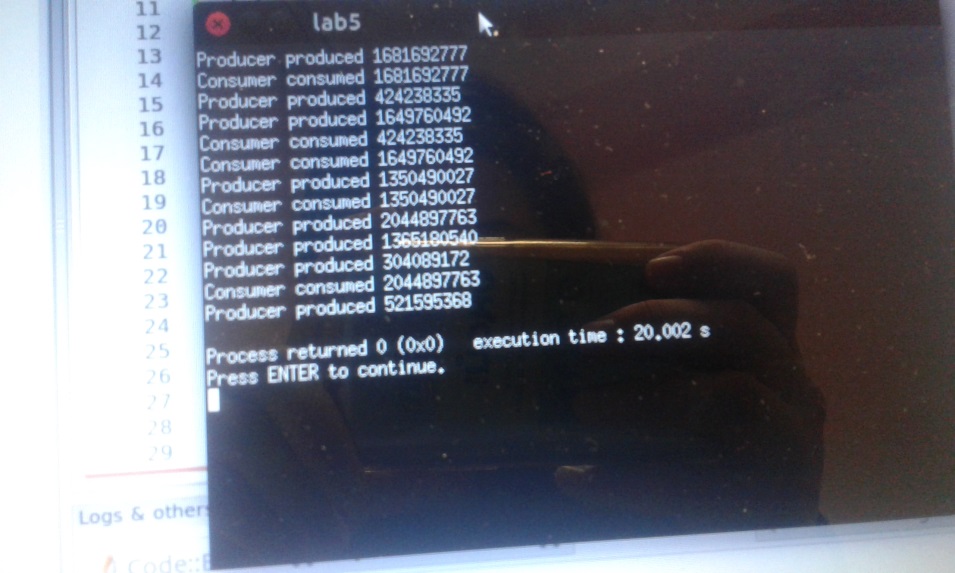
1. Producer\_consumer problem. In this problem we have producer which produces some items and put them into buffer and consumer which removes object and replaces it in item. In the beginning we have empty and full semaphores,which represent our data structure. The semaphore empty is initialized to the number of buffers, the semaphore full is initialized to the value 0. First I needed to implement functions insert\_item(buffer\_item item) and remove\_item(buffer\_item \*item) which return 0 if it is successfully done and -1 if not. In each we lock the mutex object referenced by mutex and semaphore referenced by sem by calling sem\_wait() and pthread\_mutex\_call(). If semaphore value is zero then the calling thread will not return from call until it either locks the semaphore or the call is interrupted by signal. Then we have functions pthread\_mutex\_unlock()and sem\_post() which unlock the state of mutex object and semaphore if calling process sucessfully performed lock operation. In the main() function we create the separate producer and consumer threads, then this function will sleep for 20seconds, after sleeping it will terminate program. Producer and consumer threads will alternate between sleeping for a random period of time (in my case between 1 and 3 seconds).

So result is shown below:



2.Reader-Writer problem. Here is the same logic is used. But here as our data structure we have semaphores mutex and rw\_mutex,which are initialized to 1 and readcount=0; rw mutex is common to both reader and writer, while mutex semaphore is used to ensure mutual exclusion when readcount is changed , which keeps track of how many processes are currently reading the object. rw mutex>mutual exclusion semaphore for the writers and used by the first or last reader that enters or leaves the critical section. In the main() function we create the separate reader and writer threads upon the quantity of each them, then this function will sleep for 20seconds, after sleeping it will terminate program. Reader and writer threads will alternate between sleeping for a random period of time (in my case between 1 and 3 seconds). At all, many processes can to concurrently acquire a reader–writer lock in read mode, but only one process is permitted acquire the lock for writing.

3.The-dining-philosophers problem. Here at all we have 5 philosophers who think and eat bowl of rice. Problem is that they should have two chopsticks in order to eat. It is a simply shows problem of allocation of resources in a deadlock-free and starvation-free manner. So we have semaphore chopstick[5] and all its elements are initialized to 1. Each philosopher will try to grab a chopstick by executing a sem\_wait() operation on that semaphore and go sleep to 5 seconds, then as soon as they finish eating they execute sem\_post() on the appropriate semaphores.Then in the main() function we initialize and destroy our semaphores within in loop with checking value of integer k. Likely we create and join thread philosopher[i] within in loop. As soon as all philosophers finish eating our program terminates.