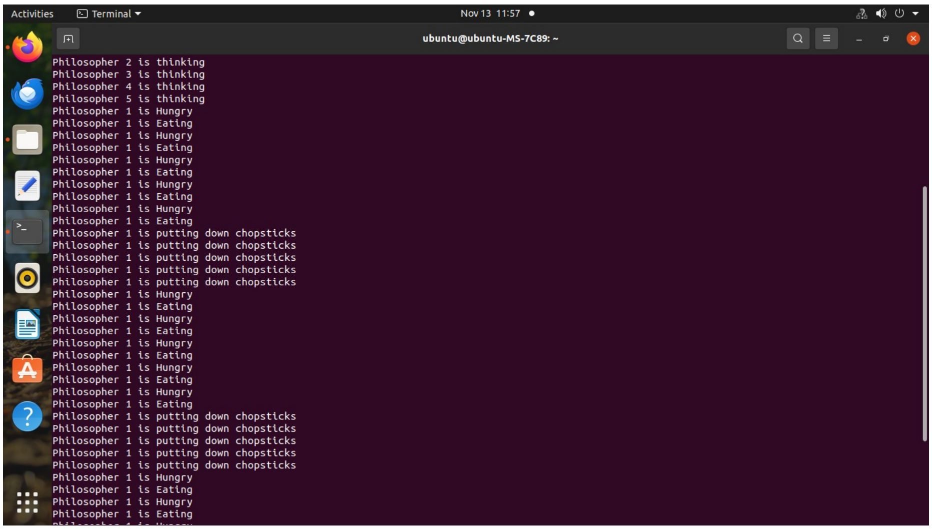
Dinning philosphoer

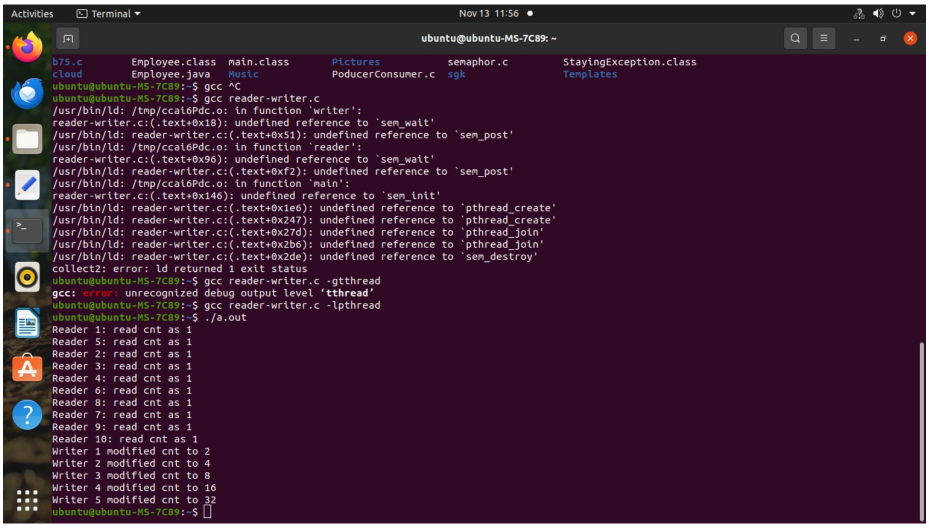
```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_PHILOSOPHERS 5
pthread_mutex_t forks[NUM_PHILOSOPHERS]; // Mutexes for forks
void* philosopher(void* num) {
int id = *(int*)num;
int left_fork = id;
int right_fork = (id + 1) % NUM_PHILOSOPHERS;
while (1) {
// Thinking
printf("Philosopher %d is thinking.\n", id);
sleep(rand() % 3);
// Picking up forks
pthread_mutex_lock(&forks[left_fork]);
printf("Philosopher %d picked up left fork %d.\n", id, left_fork);
pthread_mutex_lock(&forks[right_fork]);
printf("Philosopher %d picked up right fork %d.\n", id, right_fork);
// Eating
printf("Philosopher %d is eating.\n", id);
sleep(rand() % 3);
}
}
// Putting down forks
pthread_mutex_unlock(&forks[right_fork]);
printf("Philosopher %d put down right fork %d.\n", id, right_fork);
pthread_mutex_unlock(&forks[left_fork]);
printf("Philosopher %d put down left fork %d.\n", id, left_fork);
int main() {
pthread_t philosophers[NUM_PHILOSOPHERS];
int ids[NUM_PHILOSOPHERS];
// Initialize mutexes for forks
for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
pthread_mutex_init(&forks[i], NULL);
}
// Create philosopher threads
for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
ids[i] = i;
pthread_create(&philosophers[i], NULL, philosopher, (void*)&ids[i]);
}// Wait for philosopher threads to finish (they won't in this case)
for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
pthread_join(philosophers[i], NULL);
}
// Destroy mutexes
for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
pthread_mutex_destroy(&forks[i]);
}
return 0;
}
```



```
