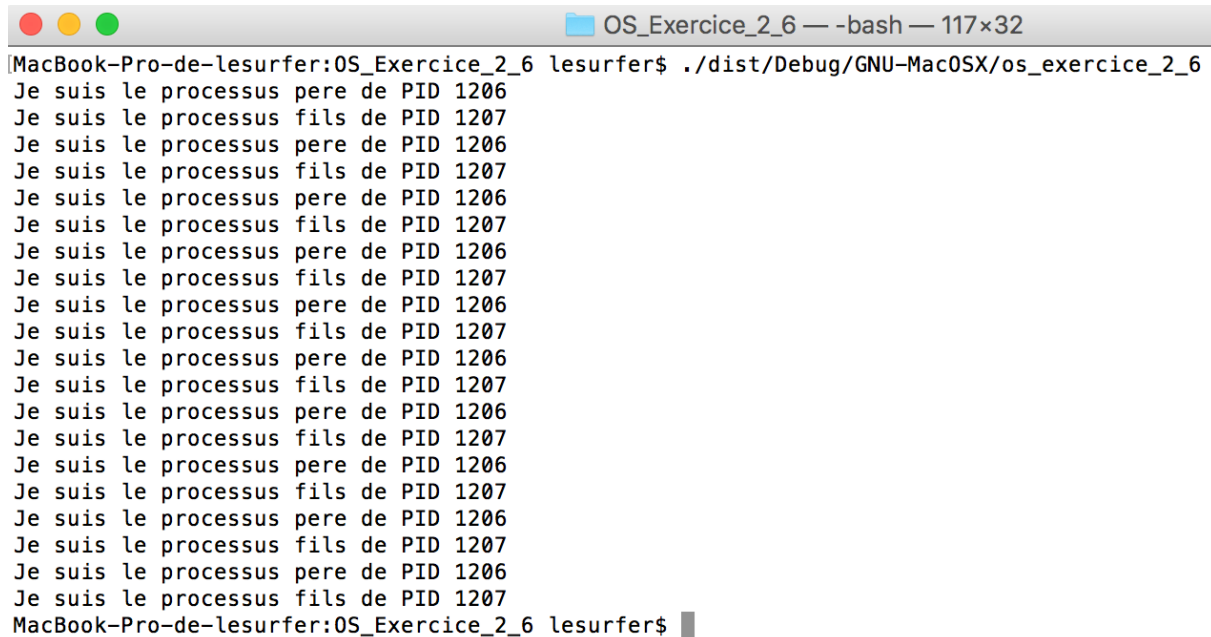
void v(int semid, int num) {
    struct sembuf op;
    op.sem_flg = 0;
    op.sem_num = num;
    op.sem_op = 1;

    semop(semid, &op, 1);
}
```

Trace d'exécution :

Au lancement du programme, le processus père prendra la main afin d'alterner avec le processus fils. Chaque processus affichera 10 fois un message de manière alternés afin de bien voir l'action des deux fonctions avec le protocole de d'acquisition et le protocole de libération.

Voici les résultats observés :



```
MacBook-Pro-de-lesurfer:OS_Exercice_2_6 lesurfer$ ./dist/Debug/GNU-MacOSX/os_exercice_2_6
Je suis le processus pere de PID 1206
Je suis le processus fils de PID 1207
Je suis le processus pere de PID 1206
Je suis le processus fils de PID 1207
Je suis le processus pere de PID 1206
Je suis le processus fils de PID 1207
Je suis le processus pere de PID 1206
Je suis le processus fils de PID 1207
Je suis le processus pere de PID 1206
Je suis le processus fils de PID 1207
Je suis le processus pere de PID 1206
Je suis le processus fils de PID 1207
Je suis le processus pere de PID 1206
Je suis le processus fils de PID 1207
Je suis le processus pere de PID 1206
Je suis le processus fils de PID 1207
Je suis le processus pere de PID 1206
Je suis le processus fils de PID 1207
Je suis le processus pere de PID 1206
Je suis le processus fils de PID 1207
MacBook-Pro-de-lesurfer:OS_Exercice_2_6 lesurfer$
```

On peut constater que les deux processus alterne correctement en affichant leur pid. Chaque processus applique donc correctement le protocole d'acquisition et de libération.

Exercice 2.7 :

Le but de ce programme est de simuler le diner de 5 philosophes en utilisant les sémaphores sur les philosophes. Ainsi chaque philosophe sera représenté par un processus. Un philosophe peut être dans différents états : Manger (E), Penser (T) et Demander à manger (A). Un philosophe à le droit de manger uniquement si les deux fourchettes sont disponibles, autrement dit, si ses voisins de droite et de gauche ne mangent pas.

Code source :

Header.h :

```
#ifndef HEADER_H
#define HEADER_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <time.h>
#include <ctype.h>
#include <sys/signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>

#define THINK 'T' //Penser
#define ASK 'A' //Demande à manger
#define EAT 'E' //Manger

void Probeer (int semid, int semNum); //P = Probeer ('Try')
void Verhoog (int semid, int semNum); //V = Verhoog
('Increment', 'Increase by one').

void printPhilState (int semid, int philNum, char* philStatus);
void checkIfPhilCanEat (int semid, int philNum, char* philStatus);
void setPhilState (char* philStatus, int philNum, char state);
void think (int semid, int philNum, char* philStatus);
void ask (int semid, int philNum, char* philStatus);
void eat (int semid, int philNum, char* philStatus);
void thinkAskEatRepeat (int semid, int philNum, char* philStatus);

#endif // HEADER_H
```

Main.c :

```
#include "headers/header.h"

int main(int argc, char** argv) {
    key_t semKey = ftok(argv[0], 'E');
    int semid = semget(semKey, 7, IPC_CREAT | IPC_EXCL | 0600);
    if (semid == -1) {
        printf("%s\n", "Creation de semaphore impossible.");
        exit(0);
    }

    semctl(semid, 0, SETVAL, 0);
    semctl(semid, 1, SETVAL, 0);
    semctl(semid, 2, SETVAL, 0);
    semctl(semid, 3, SETVAL, 0);
    semctl(semid, 4, SETVAL, 0);

    semctl(semid, 5, SETVAL, 1); //Gestion de l'accès à la ressource partagée
    semctl(semid, 6, SETVAL, 1); //Gestion de l'accès à l'affichage.

    int shmKey = ftok(argv[0], 'J');
    int shmid = shmget(shmKey, 6 * sizeof (char), IPC_CREAT | 0600);
    if (shmid == -1) {
        printf("%s\n", "Creation de shmem impossible.");
        exit(0);
    }

