CSE344 – System Programming
HW#2 – Report

Berk Pekgöz
171044041

Solving The Problem & Design:

To solve problem, I divided design to these parts:

- Parsing command line arguments
- Reading fifonames file
- Attaching process to shared memory and FIFO creation
- Playing game with FIFOs
- Game Finish
- Error/Interrupt Handling

Parsing command line arguments

To parse command line arguments getopt() is used. For each option counter will be increased and at the and if counter is not 4 (needed arguments) error occurs. Also checking for valid integer to get correct haspotatoornot. If there is unknown argument, also error occur.

Reading fifonames file

To read fifonames read_fifonames() function created. This function fills the 2D fifonames array. Size of this 2D array is defined, because Mr.Aptoula said *"I suggest you make a reasonably large initial allocation"* on moodle forum. If user wants bigger N or larger path name, user can change the #define which is on top of the code. All error checks are done for reading.

Attaching process to shared memory and FIFO creation

To solve this problem some synchronization needed. Before the entering this function named semaphore is used for sem_wait(). Because we will open the shared memory and we will make some changes on shared memory. After the function sem_post() used to let other process to acces shared memory.

When process enter this function, first opens shared memory. Then checks size of the shared memory with fstat(). If size is 0 which means shared memory is not initialized, that process initialize shared memory and mmap() is used to attach (Size of shared memory is fixed with pre-defined array sizes because . Mr.Aptoula said *"I suggest you make a* *reasonably large initial allocation"* on moodle forum). If size is equal to size of shared memory struct, then process uses mmap() to attach itself to shared memory.

After that, process creates FIFOf file with mkfifo(). Each process creates different FIFO form fifonames (shared memory is used the get unique selections). Then opens its own FIFO as read only and nonblock. Nonblock flag is used because otherwise process would be blocked, but this read end is dummy. After that process opens same fifo as read only without nonblock to read messages from other processes. Process also opens its own fifo as write only to prevent EOF (this dummy strategy is shown in lecture).

Finally, makes some updates about potato on shared memory.

Also, synchronization barrier is needed to start game simultaneously. To solve synchronization barrier, unnamed semaphore(semaphore is in shared memory) is used. This solution is shown in lecture on week 8.

Playing game with FIFOs

When game started, all processe enters a loop. If process does not have potato currently, reads its own FIFO and waits message. If message arrives and message has valid potato id, then process updates shared memory (decrease switch left count). And changes its current situation as "has_potato".

If process has a potato, process choose one fifo randomly and send the potato as message (open-write-close). And changes its current situation as "has_potato = FALSE".

Game Finish

If potato counter in the shared memory becomes 0, it means there is no potato left and game is over. If one of the processes takes last potato and this last switch of that potato, that process sends other FIFOs a message. The message is a integer with value –1. When –1 comes as message to processes, process understand that this is not a potato. This is termination message. Then all processes terminate. Sender process also unlinks shared memory.

Error/Interrupt Handling

If error or interrupt occured in process. , that process sends other FIFOs a message. The message is a integer with value –2. When –2 comes as message to processes, process understand that this is not a potato. This is termination message with "fail". Then all processes terminate. Sender process also unlinks shared memory.

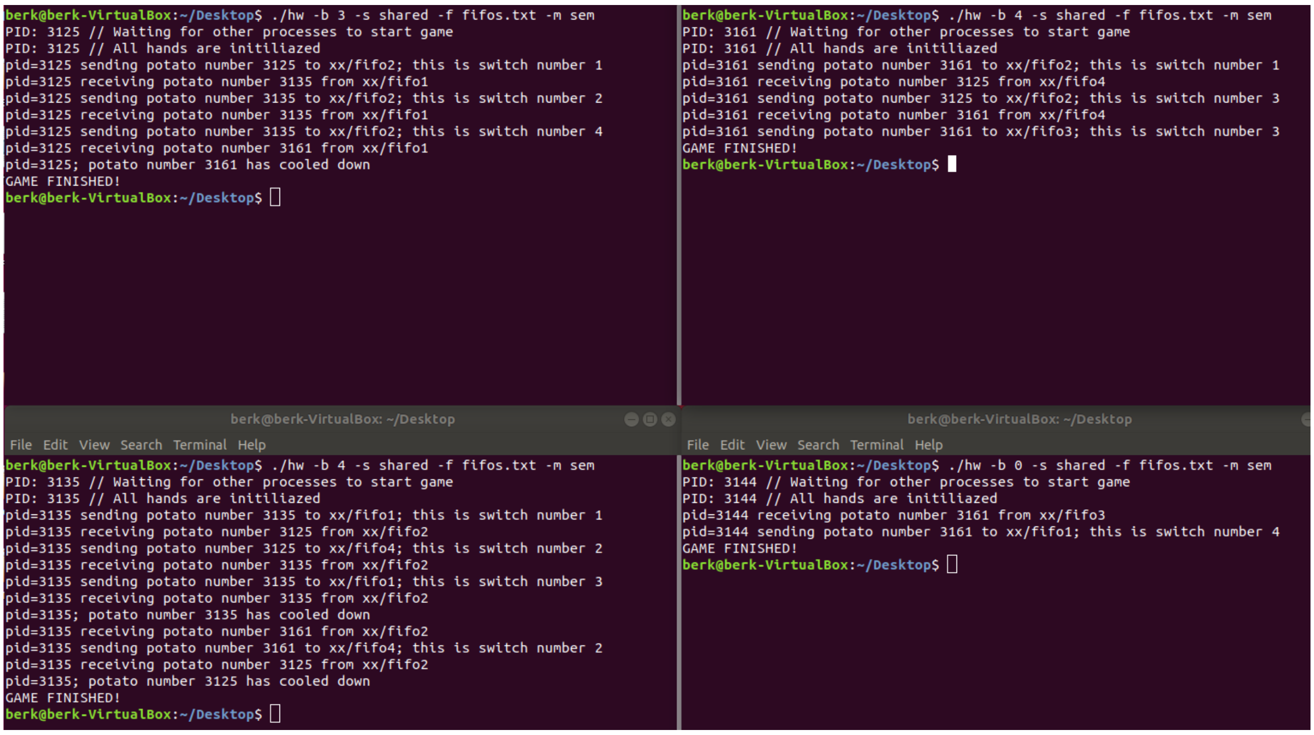
If user interrupt occur before game start, then only that process terminates.