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 1 is Eating
Philosopher 1 is Hungry
Philosopher 1 is Eating
Philosopher 1 is Hungry
Philosopher 1 is Eating
Philosopher 1 is Hungry
Philosopher 1 is Eating
Philosopher 1 is Hungry
Philosopher 1 is Eating
Philosopher 1 is putting down chopsticks
Philosopher 1 is putting down chopsticks
Philosopher 1 is putting down chopsticks
Philosopher 1 is putting down chopsticks
Philosopher 1 is putting down chopsticks
Philosopher 1 is Hungry
Philosopher 1 is Eating
Philosopher 1 is Hungry
Philosopher 1 is Eating
Philosopher 1 is Hungry
Philosopher 1 is Eating
Philosopher 1 is Hungry
Philosopher 1 is Eating
Philosopher 1 is Hungry
Philosopher 1 is Eating
Philosopher 1 is putting down chopsticks
Philosopher 1 is putting down chopsticks
Philosopher 1 is putting down chopsticks
Philosopher 1 is putting down chopsticks
Philosopher 1 is putting down chopsticks
Philosopher 1 is Hungry
Philosopher 1 is Eating
Philosopher 1 is Hungry
Philosopher 1 is Eating
```

Reader writer

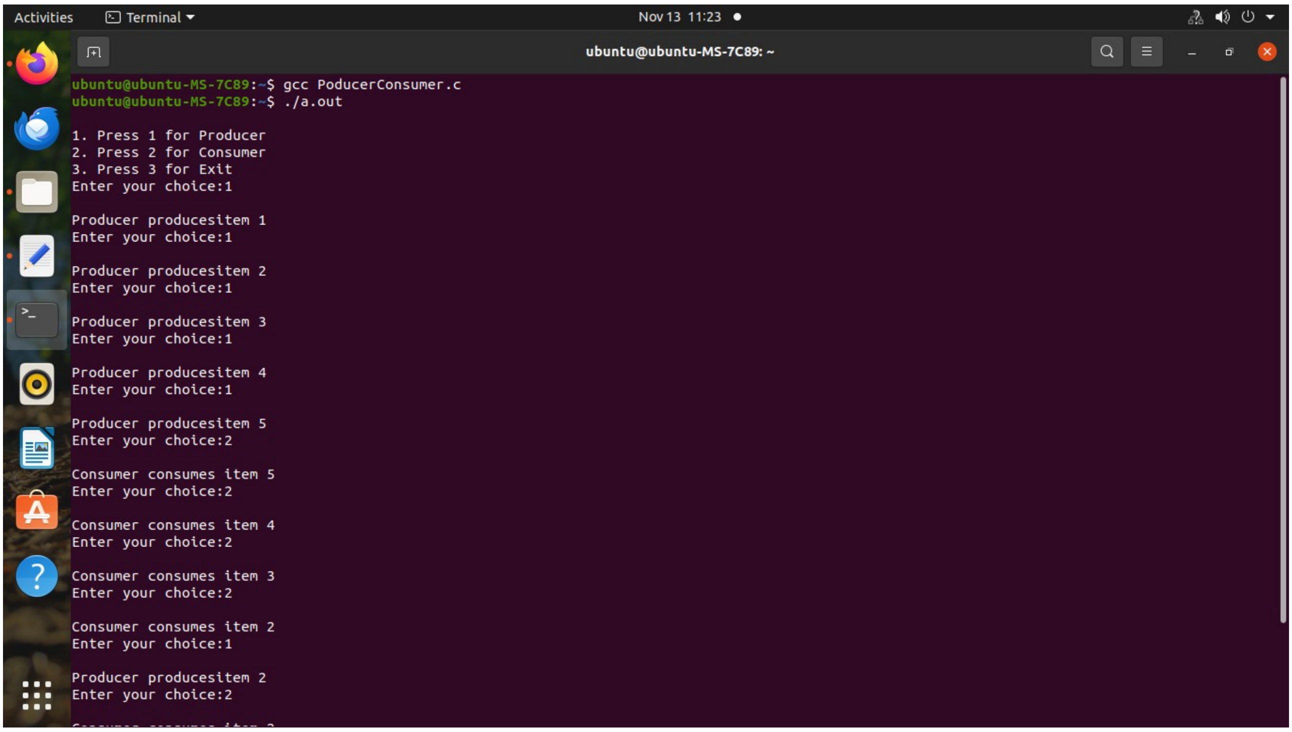
```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define NUM_READERS 5
#define NUM_WRITERS 3
sem_t mutex;
// For protecting the read_count
sem_t writeLock; // For writers
int read_count = 0; // Count of readers
void* reader(void* id) {
int reader_id = *(int*)id;
while (1) {
// Start reading
sem_wait(&mutex); // Lock the mutex to update read_count
read_count++;if (read_count == 1) {
sem_wait(&writeLock); // First reader locks the writer
}
sem_post(&mutex); // Unlock the mutex
// Reading
printf("Reader %d is reading.\n", reader_id);
sleep(rand() % 3); // Simulate reading time
// Finished reading
