LAB-4

Prahalad Vijaykumar
2003128

1)

OS LAB 04.pdf | 2003128_pipe_writer.c | 2003128_pipe_reader.c

Lab-4 > 2003128_pipe_reader.c

```c
#include<stdio.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <fcntl.h>


int main(){
    int readaccess;
    char *path="/tmp/mypipe";
    printf("Creating named pipe: /tmp/mypipe\n");
    mkfifo(path,0444);

    char *arr;
    arr=(char*)malloc(100);

    while(1){
        readaccess=open(path,O_RDONLY);
        arr=(char*)malloc(100);

        read(readaccess,arr,100);

        printf("Waiting for input...Got it: %s ",arr);
        if(strcmp("exit\n",arr)==0){
            printf("Exiting\n");
            break;
        }
        close(readaccess);
    }
    return 0;
}
```

Output:

Writer :

```
(base) MacBook-Pro:Lab-4 prahaladvijaykumar$ ./a.out
Opening named pipe: /tmp/mypipe
Enter Input:hello
Writing buffer to pipe...done
Enter Input:how are you
Writing buffer to pipe...done
Enter Input:im fine
Writing buffer to pipe...done
Enter Input:exit
Writing buffer to pipe...done
Exiting
(base) MacBook-Pro:Lab-4 prahaladvijaykumar$
```

Reader:

```
(base) MacBook-Pro:Lab-4 prahaladvijaykumar$ ./a.out
Creating named pipe: /tmp/mypipe
Waiting for input...Got it: hello
 Waiting for input...Got it: how are you
 Waiting for input...Got it: im fine
 Waiting for input...Got it: exit
 Exiting
(base) MacBook-Pro:Lab-4 prahaladvijaykumar$
```

2)(i)Observed 2 different outputs the first case is when the child creates a process exactly or nearly to the timing of creation of the parent process .before the shared memory data is updated to 1 . in that case the child process and parent process prints 0 .

```
(base) MacBook-Pro:Lab-4 prahaladvijaykumar$ ./a.out
The parent process begins.
Parent's report: current index =  0
The child process begins.
Child's report:  current value =  0
Parent's report: current index =  1
Child's report:  current value =  1
Parent's report: current index =  2
Child's report:  current value =  4
Parent's report: current index =  3
Child's report:  current value =  9
Parent's report: current index =  4
Child's report:  current value = 16
Parent's report: current index =  5
Child's report:  current value = 25
Parent's report: current index =  6
Child's report:  current value = 36
Parent's report: current index =  7
Child's report:  current value = 49
Parent's report: current index =  8
Child's report:  current value = 64
Parent's report: current index =  9
Child's report:  current value = 81
The child is done
The parent is done
(base) MacBook-Pro:Lab-4 prahaladvijaykumar$ gcc Lab4_6
```

(ii)When the child process is created after a few ms delay . the parent process at that case a 0 will be missed out by the child this is because before the child process can access the shared memory .the value would be updated by the parent to 1 so one of them is missed out.

```
(base) MacBook-Pro:Lab-4 prahaladvijaykumar$ gcc Lab4_6
.c
(base) MacBook-Pro:Lab-4 prahaladvijaykumar$ ./a.out
The parent process begins.
Parent's report: current index =  0
The child process begins.
Parent's report: current index =  1
Child's report:  current value =  1
Child's report:  current value =  1
Parent's report: current index =  2
Child's report:  current value =  4
Parent's report: current index =  3
Child's report:  current value =  9
Parent's report: current index =  4
Child's report:  current value = 16
Parent's report: current index =  5
Child's report:  current value = 25
Parent's report: current index =  6
Child's report:  current value = 36
Parent's report: current index =  7
Child's report:  current value = 49
Parent's report: current index =  8
Child's report:  current value = 64
Parent's report: current index =  9
Child's report:  current value = 81
The child is done
The parent is done
```