Note: All requirements mentioned in pdf file are achieved.

Notices

- If FIFO files are already exist. Program will give an error. Please use not created FIFO name.
- Input file must have same number of lines with number of FIFOs.
- Size of arrays is pre-defined in the code because Mr.Aptoula said *"I suggest you make a reasonably large initial allocation"* on moodle forum. If user wants bigger N or larger path name, user can change the #define which is on top of the code.
- Permissions of the files are user only. Please consider this.
- Processes cleans up all resources when termination message arrive.
- In case of an arbitrary error, exited by printing to stderr a nicely formatted informative message.
- Some function calls does not have error check because they have no valid error in our processes or checking is not needed.
- In case of CTRL-C the program stop execution and sends other processes message to terminate (after the games start).
- In case of CTRL-C before game start. Process terminates.

Example Run



```
berk@berk-VirtualBox:~/Desktop$ ./hw -b 3 -s shared -f fifos.txt -m sem
PID: 3125 // Waiting for other processes to start game
PID: 3125 // All hands are initiliazed
pid=3125 sending potato number 3125 to xx/fifo2; this is switch number 1
pid=3125 receiving potato number 3135 from xx/fifo1
pid=3125 sending potato number 3135 to xx/fifo2; this is switch number 2
pid=3125 receiving potato number 3135 from xx/fifo1
pid=3125 sending potato number 3135 to xx/fifo2; this is switch number 4
pid=3125 receiving potato number 3161 from xx/fifo1
pid=3125; potato number 3161 has cooled down
GAME FINISHED!
berk@berk-VirtualBox:~/Desktop$
```

```
berk@berk-VirtualBox:~/Desktop$ ./hw -b 4 -s shared -f fifos.txt -m sem
PID: 3161 // Waiting for other processes to start game
PID: 3161 // All hands are initiliazed
pid=3161 sending potato number 3161 to xx/fifo2; this is switch number 1
pid=3161 receiving potato number 3125 from xx/fifo4
pid=3161 sending potato number 3125 to xx/fifo2; this is switch number 3
pid=3161 receiving potato number 3161 from xx/fifo4
pid=3161 sending potato number 3161 to xx/fifo3; this is switch number 3
GAME FINISHED!
berk@berk-VirtualBox:~/Desktop$
```

berk@berk-VirtualBox: ~/Desktop
File Edit View Search Terminal Help

```
berk@berk-VirtualBox:~/Desktop$ ./hw -b 4 -s shared -f fifos.txt -m sem
PID: 3135 // Waiting for other processes to start game
PID: 3135 // All hands are initiliazed
pid=3135 sending potato number 3135 to xx/fifo1; this is switch number 1
pid=3135 receiving potato number 3125 from xx/fifo2
pid=3135 sending potato number 3125 to xx/fifo4; this is switch number 2
pid=3135 receiving potato number 3135 from xx/fifo2
pid=3135 sending potato number 3135 to xx/fifo1; this is switch number 3
pid=3135 receiving potato number 3135 from xx/fifo2
pid=3135; potato number 3135 has cooled down
pid=3135 receiving potato number 3161 from xx/fifo2
pid=3135 sending potato number 3161 to xx/fifo4; this is switch number 2
pid=3135 receiving potato number 3125 from xx/fifo2
pid=3135; potato number 3125 has cooled down
GAME FINISHED!
berk@berk-VirtualBox:~/Desktop$
```

berk@berk-VirtualBox: ~/Desktop
File Edit View Search Terminal Help

```
berk@berk-VirtualBox:~/Desktop$ ./hw -b 0 -s shared -f fifos.txt -m sem
PID: 3144 // Waiting for other processes to start game
PID: 3144 // All hands are initiliazed
pid=3144 receiving potato number 3161 from xx/fifo3
pid=3144 sending potato number 3161 to xx/fifo1; this is switch number 4
GAME FINISHED!
berk@berk-VirtualBox:~/Desktop$
```