    //Obtention des informations
    printf("semid : %d\n", semid);
    printf("shmid : %d\n\n", shmid);
    fflush(stdout);

    char* philStatus = shmat(shmid, NULL, SHM_R | SHM_W);
    for (int i = 0; i < 5; i++) {
        philStatus[i] = THINK;
    }
    philStatus[5] = '\0';

    pid_t pid;

    for (int i = 0; i < 5; i++) {
        pid = fork();
        srand(time(NULL) + getpid());

        if (pid == -1) {
            printf("%s\n", "Fork impossible.");
            exit(-1);
        } else if (pid == 0) {
            int count = 3;

            while (count > 0) {
                thinkAskEatRepeat(semid, i, philStatus);
```

```
        count--;  
    }  
  
    exit(1);  
}  
}  
  
for (int i = 0; i < 5; i++) {  
    wait(NULL);  
}  
semctl(semid, 0, IPC_RMID, 0);  
shmctl(shmid, IPC_RMID, NULL);  
return (EXIT_SUCCESS);  
}  
  
void Probeer(int semid, int num) {  
    struct sembuf op;  
    op.sem_flg = 0;  
    op.sem_num = num;  
    op.sem_op = -1;  
    semop(semid, &op, 1);  
}  
  
void Verhoog(int semid, int num) {  
    struct sembuf op;  
    op.sem_flg = 0;  
    op.sem_num = num;  
    op.sem_op = 1 ;  
    semop(semid, &op, 1);  
}  
  
void printPhilState(int semid, int philNum, char* philStatus) {  
    Probeer(semid, 6);  
    for (int i = 0; i < 5; i++) {  
        switch (philStatus[i]) {  
            case THINK:  
                printf("[%s] Philosophe %d %s\n", philStatus, i, "pense.");  
                break;  
            case ASK:  
                printf("[%s] Philosophe %d %s\n", philStatus, i, "demande à  
manger.");  
                break;  
            case EAT:  
                printf("[%s] Philosophe %d %s\n", philStatus, i, "mange.");  
                break;  
        }  
        fflush(stdout);  
    }  
    printf("\n");  
    Verhoog(semid, 6);  
}  
  
void checkIfPhilCanEat(int semid, int philNum, char* philStatus) {
```

```
    if (philStatus[(philNum + 4) % 5] != EAT) {
        if (philStatus[(philNum + 1) % 5] != EAT) {
            if (philStatus[philNum] == ASK) {
                Verhoog(semid, philNum);
            }
        }
    }
}

void setPhilState(char* philStatus, int philNum, char state) {
    switch (state) {
        case THINK:
            philStatus[philNum] = THINK;
            break;
        case ASK:
            philStatus[philNum] = ASK;
            break;
        case EAT:
            philStatus[philNum] = EAT;
            break;
        default:
            break;
    }
}

void think(int semid, int philNum, char* philStatus) {
    int timeThinking = rand() % 3 + 1;

    Probeer(semid, 5);
    setPhilState(philStatus, philNum, THINK);
    Verhoog(semid, 5);

    sleep(timeThinking);
}

void ask(int semid, int philNum, char* philStatus) {
    Probeer(semid, 5);
    setPhilState(philStatus, philNum, ASK);

    checkIfPhilCanEat(semid, philNum, philStatus);
    Verhoog(semid, 5);
}

void eat(int semid, int philNum, char* philStatus) {
    Probeer(semid, philNum);
    int timeEating = rand() % 3 + 1;

    Probeer(semid, 5);
    setPhilState(philStatus, philNum, EAT);
    Verhoog(semid, 5);

    sleep(timeEating);

    Probeer(semid, 5);
}
```

```

    setPhilState(philStatus, philNum, THINK);

    checkIfPhilCanEat(semid, (philNum + 4) % 5, philStatus);
    checkIfPhilCanEat(semid, (philNum + 1) % 5, philStatus);
    Verhoog(semid, 5);
}

void thinkAskEatRepeat(int semid, int philNum, char* philStatus) {
    think(semid, philNum, philStatus);
    printPhilState(semid, philNum, philStatus);

    ask(semid, philNum, philStatus);
    printPhilState(semid, philNum, philStatus);

