EXO 3:

- Semaphore.h code:

```c
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int create_sm(int nsems, key_t key) {
    int semid = semget(key, nsems, IPC_CREAT | IPC_EXCL | 0666);
    if (semid < 0) {
        semid = semget(key, nsems, 0);
        printf("Semaphor group already exists with id (%d)\n", semid);
    } else
        printf("New semaphore group created with id (%d)\n", semid);
    return semid;
}

void init_sm(int semid, int semnum, int val) {
        if ( semctl(semid, semnum, SETVAL, val)< 0) {
        perror("Init semctl failed\n");
    } else {
        printf("Init semctl successful\n");
    }
}

void p(int semid, int semnum) {
    struct sembuf sb;
    sb.sem_num = semnum;
    sb.sem_op = -1;
    sb.sem_flg = SEM_UNDO;
    if (semop(semid, &sb, 1) < 0) {
        perror("P semop failed\n");
    } else {
        printf("P semop executed successfully\n");
    }
}

void v(int semid, int semnum) {
    struct sembuf sb;
    sb.sem_num = semnum;
    sb.sem_op = 1;
    sb.sem_flg = 0;
    if (semop(semid, &sb, 1) < 0) {
        perror("V semop failed\n");
    } else {
        printf("V semop executed successfully\n");
    }
}
```

```c
void z(int semid, int semnum) {
    struct sembuf sb;
    sb.sem_num = semnum;
    sb.sem_op = 0;
    sb.sem_flg = 0;
    if (semop(semid, &sb, 1) < 0) {
        perror("Z semop failed\n");
    } else {
        printf("Z semop executed successfully\n");
    }
}

void sem_destroy(int semid) {
    if (semctl(semid, 0, IPC_RMID) < 0) {
        perror("semctl IPC_RMID failed\n");
    } else {
        printf("Semaphore set removed successfully.\n");
    }
}
```

- Program code:

```c
#include "semaphore.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>

void main() {
  key_t key = ftok("./exo3.c", 'a');
  if (key == -1) {
    perror("Ftork failed");
    exit(EXIT_FAILURE);
  }
  int semid = create_sm(1, key);
  init_sm(semid, 0, 1);

  srand(time(NULL));

  int i, n;
  printf("Give number of child processes: \n");
  scanf("%d", &n);
  int pids[n];

  for (i = 0; i < n; i++) {
    pids[i] = fork();
    if (pids[i] < 0) {
      perror("Fork failed");
    } else if (pids[i] == 0) {

      printf("Process %d with id (%d): Waiting\n", i, getpid());
      p(semid, 0);

      printf("Process %d with id (%d): Printing\n", i, getpid());
      time_t start_time = time(NULL);

      sleep(1 + ((double)rand() / RAND_MAX) * (3 - 1));

      time_t end_time = time(NULL);
      printf("Process %d with id (%d): Finished in %ds\n", i, getpid(),
             end_time - start_time);

      v(semid, 0);
      exit(EXIT_SUCCESS);
    }
  }

  for (i = 0; i < n; i++) {
    wait(NULL);
  }
  printf("Parent process with id (%d): All children finished\n", getpid());
  sem_destroy(semid);
}
```

