CSE344 - System Programming - Homework #2

Fatih Kaan Salgır - 171044009

1 Design Explanation

1. First process will create the shared memory, and all of them are going open it.
2. The process will create the fifo. This fifo will be the fifo that the process reads. If the process has a potato, it will register to shared memory with the pid. And wait for creation of all fifos.
3. All of the fifos are going to be opened. 1 one of them to read, other ones to write.
4. If process has a potato it will write its pid to a random fifo.
5. The process will read its fifo until it reads either a potato id or a termination message (integer, defined as -2) when all potatoes are cooled down.
6. It will gets the potato with potato id, increments its number of switch, checks if cooled down. If its the case, it will let others know and terminate. Otherwise it will write again to some random fifo.

- Maximum number of fifo to handle is defined as `N` (64).
- Shared memory size defined as `SHM_SIZE` (1024).

1.1 Shared Memory Initilization

First `shm_open` is called with both `O_CREAT` and `O_EXCL` flags. As the man page of the `shm_open` states, if the shared memory already exist, it will return -1 and `errno` will set to `EEXIST`. If shared memory already exists then, simply `mmap` can be used to establist the mapping address. If shared memory is created, some initialization steps needed:

- `ftruncate` will be called to set its size.
- Semaphore in the shared memory will be set to 0.
- number of created files will be set to 0.
- number of potatoes will be set to 0.

Shared memory initialization is protected by using named semaphore.

1.2 Shared Memory Design

`struct shared_memory` is used in order to organize shared memory.

`all fifos are created`	`[sem _t]`
`number of created fifos`	`[int]`
`size of potatoes`	`[int]`
`#1 potato`	`[struct{int, int, int}]`
`#1 id`	`[int]`
`#1 switch count`	`[int]`
`#1 number of switches`	`[int]`
`#2 potato`	`[struct{int, int, int}]`
`#2 id`	`[int]`
`#2 switch count`	`[int]`
`#2 number of switches`	`[int]`
...	

1.3 Waiting for FIFO Creation

1. A process is going to create their fifo and wait for other process using `sem_wait()`. This semaphore is different from the one given as command-line arguement. It lies in the shared memory, and inital value is 0. (`all_fifos_created`)
2. Processes will create the fifos one by one, to avoid race condition. The synchronization technique is used here is named semaphore.
3. If a process has been reached to `EOF`, so this process is the last one. This means all other processes are ready to continue. It will increment to value of `sem_post()` N times (number of processes).

1.4 Handling SIGINT

In case of receiving `SIGINT`, processs will stop switching potatoes. Let the others know that it has received `SIGINT` via fifo. As a result all processes will be terminated. And give all the resources (shared memory, semaphore, open files) back to the system. If a process receives `SIGINT` before it has a chance to open the fifos, in this case it simply exits.

2 Running & Test Cases

Program tested with different cases and also with the valgrind to check memory leaks. According the output of the valgrind, "All heap blocks were freed – no leaks are possible".

```
valgrind --leak-check=full \
         --show-leak-kinds=all \
         --track-origins=yes \
         --verbose \
         --log-file=valgrind-out.txt \
         ...
```

Running with 4 processes and 3 potatoes:

```
./player -b 5 -f fifonames -m s1 -s sharedmem &
./player -b 0 -f fifonames -m s1 -s sharedmem &
./player -b 8 -f fifonames -m s1 -s sharedmem &
./player -b 3 -f fifonames -m s1 -s sharedmem &
```

```
pid=2726573 sending potato number 2726573 to fifos/fifo3; this is switch number 1 / 3
pid=2726572 sending potato number 2726572 to fifos/fifo4; this is switch number 1 / 8
pid=2726572 receiving potato number 2726573 from fifos/fifo3
pid=2726572 sending potato number 2726573 to fifos/fifo2; this is switch number 2 / 3
pid=2726570 sending potato number 2726570 to fifos/fifo3; this is switch number 1 / 5
pid=2726572 receiving potato number 2726570 from fifos/fifo3
pid=2726572 sending potato number 2726570 to fifos/fifo4; this is switch number 2 / 5
pid=2726571; potato number 2726573 has cooled down
pid=2726573 receiving potato number 2726572 from fifos/fifo4
pid=2726573 sending potato number 2726572 to fifos/fifo2; this is switch number 2 / 8
pid=2726573 receiving potato number 2726570 from fifos/fifo4
pid=2726573 sending potato number 2726570 to fifos/fifo1; this is switch number 3 / 5
pid=2726571 receiving potato number 2726572 from fifos/fifo2
pid=2726571 sending potato number 2726572 to fifos/fifo3; this is switch number 3 / 8
pid=2726572 receiving potato number 2726572 from fifos/fifo3
pid=2726572 sending potato number 2726572 to fifos/fifo4; this is switch number 4 / 8
pid=2726573 receiving potato number 2726572 from fifos/fifo4
pid=2726573 sending potato number 2726572 to fifos/fifo3; this is switch number 5 / 8
pid=2726572 receiving potato number 2726572 from fifos/fifo3
pid=2726572 sending potato number 2726572 to fifos/fifo4; this is switch number 6 / 8
pid=2726573 receiving potato number 2726572 from fifos/fifo4
pid=2726573 sending potato number 2726572 to fifos/fifo1; this is switch number 7 / 8
pid=2726570 receiving potato number 2726570 from fifos/fifo1
pid=2726570 sending potato number 2726570 to fifos/fifo2; this is switch number 4 / 5
pid=2726570; potato number 2726572 has cooled down
pid=2726571; potato number 2726570 has cooled down
```