    eat(semid, philNum, philStatus);
    printPhilState(semid, philNum, philStatus);
}

```

Trace d'exécution :

Au lancement du programme, les 5 philosophes pensent pendant un temps aléatoire. L'un d'entre eux va vérifier que ses voisins ne sont pas a table il demande à manger. Si la place est libre, il mange. A la fin du repas d'un philosophe, on vérifie que l'un des voisins demande à manger. Si c'est le cas, on place un ticket (Verhoog) pour ce philosophe.

Voici les résultats observés :

semid : 0	[TAETTT] Philosophe 0 pense.
shmid : 24150017	[TAETTT] Philosophe 1 demande à manger.
	[TAETTT] Philosophe 2 mange.
	[TAETTT] Philosophe 3 pense.
	[TAETTT] Philosophe 4 pense.
[TTTTT] Philosophe 0 pense.	[AAETTT] Philosophe 0 demande à manger.
[TTTTT] Philosophe 1 pense.	[AAETTT] Philosophe 1 demande à manger.
[TTTTT] Philosophe 2 pense.	[AAETTT] Philosophe 2 mange.
[TTTTT] Philosophe 3 pense.	[AAETTT] Philosophe 3 pense.
[TTTTT] Philosophe 4 pense.	[AAETTT] Philosophe 4 pense.
[TTATT] Philosophe 0 pense.	[EAETTT] Philosophe 0 mange.
[TTATT] Philosophe 1 pense.	[EAETTT] Philosophe 1 demande à manger.
[TTATT] Philosophe 2 demande à manger.	[EAETTT] Philosophe 2 mange.
[TTATT] Philosophe 3 pense.	[EAETTT] Philosophe 3 pense.
[TTATT] Philosophe 4 pense.	[EAETTT] Philosophe 4 pense.
[TTETT] Philosophe 0 pense.	[EAEAT] Philosophe 0 mange.
[TTETT] Philosophe 1 pense.	[EAEAT] Philosophe 1 demande à manger.
[TTETT] Philosophe 2 mange.	[EAEAT] Philosophe 2 mange.
[TTETT] Philosophe 3 pense.	[EAEAT] Philosophe 3 demande à manger.
[TTETT] Philosophe 4 pense.	[EAEAT] Philosophe 4 pense.
[TAETTT] Philosophe 0 pense.	[EAEAA] Philosophe 0 mange.
[TAETTT] Philosophe 1 demande à manger.	[EAEAA] Philosophe 1 demande à manger.
[TAETTT] Philosophe 2 mange.	[EAEAA] Philosophe 2 mange.
[TAETTT] Philosophe 3 pense.	
[TAETTT] Philosophe 4 pense.	

[EAEAA] Philosophe 3 demande à manger.	[ATAET] Philosophe 3 mange.
[EAEAA] Philosophe 4 demande à manger.	[ATAET] Philosophe 4 pense.
[EATEA] Philosophe 0 mange.	[ATAET] Philosophe 0 demande à manger.
[EATEA] Philosophe 1 demande à manger.	[ATAET] Philosophe 1 pense.
[EATEA] Philosophe 2 pense.	[ATAET] Philosophe 2 demande à manger.
[EATEA] Philosophe 3 mange.	[ATAET] Philosophe 3 mange.
[EATEA] Philosophe 4 demande à manger.	[ATAET] Philosophe 4 pense.
[EATEA] Philosophe 0 mange.	[TTAET] Philosophe 0 mange.
[EATEA] Philosophe 1 demande à manger.	[TTAET] Philosophe 1 pense.
[EATEA] Philosophe 2 pense.	[TTAET] Philosophe 2 demande à manger.
[EATEA] Philosophe 3 mange.	[TTAET] Philosophe 3 mange.
[EATEA] Philosophe 4 demande à manger.	[TTAET] Philosophe 4 pense.
[EATEA] Philosophe 0 mange.	[TAAET] Philosophe 0 pense.
[EATEA] Philosophe 1 demande à manger.	[TAAET] Philosophe 1 demande à manger.
[EATEA] Philosophe 2 pense.	[TAAET] Philosophe 2 demande à manger.
[EATEA] Philosophe 3 mange.	[TAAET] Philosophe 3 mange.
[EATEA] Philosophe 4 demande à manger.	[TAAET] Philosophe 4 pense.
[EAAEA] Philosophe 0 mange.	[TAAET] Philosophe 0 pense.
[EAAEA] Philosophe 1 demande à manger.	[TAAET] Philosophe 1 demande à manger.
[EAAEA] Philosophe 2 demande à manger.	[TAAET] Philosophe 2 demande à manger.
[EAAEA] Philosophe 3 mange.	[TAAET] Philosophe 3 mange.
[EAAEA] Philosophe 4 demande à manger.	[TAAET] Philosophe 4 pense.
[TAAEA] Philosophe 0 pense.	[TEAET] Philosophe 0 pense.
[TAAEA] Philosophe 1 demande à manger.	[TEAET] Philosophe 1 mange.
[TAAEA] Philosophe 2 demande à manger.	[TEAET] Philosophe 2 demande à manger.
[TAAEA] Philosophe 3 mange.	[TEAET] Philosophe 3 mange.
[TAAEA] Philosophe 4 demande à manger.	[TEAET] Philosophe 4 pense.
[TEATA] Philosophe 0 pense.	[TEAEA] Philosophe 0 pense.
[TEATA] Philosophe 1 mange.	[TEAEA] Philosophe 1 mange.
[TEATA] Philosophe 2 demande à manger.	[TEAEA] Philosophe 2 demande à manger.
[TEATE] Philosophe 3 pense.	[TEAEA] Philosophe 3 mange.
[TEATE] Philosophe 4 mange.	[TEAEA] Philosophe 4 demande à manger.
[TEATT] Philosophe 0 pense.	[TEAEA] Philosophe 0 pense.
[TEATT] Philosophe 1 mange.	[TEAEA] Philosophe 1 mange.
[TEATT] Philosophe 2 demande à manger.	[TEAEA] Philosophe 2 demande à manger.
[TEATT] Philosophe 3 pense.	[TEAEA] Philosophe 3 mange.
[TEATT] Philosophe 4 pense.	[TEATE] Philosophe 4 demande à manger.
[TEATT] Philosophe 0 pense.	[AEATE] Philosophe 0 demande à manger.
[TEATT] Philosophe 1 mange.	[AEATE] Philosophe 1 mange.
[TEATT] Philosophe 2 demande à manger.	[AEATE] Philosophe 2 demande à manger.
[TEATT] Philosophe 3 pense.	[AEATE] Philosophe 3 pense.
[TEATT] Philosophe 4 pense.	[AEATE] Philosophe 4 mange.
[TEAAT] Philosophe 0 pense.	[AEATE] Philosophe 0 demande à manger.
[TEAAT] Philosophe 1 mange.	[AEATE] Philosophe 1 mange.
[TEAAT] Philosophe 2 demande à manger.	[AEATE] Philosophe 2 demande à manger.
[TEAAT] Philosophe 3 demande à manger.	[AEATE] Philosophe 3 pense.
[TEAAT] Philosophe 4 pense.	[AEATE] Philosophe 4 mange.
[TTAET] Philosophe 0 pense.	[AEATT] Philosophe 0 demande à manger.
[TTAET] Philosophe 1 pense.	[AEATT] Philosophe 1 mange.
[TTAET] Philosophe 2 demande à manger.	[AEATT] Philosophe 2 demande à manger.
[TTAET] Philosophe 3 mange.	[AEATT] Philosophe 3 pense.
[TTAET] Philosophe 4 pense.	[AEATT] Philosophe 4 pense.
[ATAET] Philosophe 0 demande à manger.	[ATATT] Philosophe 0 demande à manger.
[ATAET] Philosophe 1 pense.	[ATATT] Philosophe 1 pense.
[ATAET] Philosophe 2 demande à manger.	[ATATT] Philosophe 2 demande à manger.

[ATATT] Philosophe 3 pense.	[TTTET] Philosophe 4 pense.
[ATATT] Philosophe 4 pense.	
	[TATET] Philosophe 0 pense.
[ETETT] Philosophe 0 mange.	[TATET] Philosophe 1 demande à manger.
[ETETT] Philosophe 1 pense.	[TATET] Philosophe 2 pense.
[ETETT] Philosophe 2 mange.	[TATET] Philosophe 3 mange.
[ETETT] Philosophe 3 pense.	[TATET] Philosophe 4 pense.
[ETETT] Philosophe 4 pense.	
	[TATET] Philosophe 0 pense.
[ETETA] Philosophe 0 mange.	[TATET] Philosophe 1 demande à manger.
[ETETA] Philosophe 1 pense.	[TATET] Philosophe 2 pense.
[ETETA] Philosophe 2 mange.	[TATET] Philosophe 3 mange.
[ETETA] Philosophe 3 pense.	[TATET] Philosophe 4 pense.
[ETETA] Philosophe 4 demande à manger.	
	[TETTT] Philosophe 0 pense.
[TTETA] Philosophe 0 pense.	[TETTT] Philosophe 1 mange.
[TTETA] Philosophe 1 pense.	[TETTT] Philosophe 2 pense.
[TTETA] Philosophe 2 mange.	[TETTT] Philosophe 3 pense.
[TTETA] Philosophe 3 pense.	[TETTT] Philosophe 4 pense.
[TTETA] Philosophe 4 demande à manger.	
	[TTTTT] Philosophe 0 pense.
[TTETE] Philosophe 0 pense.	[TTTTT] Philosophe 1 pense.
[TTETE] Philosophe 1 pense.	[TTTTT] Philosophe 2 pense.
[TTETE] Philosophe 2 mange.	[TTTTT] Philosophe 3 pense.
[TTETE] Philosophe 3 pense.	[TTTTT] Philosophe 4 pense.
[TTETE] Philosophe 4 mange.	
	[TTTTT] Philosophe 0 pense.
[TTEAE] Philosophe 0 pense.	[TTTTT] Philosophe 1 pense.
[TTEAE] Philosophe 1 pense.	[TTTTT] Philosophe 2 pense.
[TTEAE] Philosophe 2 mange.	[TTTTT] Philosophe 3 pense.
[TTEAE] Philosophe 3 demande à manger.	[TTTTT] Philosophe 4 pense.
[TTEAE] Philosophe 4 mange.	
	[TTATT] Philosophe 0 pense.
[TTEAT] Philosophe 0 pense.	[TTATT] Philosophe 1 pense.
[TTEAT] Philosophe 1 pense.	[TTATT] Philosophe 2 demande à manger.
[TTEAT] Philosophe 2 mange.	[TTATT] Philosophe 3 pense.
[TTEAT] Philosophe 3 demande à manger.	[TTATT] Philosophe 4 pense.
[TTEAT] Philosophe 4 pense.	
	[TTTTT] Philosophe 0 pense.
[TTTAT] Philosophe 0 pense.	[TTTTT] Philosophe 1 pense.
[TTTET] Philosophe 1 pense.	[TTTTT] Philosophe 2 pense.
[TTTET] Philosophe 2 pense.	[TTTTT] Philosophe 3 pense.
[TTTET] Philosophe 3 mange.	[TTTTT] Philosophe 4 pense.

On peut constater que les 5 philosophes pensent au départ puis au fur et à mesure, aléatoirement, l'un d'entre eux fait une demande pour manger et passe dans l'état « mange ».

Pour que les philosophes puissent vérifier l'état de leurs voisins, on utilise un segment de mémoire partagée qui contient une chaîne de caractère pour en faciliter l'affichage (l'état d'un philosophe correspond au caractère situé à l'indice du philosophe dans la chaîne).

Exercice 2.8 :

Le but de cet exercice est le même que celui de l'exercice 2.7 à ceci près que les sémaphores sont désormais attribués aux fourchettes plutôt qu'aux philosophes. De cette façon, nous n'avons qu'à modifier les étapes de vérification des demandes et effectuer la prise et la relâche de deux sémaphores chaque fois qu'un processus (philosophe) cherche à manger. Si les sémaphores ne sont pas disponibles, le philosophe sera mis en attente par le système.

Code source :

Header.h :