- Execution output:

```
┌─repo) ~/Desktop/M1/SE/TP
└─cmd──) ./exo3
New semaphore group created with id (10)
Init semctl successful
Give number of child processes:
9
Process 0 with id (8197): Waiting
P semop executed successfully
Process 0 with id (8197): Printing
Process 1 with id (8198): Waiting
Process 3 with id (8200): Waiting
Process 2 with id (8199): Waiting
Process 4 with id (8201): Waiting
Process 5 with id (8202): Waiting
Process 6 with id (8203): Waiting
Process 7 with id (8204): Waiting
Process 8 with id (8205): Waiting
Process 0 with id (8197): Finished in 1s
V semop executed successfully
P semop executed successfully
Process 3 with id (8200): Printing
P semop executed successfully
Process 1 with id (8198): Printing
Process 3 with id (8200): Finished in 1s
P semop executed successfully
Process 2 with id (8199): Printing
V semop executed successfully
P semop executed successfully
Process 4 with id (8201): Printing
Process 1 with id (8198): Finished in 1s
P semop executed successfully
Process 5 with id (8202): Printing
V semop executed successfully
P semop executed successfully
Process 6 with id (8203): Printing
Process 2 with id (8199): Finished in 1s
V semop executed successfully
P semop executed successfully
Process 7 with id (8204): Printing
Process 4 with id (8201): Finished in 1s
V semop executed successfully
P semop executed successfully
Process 8 with id (8205): Printing
Process 5 with id (8202): Finished in 1s
V semop executed successfully
Process 6 with id (8203): Finished in 1s
V semop executed successfully
Process 7 with id (8204): Finished in 1s
V semop executed successfully
Process 8 with id (8205): Finished in 1s
V semop executed successfully
Parent process with id (8196): All children finished
Semaphore set removed successfully.
```

- Question:
  Si l'un des processus est terminé accidentellement (par exemple, en envoyant le signal `SIGKILL` avec la commande `kill -9 <pid>`) alors qu'il utilise la ressource partagée, la valeur du sémaphore peut rester décrémentée, verrouillant ainsi la ressource indéfiniment. Cela se produit parce que l'opération P (appel à `semop` avec `sem_op = -1`) décrémente le sémaphore, mais l'opération correspondante V (avec `sem_op = 1`) n'est jamais appelée en raison de la terminaison prématurée du processus.
  Pour gérer cette situation, utilisez le drapeau `SEM_UNDO` dans l'opération P (`semop`). Ce drapeau garantit que toute décrémentation effectuée sur le sémaphore par un processus est automatiquement annulée par le système d'exploitation si le processus se termine de manière inattendue.

Exo 4:

- Create.c code:

```c
#include "semaphore.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

typedef struct data {
  int indxp;
  int indxc;
  int tab[10];
} sdata;

int main() {

  key_t key = ftok(".", 'a');
  int shmid = shmget(key, sizeof(sdata), IPC_CREAT | IPC_EXCL | 0666);

  if (shmid < 0) { // la zone existe deja
    shmid = shmget(key, sizeof(sdata), 0);
    printf("Memory segment already exists with ID (%d)\n", shmid);
  } else {
    printf("New memory segment created with ID (%d)\n", shmid);
  }

  sdata *sd = NULL;
  sd = shmat(shmid, sd, 0);
  // maintenant on peut ecrire dans la zone via la structure
  sd->indxc = 0;
  sd->indxp = 0;

  printf("Indexes written (P: %d, C: %d)\n", sd->indxp, sd->indxc);
  int semid = create_sm(4, key);
  init_sm(semid, 0, 10);
  init_sm(semid, 1, 0);
  init_sm(semid, 2, 1);
  init_sm(semid, 3, 1);
  return 0;
}
```

- Consumer.c code:

```c
#include "semaphore.h"
#include <stdlib.h>
#include <sys/shm.h>
#include <unistd.h>

typedef struct data {
  int indxp;
  int indxc;
  int tab[10];
} sdata;

int main() {

  key_t key = ftok(".", 'a');
  int shmid = shmget(key, sizeof(sdata), IPC_CREAT | IPC_EXCL | 0666);

  if (shmid < 0) {
    shmid = shmget(key, sizeof(sdata), 0);
    printf("Memory segment already exists with ID (%d)\n", shmid);
  } else {
    printf("New memory segment created with ID (%d)\n", shmid);
  }

  sdata *sd = NULL;
  sd = shmat(shmid, sd, 0);

  int semid = create_sm(4, key);

  int val;
  while (1) {
    if (sd->tab[sd->indxc] == 0) {
      continue;
    }
    p(semid, 1);
    p(semid, 3);
    val = sd->tab[sd->indxc];
    sd->tab[sd->indxc] = 0;
    sd->indxc = (sd->indxc + 1) % 10;
    v(semid, 3);
    printf("Consuming value %d, at index %d\n", val, sd->indxc);
    printf("Tab=[%d,%d,%d,%d,%d,%d,%d,%d,%d,%d]\n", sd->tab[0], sd->tab[1],
           sd->tab[2], sd->tab[3], sd->tab[4], sd->tab[5], sd->tab[6],
           sd->tab[7], sd->tab[8], sd->tab[9]);
    v(semid, 0);
    sleep(3);
  }
  return 0;
}
```

