Hw3 Report

Muhammed Yasir Fidan

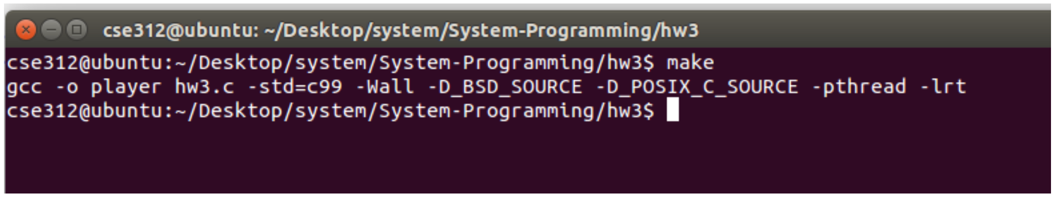
April 2021

1 Abstract

Firstly, My program only works fine for 2 processes, one process has potato and other one not in the beginning. If there are more than 2 processes program stuck in a deadlock. When program starts They exchange potato between their fifos and there is a shared memory for track potato switch and ensure both process dont try to create same fifo. The semaphore I take as argument used protect shared memory to block access two process at the same time. Also I create a second semophore. This semaphore used for making the potato process is waiting for the non-potato process to create its fifos first. If there is not this semaphore, sometimes potato process can reach sending potato case before non-potato process create his fifo, so potato process cant find fifo. For solving this I use this semaphore.

2 Compile And Running

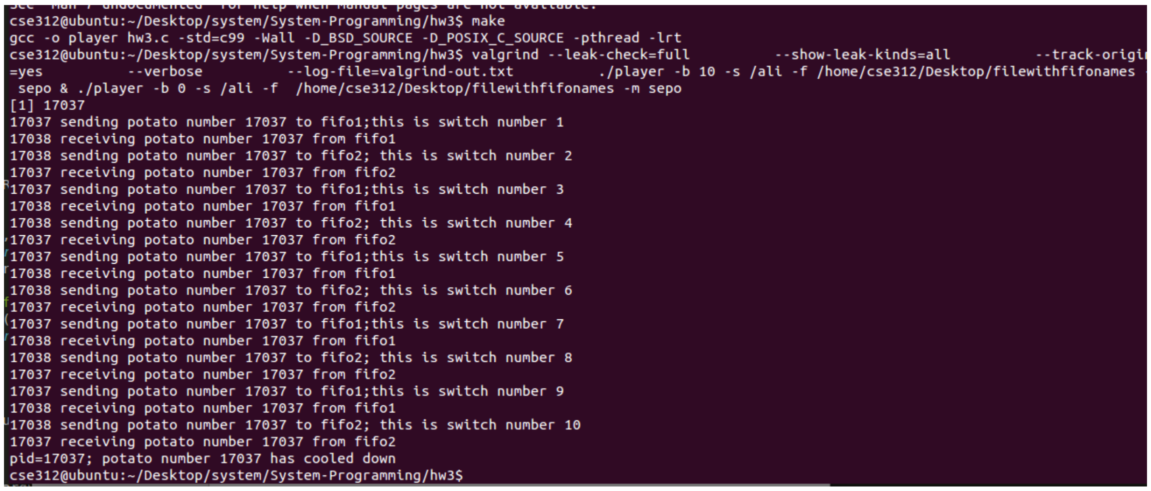
For compiling just use makefile, enter make. In makefile I use -Wall flag too so there is no warning when compiling as you can see.



```
cse312@ubuntu: ~/Desktop/system/System-Programming/hw3
cse312@ubuntu:~/Desktop/system/System-Programming/hw3$ make
gcc -o player hw3.c -std=c99 -Wall -D_BSD_SOURCE -D_POSIX_C_SOURCE -pthread -lrt
cse312@ubuntu:~/Desktop/system/System-Programming/hw3$
```