```
#ifndef HEADER_H
#define HEADER_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <time.h>
#include <ctype.h>
#include <sys/signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>

#define THINK 'T'
#define ASK 'A'
#define EAT 'E'

void Probeer_one (int semid, int semNum); //P = Probeer ('Try')
void Verhoog_one (int semid, int semNum); //V = Verhoog
('Increment', 'Increase by one').
void Probeer_two (int semid, int semNum_1, int semNum_2); //P =
Probeer ('Try')
void Verhoog_two (int semid, int semNum_1, int semNum_2); //V =
Verhoog ('Increment', 'Increase by one').
void printPhilState (int semid, int philNum, char* philStatus);
void setPhilState (char* philStatus, int philNum, char state);
void think (int semid, int philNum, char* philStatus);
void ask (int semid, int philNum, char* philStatus);
void eat (int semid, int philNum, char* philStatus);
void thinkAskEatRepeat (int semid, int philNum, char* philStatus);

#endif // HEADER_H
```

Main.c :

```
#include "headers/header.h"

int main(int argc, char** argv) {
    key_t semKey = ftok(argv[0], 'E');
    int semid = semget(semKey, 7, IPC_CREAT | IPC_EXCL | 0600);
    if (semid == -1) {
        printf("%s\n", "Creation de semaphore impossible.");
        exit(0);
    }

    semctl(semid, 0, SETVAL, 1); //fourchette 0
    semctl(semid, 1, SETVAL, 1); //fourchette 1
    semctl(semid, 2, SETVAL, 1); //fourchette 2
    semctl(semid, 3, SETVAL, 1); //fourchette 3
    semctl(semid, 4, SETVAL, 1); //fourchette 4

    semctl(semid, 5, SETVAL, 1); //Gestion de l'accès à la ressource partagée
    semctl(semid, 6, SETVAL, 1); //Gestion de l'accès à l'affichage.

    int shmKey = ftok(argv[0], 'J');
    int shmid = shmget(shmKey, 6 * sizeof (char), IPC_CREAT | 0600);
    if (shmid == -1) {
        printf("%s\n", "Creation de shmem impossible.");
        exit(0);
    }

    //Obtention des informations
    printf("semid : %d\n", semid);
    printf("shmid : %d\n\n", shmid);
    fflush(stdout);

    char* philStatus = shmat(shmid, NULL, SHM_R | SHM_W);
    for (int i = 0; i < 5; i++) {
        philStatus[i] = THINK;
    }
    philStatus[5] = '\0';

    pid_t pid;

    for (int i = 0; i < 5; i++) {
        pid = fork();
        srand(time(NULL) + getpid());

        if (pid == -1) {
            printf("%s\n", "Fork impossible.");
            exit(-1);
        } else if (pid == 0) {
            int count = 3;

            while (count > 0) {
                thinkAskEatRepeat(semid, i, philStatus);
```

```
        count--;  
    }  
  
    printPhilState(semid, i, philStatus);  
  
    exit(1);  
}  
}  
  
for (int i = 0; i < 5; i++) {  
    wait(NULL);  
}  
  
semctl(semid, 0, IPC_RMID, 0);  
shmctl(shmid, IPC_RMID, NULL);  
return (EXIT_SUCCESS);  
}  
  
void Probeer_one(int semid, int num) {  
    struct sembuf op;  
    op.sem_flg = 0;  
    op.sem_num = num;  
    op.sem_op = -1;  
  
    semop(semid, &op, 1);  
}  
  
void Verhoog_one(int semid, int num) {  
    struct sembuf op;  
    op.sem_flg = 0;  
    op.sem_num = num;  
    op.sem_op = 1;  
  
    semop(semid, &op, 1);  
}  
  
void Probeer_two(int semid, int num_1, int num_2) {  
    struct sembuf ops[2];  
    ops[0].sem_flg = 0;  
    ops[0].sem_num = num_1;  
    ops[0].sem_op = -1;  
  
    ops[1].sem_flg = 0;  
    ops[1].sem_num = num_2;  
    ops[1].sem_op = -1;  
  
    semop(semid, ops, 2);  
}  
  
void Verhoog_two(int semid, int num_1, int num_2) {  
    struct sembuf ops[2];  
    ops[0].sem_flg = 0;  
    ops[0].sem_num = num_1;
```