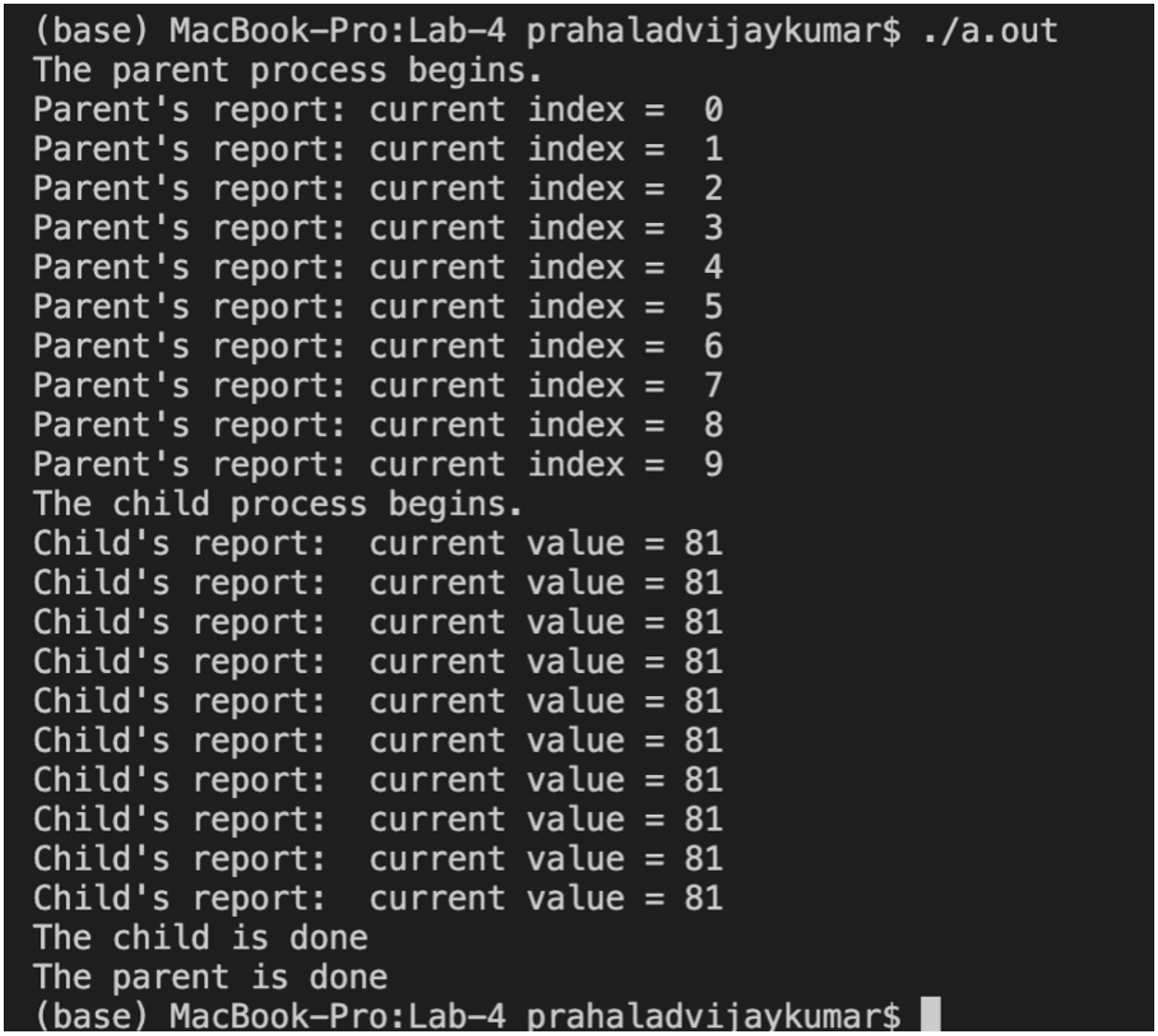
3)

```
(base) MacBook-Pro:Lab-4 prahaladvijaykumar$ ./a.out
The parent process begins.
Parent's report: current index =  0
The child process begins.
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
The child is done
Parent's report: current index =  1
Parent's report: current index =  2
Parent's report: current index =  3
Parent's report: current index =  4
Parent's report: current index =  5
Parent's report: current index =  6
Parent's report: current index =  7
Parent's report: current index =  8
Parent's report: current index =  9
The parent is done
```

Since the parent process is always waiting for 1sec after updating the value in the shared memory and the time taken by the child process for its creation will be less than a second and the execution of each calculation of square is surely gonna be less than each updation so each time the child process access the shared memory the value which will be extracted will be 0 so child reports only 0*0 = 0 .

4)

Here since the parent process doesn't have the sleep function so it can execute number from 0-9 in the shared memory in less than a seconds so now it only has to wait for its child process. And now the child process after waiting for 1s the value found in the shared memory will be 9 so everytime the child process will execute it will print 9*9 = 81



5) Spinlock is just like is yet another locking mechanism in which the writer targets to update the value in the shared memory only if the value in it is -1 otherwise place in an infinite loop until that happens

While on the readers side the infinite loop is run until the value in the shared memory is not -1 if not then takes the value breaks out of the loop and replaces the value in the memory block to -1 .

Output:

Here the parent process will first be created and the value initially placed in the shared memory is -1 so updates it to 0 and then child process reads the 0 and updates the shared memory is updated to -1 this keeps iteratively occurring until the index of parent reaches 9 and same for

the child. To make them wait for the update of the value they are put in a while(True) loop waiting until desired update has occurred.

```
(base) MacBook-Pro:Lab-4 prahaladvijaykumar$ gcc Lab4_6.c
(base) MacBook-Pro:Lab-4 prahaladvijaykumar$ ./a.out   The parent process begins.
Parent's report: current index =  0
The child process begins.
Child's report:  current value =  0
Child's report:  current value =  1
Parent's report: current index =  1
Parent's report: current index =  2
Child's report:  current value =  4
Parent's report: current index =  3
Child's report:  current value =  9
Parent's report: current index =  4
Child's report:  current value = 16
Parent's report: current index =  5
Child's report:  current value = 25
Parent's report: current index =  6
Child's report:  current value = 36
Parent's report: current index =  7
Child's report:  current value = 49
Parent's report: current index =  8
Child's report:  current value = 64
Parent's report: current index =  9
Child's report:  current value = 81
The child is done
The parent is done
(base) MacBook-Pro:Lab-4 prahaladvijaykumar$
```