- Producer.c code:

```c
#include "semaphore.h"
#include <stdlib.h>
#include <sys/shm.h>
#include <unistd.h>

typedef struct data {
  int indxp;
  int indxc;
  int tab[10];
} sdata;
int main() {

  key_t key = ftok(".", 'a');
  int shmid = shmget(key, sizeof(sdata), IPC_CREAT | IPC_EXCL | 0666);

  if (shmid < 0) {
    shmid = shmget(key, sizeof(sdata), 0);
    printf("Memory segment already exists with ID (%d)\n", shmid);
  } else {
    printf("New memory segment created with ID (%d)\n", shmid);
  }

  sdata *sd = NULL;
  sd = shmat(shmid, sd, 0);

  int semid = create_sm(4, key);
  int val;
  while (1) {
    if (sd->tab[sd->indxp] != 0) {
      continue;
    }
    val = rand() % 10;
    printf("Produced value (%d)\n", val);
    p(semid, 0);
    p(semid, 2);
    sd->tab[sd->indxp] = val;
    sd->indxp = (sd->indxp + 1) % 10;
    v(semid, 2);
    printf("tab =[%d,%d,%d,%d,%d,%d,%d,%d,%d,%d]\n", sd->tab[0], sd->tab[1],
           sd->tab[2], sd->tab[3], sd->tab[4], sd->tab[5], sd->tab[6],
           sd->tab[7], sd->tab[8], sd->tab[9]);
    v(semid, 1);
    sleep(3);
  }
  return 0;
}
```

- Execution output:
  - create.c:

```
┌─repo) ~/Desktop/M1/SE/TP/exo4
└─cmd──) ./create
New memory segment created with ID (229415)
Indexes written (P: 0, C: 0)
New semaphore group created with id (19)
Init semctl successful
Init semctl successful
Init semctl successful
Init semctl successful
```

- 2 Producer.c + 2 Consumers.c:

- Question:
  1. **Si le producteur est arrêté** : Le consommateur pourra accéder à la mémoire partagée et vérifier le tampon (`tab`) pour consommer des valeurs. Cependant, puisque le producteur ne produit plus de valeurs, le consommateur trouvera constamment le tampon vide (les valeurs resteront à 0). Le sémaphore `empty_slots` (sémaphore 0) restera inchangé car le producteur ne met pas à jour le tampon. Ce sémaphore devrait être dans un état où le consommateur attend que le sémaphore `empty_slots` soit signalé par le producteur, mais comme aucune valeur n'est produite, le consommateur attendra indéfiniment.
  2. **Si le consommateur est arrêté** : Le producteur continuera à produire des valeurs et à les placer dans le tampon, en mettant à jour `tab` et en signalant le sémaphore `full_slots` (sémaphore 1) pour indiquer que des données sont disponibles pour la consommation. Cependant, comme le consommateur ne consomme pas les éléments, le sémaphore `full_slots` continuera à augmenter, mais le sémaphore `empty_slots` du consommateur ne sera pas libéré. Cela peut amener le producteur à se bloquer ou à ne pas pouvoir insérer de nouvelles valeurs une fois que le tampon est plein, car le producteur attend de l'espace dans le tampon (`empty_slots`).