```
ops[0].sem_op = 1;

ops[1].sem_flg = 0;
ops[1].sem_num = num_2;
ops[1].sem_op = 1;

semop(semid, ops, 2);
}

void printPhilState(int semid, int philNum, char* philStatus) {
    Probeer_one(semid, 6);
    for (int i = 0; i < 5; i++) {
        switch (philStatus[i]) {
            case THINK:
                printf("[%s] Philosophe %d %s\n", philStatus, i, "pense.");
                break;
            case ASK:
                printf("[%s] Philosophe %d %s\n", philStatus, i, "demande à
manger.");
                break;
            case EAT:
                printf("[%s] Philosophe %d %s\n", philStatus, i, "mange.");
                break;
        }
        fflush(stdout);
    }

    printf("\n");
    Verhoog_one(semid, 6);
}

/*void checkIfPhilCanEat(int semid, int philNum, char* philStatus) {
    if (philStatus[(philNum + 4) % 5] != EAT) {
        if (philStatus[(philNum + 1) % 5] != EAT) {
            if (philStatus[philNum] == ASK) {
                Verhoog_one(semid, philNum);
            }
        }
    }
}*/

void setPhilState(char* philStatus, int philNum, char state) {
    switch (state) {
        case THINK:
            philStatus[philNum] = THINK;
            break;
        case ASK:
            philStatus[philNum] = ASK;
            break;
        case EAT:
            philStatus[philNum] = EAT;
            break;
        default:
```

```
        break;
    }
}

void think(int semid, int philNum, char* philStatus) {
    int timeThinking = rand() % 3 + 1;

    Probeer_one(semid, 5);
    setPhilState(philStatus, philNum, THINK);
    Verhoog_one(semid, 5);

    printPhilState(semid, philNum, philStatus);
    sleep(timeThinking);
}

void ask(int semid, int philNum, char* philStatus) {
    Probeer_one(semid, 5);
    setPhilState(philStatus, philNum, ASK);
    Verhoog_one(semid, 5);

    printPhilState(semid, philNum, philStatus);
}

void eat(int semid, int philNum, char* philStatus) {
    Probeer_two(semid, (philNum + 4) % 5, philNum);
    int timeEating = rand() % 3 + 1;

    Probeer_one(semid, 5);
    setPhilState(philStatus, philNum, EAT);
    Verhoog_one(semid, 5);

    printPhilState(semid, philNum, philStatus);
    sleep(timeEating);

    Probeer_one(semid, 5);
    setPhilState(philStatus, philNum, THINK);
    Verhoog_one(semid, 5);
    Verhoog_two(semid, (philNum + 4) % 5, philNum);
}