Figure 1: compile

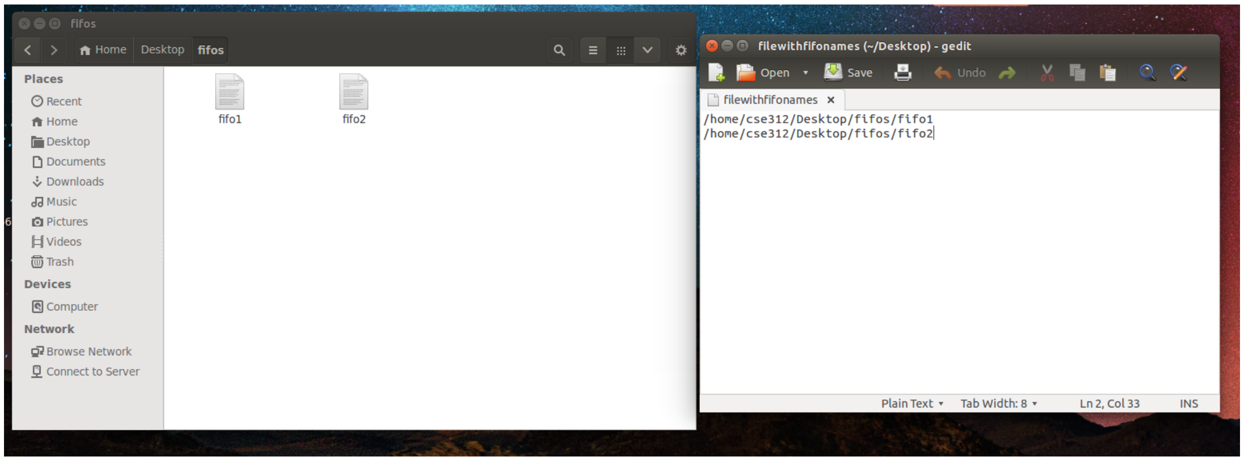
For running execute 2 program by using in terminal. For example here I execute 2 program they have exacly same parameter but one of them -b 10 and other 0. This means -b 10 has a potato that must switch 10 times and other process start without a potato.



```
cse312@ubuntu:~/Desktop/system/System-Programming/hw3$ make
gcc -o player hw3.c -std=c99 -Wall -D_BSD_SOURCE -D_POSIX_C_SOURCE -pthread -lrt
cse312@ubuntu:~/Desktop/system/System-Programming/hw3$ valgrind --leak-check=full          --show-leak-kinds=all          --track-origi
=yes          --verbose          --log-file=valgrind-out.txt          ./player -b 10 -s /ali -f /home/cse312/Desktop/filewithfifonames
 sepo & ./player -b 0 -s /ali -f  /home/cse312/Desktop/filewithfifonames -m sepo
[1] 17037
17037 sending potato number 17037 to fifo1;this is switch number 1
17038 receiving potato number 17037 from fifo1
17038 sending potato number 17037 to fifo2; this is switch number 2
17037 receiving potato number 17037 from fifo2
17037 sending potato number 17037 to fifo1;this is switch number 3
17038 receiving potato number 17037 from fifo1
17038 sending potato number 17037 to fifo2; this is switch number 4
17037 receiving potato number 17037 from fifo2
17037 sending potato number 17037 to fifo1;this is switch number 5
17038 receiving potato number 17037 from fifo1
17038 sending potato number 17037 to fifo2; this is switch number 6
17037 receiving potato number 17037 from fifo2
17037 sending potato number 17037 to fifo1;this is switch number 7
17038 receiving potato number 17037 from fifo1
17038 sending potato number 17037 to fifo2; this is switch number 8
17037 receiving potato number 17037 from fifo2
17037 sending potato number 17037 to fifo1;this is switch number 9
17038 receiving potato number 17037 from fifo1
17038 sending potato number 17037 to fifo2; this is switch number 10
17037 receiving potato number 17037 from fifo2
pid=17037; potato number 17037 has cooled down
cse312@ubuntu:~/Desktop/system/System-Programming/hw3$
```

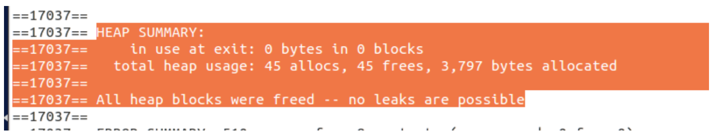
Figure 2: running

Here as you can see processes send potato each other 10 times because -b argument is 10. Then it print potato cool down and both process end. For this running my shared memory name /ali, my semaphore name sepo and -f shows fifo file path. This is my fifo file, here there is 2 fifo path. So my program create this fifos under those paths when my program executed.



```
/home/cse312/Desktop/fifos/fifo1
/home/cse312/Desktop/fifos/fifo2
```

Figure 3: fifo file



```
==17037==
==17037== HEAP SUMMARY:
==17037==     in use at exit: 0 bytes in 0 blocks
==17037==   total heap usage: 45 allocs, 45 frees, 3,797 bytes allocated
==17037==
==17037== All heap blocks were freed -- no leaks are possible
==17037==
```

Figure 4: Valgrind memory leak

And in valgrind output there is no memory leak.