System Programming HW2

Zafer ALTAY 161044063

April 2021

1 Problems and Solutions

In homework, I first started by creating a struct for shared memory.Because I wanted to make my job easier because I was going to hold more than one thing. Inside this structure are arrays that hold potato ids, fifo filenames, and switch numbers.I also have an integer value that keeps the number of potatoes that have not cooled. Thanks to this variable, I will explain the details of the process that exit instead of waiting when it checks this number.

Then I parsed the values taken from the commandline using getopt.I created a semaphore and shared memory using the values I parsed.These values will be tried to be created in each newly created process, if not, they will be created, if any, previously created ones will be used.

Then I read the file where the fifos are kept, split it by the ' n' character and keep it in a 2-dimensional character array, so it is now easier to access the whole fifo list.Now we are here to create fifo. I created fifos using an mkfifo for all the names I got with a loop in a process.So I ensured that fifo is created for each file.

My next action is to save the current process to shared memory.Now let's get into some details for shared memory.I will use the arrays that I created in shared memory as parallel arrays. For example, an index x holds the details of a certain process in all arrays.Example; patatoId [3] $->$ 4th is the PID of the recorded transaction.PatatoSwitch [3] keeps track of how many changes the potato of this process will cool down. If a potato is sent to another process, this value is reduced.At the same time, fifosnames stores the name of the fifo file selected by this operation in parallel.

So how does it save itself? I overcame this problem as follows; I started to search from the first index of the array holding the ids in shared memory. If I see an index with a value of 0, I saved myself there.This index number is important to us because I saved other information in those indexes in parallel arrays.By the way, I definitely lock the semaphore while doing these save operations, because there is a race condition formation situation, this is something we do not want because the information can be mixed or maybe lost forever.Also during these saves, I choose a fifo file for myself and save it as well. So when the semaphore is off, because they should all be different from each other and not get involved when recording.

So how do I make sure they are all different?I overcame this problem as follows; I had the computer guess a random number and chose that number from the fifos I had previously saved in 2d sequences.Then I searched if that fifo is stored in shared memory. If it was registered, I selected and checked a random number again, but if it was not, I immediately saved it.I used this loop until I found an unregistered fifo.After doing all these recording operations, I opened the semaphore so that other processes can use it.

Now we can play our game. The biggest problem here is that the reading and writing part of a fifo file does not continue until it is opened.So I am creating the file that I will write, but if the reading part is not open, I get blocked, but the other process I expect to open the read part is waiting for opened another read part for write.If we are not very lucky, we live here a deadlock.So to get rid of this problem, I open the fifo file of each process for both read and write permissions. Then I choose a random number and send a request to write to that fifo from the fifo list.At this point, the process is waiting for the other end of the fifo to which it sends a request to open for read.When the fifo to which the request is sent is opened for both reading and writing, We will now be able to write.In other words, when all processes run, the read part of each fifo file is opened. Therefore, if blocked processes for write can continue.This eliminates our deadlock problem.

Now let's examine the details of the reading and writing processes, namely the gameplay of the game.As I just mentioned, the process randomly selects a fifo file. Then he sends the information to be written to him. So what is this information to be written? I keep the potatoes he holds in an array. He chooses one from there and sends it.So what's the potatoes he keeps? We start with 1 potato. The potatoes he kept are as follows; We may be starting with one potato, but a few of the processes may want to write to the same fifo file, we don't know that.Because we let them send it randomly. Here we cannot read only one potato and ignore the others. If we do this, the program will result in lost potatoes. For this, I keep the potatoes we read in an array where we keep the potatoes in our hands, I will explain the detailed storage in the read section.Now we are in the write part, I will explain which potato and how I send it.I do this like this, I select the potato in the zeroth index of the array, type in the fifo I randomly picked, and then shift the potatoes to the left.So the potato I sent has been deleted from the array.Then the write part is finished. In the meantime, while doing the write part and the part of deleting potatoes from the series, I lock the semaphore to avoid any race condition.When the writing part is finished, I open the semaphore back.

Now we have moved on to the reading process section, where I explain what I am doing.I do the reading and save it to a character array. Then I convert the string I read to a number because what I'm reading is the potato id.Then I check whether the number I read is in shared memory. Why? I was already typing id. Is there a need for this? Actually, I am bringing a solution to a problem here. I will explain this in a moment and I will not tell you now because we are going in order. I do not want to spoil the order.I checked if it is in memory. If the number I read is not in shared memory, I break out the loop and terminate the

process.If the number I read is in memory, I find the location of that number and decrease the number of switches in the parallel array.If this number drops to zero, the potato has cooled down and I'm writing this on the screen.Then, after the reading is done, I check all the potato counts. If they are all zeroes, the program will terminate, but there is a problem. When the problem realizes that this process has ended, there may be processes waiting to be finished in the read section. It needs to know that these will end. I send a special number to all fifos to understand that these processes will end.When the processes see that the number they read is not in memory, they will be out of the loop and finished. This is how i solved this problem.And then our process that makes these writes . It exits the loop. It closes and finishes the documents that are opened.

2 Pass File

In general, I cite the expectations and shortcomings for convenience:

First of all, I use shared memory,named semaphore and fifo pipes.

The program works completely as expected.

Define N is defined how many fifo files and operations will be in the assignment. If changes are made in the numbers, if this is edited, the program will run completely.

Ctrl + C control is missing actually I have a solution but I didn't do it because it is not a recommended method.In the Signal Handler, I could send a kill signal to all process ids in shared memory. But I couldn't because it would cause a problem if N numbers were too large. So I couldn't find a different solution.