void thinkAskEatRepeat(int semid, int philNum, char* philStatus) {
    think(semid, philNum, philStatus);
    ask(semid, philNum, philStatus);
    eat(semid, philNum, philStatus);
}
```

Trace d'exécution :

Au lancement du programme chaque philosophe est en train de penser. Aléatoirement, l'un d'entre eux fait une demande pour manger, se saisi de deux fourchettes et se met à table. En fin de repas, il repose les deux fourchettes. Si un autre philosophe voisin avait eu l'envie de se mettre à table et que les fourchettes étaient déjà prises, le système les aurait mis en attente, une fois les fourchettes libérées, le système les réveille et ils reprennent leur évolution.

Voici les résultats observés :

semid : 262144	[TETET] Philosophe 4 pense.
shmid : 13795348	
[TTTTT] Philosophe 0 pense.	[AETET] Philosophe 0 demande à manger.
[TTTTT] Philosophe 1 pense.	[AETET] Philosophe 1 mange.
[TTTTT] Philosophe 2 pense.	[AETET] Philosophe 2 pense.
[TTTTT] Philosophe 3 pense.	[AETET] Philosophe 3 mange.
[TTTTT] Philosophe 4 pense.	[AETET] Philosophe 4 pense.
[TTTTT] Philosophe 0 pense.	[AEAET] Philosophe 0 demande à manger.
[TTTTT] Philosophe 1 pense.	[AEATT] Philosophe 1 mange.
[TTTTT] Philosophe 2 pense.	[AEATT] Philosophe 2 demande à manger.
[TTTTT] Philosophe 3 pense.	[AEATT] Philosophe 3 pense.
[TTTTT] Philosophe 4 pense.	[AEATT] Philosophe 4 pense.
[TTTTT] Philosophe 0 pense.	
[TTTTT] Philosophe 1 pense.	[AEATT] Philosophe 0 demande à manger.
[TTTTT] Philosophe 2 pense.	[AEATT] Philosophe 1 mange.
[TTTTT] Philosophe 3 pense.	[AEATT] Philosophe 2 demande à manger.
[TTTTT] Philosophe 4 pense.	[AEATT] Philosophe 3 pense.
[TTTTT] Philosophe 0 pense.	[AEATT] Philosophe 4 pense.
[TTTTT] Philosophe 1 pense.	
[TTTTT] Philosophe 2 pense.	[AEATA] Philosophe 0 demande à manger.
[TTTTT] Philosophe 3 pense.	[AEATA] Philosophe 1 mange.
[TTTTT] Philosophe 4 pense.	[AEATA] Philosophe 2 demande à manger.
[TTTTT] Philosophe 0 pense.	[AEATA] Philosophe 3 pense.
[TTTTT] Philosophe 1 pense.	[AEATA] Philosophe 4 demande à manger.
[TTTTT] Philosophe 2 pense.	
[TTTTT] Philosophe 3 pense.	[AEATE] Philosophe 0 demande à manger.
[TTTTT] Philosophe 4 pense.	[AEATE] Philosophe 1 mange.
	[AEATE] Philosophe 2 demande à manger.
[TATTT] Philosophe 0 pense.	[AEATE] Philosophe 3 pense.
[TATTT] Philosophe 1 demande à manger.	[AEATE] Philosophe 4 mange.
[TATTT] Philosophe 2 pense.	
[TATTT] Philosophe 3 pense.	[ATATE] Philosophe 0 demande à manger.
[TATTT] Philosophe 4 pense.	[ATETE] Philosophe 1 pense.
	[ATETE] Philosophe 2 mange.
[TETTT] Philosophe 0 pense.	[ATETE] Philosophe 3 pense.
[TETTT] Philosophe 1 mange.	[ATETE] Philosophe 4 mange.
[TETTT] Philosophe 2 pense.	
[TETTT] Philosophe 3 pense.	[ATETE] Philosophe 0 demande à manger.
[TETTT] Philosophe 4 pense.	[ATETE] Philosophe 1 pense.
	[ATETE] Philosophe 2 mange.
[TETAT] Philosophe 0 pense.	[ATETE] Philosophe 3 pense.
[TETAT] Philosophe 1 mange.	[ATETE] Philosophe 4 mange.
[TETAT] Philosophe 2 pense.	
[TETAT] Philosophe 3 demande à manger.	[ATETT] Philosophe 0 demande à manger.
[TETAT] Philosophe 4 pense.	[ETETT] Philosophe 1 pense.
	[ETETT] Philosophe 2 mange.
[TETET] Philosophe 0 pense.	[ETETT] Philosophe 3 pense.
[TETET] Philosophe 1 mange.	[ETETT] Philosophe 4 pense.
[TETET] Philosophe 2 pense.	
[TETET] Philosophe 3 mange.	[ETETT] Philosophe 0 mange.

[ETETT] Philosophe 1 pense.	[ATETE] Philosophe 1 pense.
[ETETT] Philosophe 2 mange.	[ATETE] Philosophe 2 mange.
[ETETT] Philosophe 3 pense.	[ATETE] Philosophe 3 pense.
[ETETT] Philosophe 4 pense.	[ATETE] Philosophe 4 mange.
[EAEAT] Philosophe 0 mange.	[ATETE] Philosophe 0 demande à manger.
[EAEAT] Philosophe 1 demande à manger.	[ATETE] Philosophe 1 pense.
[EAEAT] Philosophe 2 mange.	[ATETE] Philosophe 2 mange.
[EAEAT] Philosophe 3 demande à manger.	[ATETE] Philosophe 3 pense.
[EAEAT] Philosophe 4 pense.	[ATETE] Philosophe 4 mange.
[EAEAT] Philosophe 0 mange.	[ATETE] Philosophe 0 demande à manger.
[EAEAT] Philosophe 1 demande à manger.	[ATETE] Philosophe 1 pense.
[EAEAT] Philosophe 2 mange.	[ATETE] Philosophe 2 mange.
[EAEAT] Philosophe 3 demande à manger.	[ATETE] Philosophe 3 pense.
[EAEAT] Philosophe 4 pense.	[ATETE] Philosophe 4 mange.
[TAEAT] Philosophe 0 pense.	[ATTTT] Philosophe 0 demande à manger.
[TAEAT] Philosophe 1 demande à manger.	[ETTTT] Philosophe 1 pense.
[TAEAT] Philosophe 2 mange.	[ETTTT] Philosophe 2 pense.
[TAEAT] Philosophe 3 demande à manger.	[ETTTT] Philosophe 3 pense.
[TAEAT] Philosophe 4 pense.	[ETTTT] Philosophe 4 pense.
[TATAT] Philosophe 0 pense.	[ETTTT] Philosophe 0 mange.
[TETET] Philosophe 1 mange.	[ETTTT] Philosophe 1 pense.
[TETET] Philosophe 2 pense.	[ETTTT] Philosophe 2 pense.
[TETET] Philosophe 3 mange.	[ETTTT] Philosophe 3 pense.
[TETET] Philosophe 4 pense.	[ETTTT] Philosophe 4 pense.
[TETET] Philosophe 0 pense.	[ETTTT] Philosophe 0 mange.
[TETET] Philosophe 1 mange.	[ETTTT] Philosophe 1 pense.
[TETET] Philosophe 2 pense.	[ETTTT] Philosophe 2 pense.
[TETET] Philosophe 3 mange.	[ETTTT] Philosophe 3 pense.
[TETET] Philosophe 4 pense.	[ETTTT] Philosophe 4 pense.
[TETET] Philosophe 0 pense.	[EATAT] Philosophe 0 mange.
[TETET] Philosophe 1 mange.	[EATAT] Philosophe 1 demande à manger.
[TETET] Philosophe 2 pense.	[EATAT] Philosophe 2 pense.
[TETET] Philosophe 3 mange.	[EATAT] Philosophe 3 demande à manger.
[TETET] Philosophe 4 pense.	[EATAT] Philosophe 4 pense.
[TETEA] Philosophe 0 pense.	[EATAT] Philosophe 0 mange.
[AETEA] Philosophe 1 mange.	[EATAT] Philosophe 1 demande à manger.
[AETEA] Philosophe 2 pense.	[EATAT] Philosophe 2 pense.
[AETEA] Philosophe 3 mange.	[EATAT] Philosophe 3 demande à manger.
[AETEA] Philosophe 4 demande à manger.	[EATAT] Philosophe 4 pense.
[AETEA] Philosophe 0 demande à manger.	[EATET] Philosophe 0 mange.
[AETEA] Philosophe 1 mange.	[EATET] Philosophe 1 demande à manger.
[AETEA] Philosophe 2 pense.	[EATET] Philosophe 2 pense.
[AETEA] Philosophe 3 mange.	[EATET] Philosophe 3 mange.
[AETEA] Philosophe 4 demande à manger.	[EATET] Philosophe 4 pense.
[AAAEA] Philosophe 0 demande à manger.	[EAAEA] Philosophe 0 mange.
[ATATE] Philosophe 1 pense.	[EAAEA] Philosophe 1 demande à manger.
[ATATE] Philosophe 2 demande à manger.	[EAAEA] Philosophe 2 demande à manger.
[ATATE] Philosophe 3 pense.	[EAAEA] Philosophe 3 mange.
[ATATE] Philosophe 4 mange.	[EAAEA] Philosophe 4 demande à manger.
[ATETE] Philosophe 0 demande à manger.	[EAAEA] Philosophe 0 mange.
[ATETE] Philosophe 1 pense.	[EAAEA] Philosophe 1 demande à manger.
[ATETE] Philosophe 2 mange.	[EAAEA] Philosophe 2 demande à manger.
[ATETE] Philosophe 3 pense.	[EAAEA] Philosophe 3 mange.
[ATETE] Philosophe 4 mange.	[EAAEA] Philosophe 4 demande à manger.
[ATETE] Philosophe 0 demande à manger.	[TAAEA] Philosophe 0 pense.

[TEAEA] Philosophe 1 mange.	[ATETE] Philosophe 4 mange.
[TEAEA] Philosophe 2 demande à manger.	
[TEAEA] Philosophe 3 mange.	[ATETE] Philosophe 0 demande à manger.
[TEAEA] Philosophe 4 demande à manger.	[ATETE] Philosophe 1 pense.
[TEAEA] Philosophe 0 pense.	[ATETE] Philosophe 2 mange.
[TEAEA] Philosophe 1 mange.	[ATETE] Philosophe 3 pense.
[TEAEA] Philosophe 2 demande à manger.	[ATETE] Philosophe 4 mange.
[TEAEA] Philosophe 3 mange.	
[TEAEA] Philosophe 4 demande à manger.	[ATETT] Philosophe 0 demande à manger.
	[ETTTT] Philosophe 1 pense.
[AEAEA] Philosophe 0 demande à manger.	[ETTTT] Philosophe 2 pense.
[AEAEA] Philosophe 1 mange.	[ETTTT] Philosophe 3 pense.
[AEAEA] Philosophe 2 demande à manger.	[ETTTT] Philosophe 4 pense.
[AEAEA] Philosophe 3 mange.	
[AEAEA] Philosophe 4 demande à manger.	
	[ETTTT] Philosophe 0 mange.
[AEATA] Philosophe 0 demande à manger.	[ETTTT] Philosophe 1 pense.
[AEATE] Philosophe 1 mange.	[ETTTT] Philosophe 2 pense.
[AEATE] Philosophe 2 demande à manger.	[ETTTT] Philosophe 3 pense.
[AEATE] Philosophe 3 pense.	[ETTTT] Philosophe 4 pense.
[AEATE] Philosophe 4 mange.	
	[ETTTT] Philosophe 0 mange.
	[ETTTT] Philosophe 1 pense.
[ATETE] Philosophe 0 demande à manger.	[ETTTT] Philosophe 2 pense.
[ATETE] Philosophe 1 pense.	[ETTTT] Philosophe 3 pense.
[ATETE] Philosophe 2 mange.	[ETTTT] Philosophe 4 pense.
[ATETE] Philosophe 3 pense.	
[ATETE] Philosophe 4 mange.	
	[TTTTT] Philosophe 0 pense.
[ATETE] Philosophe 0 demande à manger.	[TTTTT] Philosophe 1 pense.
[ATETE] Philosophe 1 pense.	[TTTTT] Philosophe 2 pense.
[ATETE] Philosophe 2 mange.	[TTTTT] Philosophe 3 pense.
[ATETE] Philosophe 3 pense.	[TTTTT] Philosophe 4 pense.

Dans l'exercice précédent, pour une raison inconnue, certains philosophes trop gourmands passaient à table en même temps que les autres. Dans le cas de l'exercice 2.8, aucun philosophe ne mange côte à côte.

CHAPITRE 3 – PROCESSUS LEGERS

Exercice 3.1 :

Dans cet exercice, nous allons tenter de montrer l'utilité d'un mutex lorsque l'on utilise des threads. Autrement dit, lorsque l'on déclare plusieurs fils d'exécution au sein d'un même programme. Dans le programme qui nous est fourni, on tente d'incrémenter une variable en décomposant cette opération en plusieurs sous opérations inutiles. De cette façon, l'ordonnanceur peut interrompre notre programme au moment où il effectue une action critique.

Code source :

Attention, pour compiler ce code, il ne faut pas oublier d'ajouter l'option « -lpthread » lors de l'appel de GCC.

Header.h :

```
#ifndef HEADER_H
#define HEADER_H

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#endif // HEADER_H
```

Main.c :

```
#include "headers/header.h"

int count = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void * counting(void * arg) {
    int i, a;

    for (i = 0; i < 1000000; i++) {
        pthread_mutex_lock(&mutex);

        a = count;
        a *= 3;
        a = (a / 3) + 1;
        count = a;

        pthread_mutex_unlock(&mutex);
    }

    return 0;
}
```

```
}  
  
int main(int argc, char** argv) {  
    int init = pthread_mutex_init(&mutex, NULL);  
    if (init != 0) {  
        printf("%s\n", "Création de mutex impossible");  
        exit(0);  
    }  
  
    int res1, res2;  
    pthread_t a_thread1, a_thread2;  
    void *thread_result1, *thread_result2;  
  
    res1 = pthread_create(&a_thread1, NULL, counting, NULL);  
    res2 = pthread_create(&a_thread2, NULL, counting, NULL);  
  
    printf("On attend la fin des threads\n");  
    pthread_join(a_thread1, &thread_result1);  
    pthread_join(a_thread2, &thread_result2);  
  
    printf("count = %d\n", count);  
  
    pthread_mutex_destroy(&mutex);  
  
    return (EXIT_SUCCESS);  
}
```

Trace d'exécution :

A l'exécution du programme :

```
On attend la fin des threads  
count = 2000000
```

En ajoutant un mutex, on peut protéger l'accès à la section critique de notre programme et ainsi garantir l'alternance stricte de nos deux fils d'exécution. De cette façon, notre variable est bien incrémentée à chaque passage dans la fonction counting.

Exercice 3.2 :

Dans cet exercice, nous allons créer plusieurs threads qui n'auront que pour objectif de compter un certain nombre de fois de 1 à N avec N quelconque. A la fin du comptage de chaque thread, celui-ci indique son nom et combien de fois il a compté. Dans un second temps, on ajoutera une exclusion mutuelle pour que les compteurs affichent leur ordre d'arrivée dès qu'ils ont fini de compter.

Code source :

Header.h :

```
#ifndef HEADER_H
#define HEADER_H

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <unistd.h>
#include <string.h>

typedef struct data {
    char* name;
    int n;
} DATA;

#endif // HEADER_H
```

Main.c :

```
#include "headers/header.h"

int rank = 1;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void* counting(void* arg) {
    DATA* data = (DATA*) arg;
    int time_sleeping;

    for (int i = 0; i < data->n; i++) {
        time_sleeping = rand() % 3 + 1;
        printf("%s : %d\n", data->name, i);
        sleep(time_sleeping);
    }

    pthread_mutex_lock(&mutex);
    printf("%s a fini de compter jusqu'a : %d en position %d\n", data->name,
data->n, rank);
    rank += 1;
    pthread_mutex_unlock(&mutex);
}
```

```
    pthread_exit(NULL);
}

int main(int argc, char** argv) {
    int init = pthread_mutex_init(&mutex, NULL);
    if (init != 0) {
        printf("%s\n", "Création de mutex impossible");
        exit(0);
    }

    pthread_t counter_1, counter_2, counter_3;
    char* counter_1_name = "counter 1";
    char* counter_2_name = "counter 2";
    char* counter_3_name = "counter 3";

    srand(time(NULL));

    DATA one = {counter_1_name, 5};
    DATA two = {counter_2_name, 5};
    DATA three = {counter_3_name, 5};

    pthread_create(&counter_1, NULL, counting, (void*) &one);
    pthread_create(&counter_2, NULL, counting, (void*) &two);
    pthread_create(&counter_3, NULL, counting, (void*) &three);

    pthread_join(counter_1, NULL);
    pthread_join(counter_2, NULL);
    pthread_join(counter_3, NULL);

    pthread_mutex_destroy(&mutex);

    return (EXIT_SUCCESS);
}
```

Trace d'exécution :

Pour afficher l'ordre d'arrivée de chaque compteur, nous devons faire appel à un mutex pour éviter le cas où deux compteurs souhaiteraient accéder en même temps à la même ressource et obtenir le même numéro d'arrivée.

```
counter 1 : 0
counter 2 : 0
counter 3 : 0
counter 1 : 1
counter 2 : 1
counter 3 : 1
counter 2 : 2
counter 3 : 2
counter 1 : 2
counter 2 : 3
counter 1 : 3
counter 3 : 3
counter 2 : 4
counter 1 : 4
counter 3 : 4
counter 2 a fini de compter jusqu'a : 5 en position 1
counter 1 a fini de compter jusqu'a : 5 en position 2
counter 3 a fini de compter jusqu'a : 5 en position 3
```

Exercice 3.3 :

Dans ce dernier exercice, le but est de réaliser un produit de matrice carrées de deux façons. La première en lançant un seul thread qui se charge de la multiplication complète de deux matrices de 1000 par 1000. La seconde méthode consiste à découper les deux matrices en 4 et à laisser 4 threads différents se charger la multiplication des 8 sous matrices entre elles. On regarde alors le temps mis par chacune des deux méthodes pour faire la même opération.

Code source :

Header.h :

```
#ifndef HEADER_H
#define HEADER_H

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <unistd.h>

#define NBTHREAD 4
#define MAXVAL 9999
#define MAT_SIZE 1000    //La matrice doit etre de taille paire

typedef struct thread_Args {
    int rowFrom;
    int rowTo;
    double** matrix_1;
    double** matrix_2;
    double** resMat;
    unsigned int rows_1;
    unsigned int columns_1;
    unsigned int rows_2;
    unsigned int columns_2;
} THREAD_ARGS;

#endif // HEADER_H
```

Main.c :

```
#include "headers/header.h"

double** alloc_matrix(unsigned int rows, unsigned int columns) {
    double** matrix = (double**) calloc(rows, sizeof (double*));
    if (!matrix) {
        return NULL;
    }

    for (int i = 0; i < rows; i++) {
```

```
        matrix[i] = (double*) calloc(columns, sizeof (double));
    }

    return matrix;
}

void free_matrix(double** matrix) {
    free(matrix[0]);
    free(matrix);
}

void random_matrix(double*** matrix, unsigned int rows, unsigned int columns)
{
    for (int y = 0; y < rows; y++) {
        for (int x = 0; x < columns; x++) {
            (*matrix)[y][x] = rand() % MAXVAL;
        }
    }
}

void* product_thread(void* arg) {
    THREAD_ARGS* argsThread = (THREAD_ARGS*) arg;
    for (int y = argsThread->rowFrom; y < argsThread->rowTo; y++) {
        for (int x = 0; x < argsThread->columns_2; x++) {
            for (int k = 0; k < argsThread->rows_2; k++) {
                argsThread->resMat[y][x] += argsThread->matrix_1[y][k] *
argsThread->matrix_2[k][x];
            }
        }
    }

    return NULL;
}

double** product(double** matrix_1, unsigned int rows_1, unsigned int
columns_1, double** matrix_2, unsigned int rows_2, unsigned int columns_2) {
    double** resMat = alloc_matrix(columns_2, rows_1);

    for (int y = 0; y < rows_1; y++) {
        for (int x = 0; x < columns_2; x++) {
            for (int k = 0; k < rows_2; k++) {
                resMat[y][x] += matrix_1[y][k] * matrix_2[k][x];
            }
        }
    }

    return resMat;
}

void print_matrix(double** matrix, unsigned int rows, unsigned int columns)
{
    for (int y = 0; y < rows; y++) {
        for (int x = 0; x < columns; x++) {
```

```
        printf("%.3f | ", matrix[y][x]);
    }
    printf("\n");
}

}

int main(int argc, char** argv) {
    int resThread[NBTHREAD];
    pthread_t threadTab[NBTHREAD];
    void* threadResults[NBTHREAD];
    THREAD_ARGS argsTab[NBTHREAD];
    clock_t startClock;
    clock_t endClock;
    time_t startTime;
    time_t endTime;
    int spaceLength;

    double** matrix_1 = alloc_matrix(MAT_SIZE, MAT_SIZE);
    double** matrix_2 = alloc_matrix(MAT_SIZE, MAT_SIZE);
    double** resMat = alloc_matrix(MAT_SIZE, MAT_SIZE);

    srand(time(NULL));
    random_matrix(&matrix_1, MAT_SIZE, MAT_SIZE);
    random_matrix(&matrix_2, MAT_SIZE, MAT_SIZE);

    printf("Produit matriciel classique : \n");
    startClock = clock();
    startTime = time(NULL);

    resMat = product(matrix_1, MAT_SIZE, MAT_SIZE, matrix_2, MAT_SIZE,
MAT_SIZE);

    endClock = clock();
    endTime = time(NULL);

    printf("Clock : Avec un seul thread il faut : %f secondes\n", ((double)
(endClock - startClock) / CLOCKS_PER_SEC));
    printf("Time : Avec un seul thread il faut : %f secondes\n\n",
difftime(endTime, startTime));

    //Séparation du calcul
    spaceLength = MAT_SIZE / NBTHREAD;

    printf("Produit matriciel multithreadé : \n");
    startClock = clock();
    startTime = time(NULL);

    //Parametrage des threads
    for (int i = 0; i < NBTHREAD; i++) {
        argsTab[i].rowFrom = i * spaceLength;
        argsTab[i].rowTo = (i + 1) * spaceLength;

        argsTab[i].matrix_1 = matrix_1;
```



```
        argsTab[i].rows_1 = MAT_SIZE;
        argsTab[i].columns_1 = MAT_SIZE;

        argsTab[i].matrix_2 = matrix_2;
        argsTab[i].rows_2 = MAT_SIZE;
        argsTab[i].columns_2 = MAT_SIZE;

        argsTab[i].resMat = resMat;
    }

    //Lancement des threads
    for (int i = 0; i < NBTHREAD; i++) {
        resThread[i] = pthread_create(&threadTab[i], NULL, product_thread,
&argsTab[i]);
    }

    //Attente de la fin des threads
    for (int i = 0; i < NBTHREAD; i++) {
        pthread_join(threadTab[i], &threadResults[i]);
    }

    endClock = clock();
    endTime = time(NULL);

    printf("Clock : Avec %d threads il faut : %f secondes\n", NBTHREAD,
((double) (endClock - startClock) / CLOCKS_PER_SEC));
    printf("Time : Avec %d threads il faut : %f secondes\n", NBTHREAD,
difftime(endTime, startTime));

    free_matrix(matrix_1);
    free_matrix(matrix_2);
    free_matrix(resMat);

    return EXIT_SUCCESS;
}
```

Trace d'exécution :

Voici le résultat obtenu :

Produit matriciel classique :

Clock : Avec un seul thread il faut : 13.672880 secondes

Time : Avec un seul thread il faut : 14.000000 secondes

Produit matriciel multithreadé :

Clock : Avec 4 threads il faut : 28.433513 secondes

Time : Avec 4 threads il faut : 7.000000 secondes

On constate effectivement une diminution du temps de calcul ainsi qu'un meilleur usage des CPU multicoeurs / multithread. Ce gain de temps n'est cependant significatif que si les matrices à multiplier sont grandes. Dans notre cas, 1000 par 1000.