sem_wait(&mutex); // Lock the mutex to update read_count
read_count--;
if (read_count == 0) {
sem_post(&writeLock); // Last reader unlocks the writer
}
sem_post(&mutex); // Unlock the mutex
}
}
sleep(rand() % 2); // Simulate time between reads
void* writer(void* id) {
int writer_id = *(int*)id;
while (1) {
// Start writing
sem_wait(&writeLock); // Lock for writing
// Writing
printf("Writer %d is writing.\n", writer_id);
sleep(rand() % 3); // Simulate writing time
}
}
// Finished writing
sem_post(&writeLock); // Unlock for writing
sleep(rand() % 2); // Simulate time between writes
int main() {
pthread_t readers[NUM_READERS];
pthread_t writers[NUM_WRITERS];
int ids[NUM_READERS + NUM_WRITERS];
// Initialize semaphores
sem_init(&mutex, 0, 1); // Mutex for read_count
sem_init(&writeLock, 0, 1); // Semaphore for writers
// Create reader threads
for (int i = 0; i < NUM_READERS; i++) {
ids[i] = i;
pthread_create(&readers[i], NULL, reader, (void*)&ids[i]);
}// Create writer threads
for (int i = 0; i < NUM_WRITERS; i++) {
ids[NUM_READERS + i] = i;
pthread_create(&writers[i], NULL, writer, (void*)&ids[NUM_READERS + i]);
}
// Wait for reader threads to finish (they won't in this case)
for (int i = 0; i < NUM_READERS; i++) {
pthread_join(readers[i], NULL);
}
// Wait for writer threads to finish (they won't in this case)
for (int i = 0; i < NUM_WRITERS; i++) {
pthread_join(writers[i], NULL);
}
// Destroy semaphores (won't be reached in this case)
sem_destroy(&mutex);
sem_destroy(&writeLock);
}
return 0;
```



```
b75.c         Employee.class  main.class      Pictures          semaphor.c         StayingException.class
cloud         Employee.java   Music           PoducerConsumer.c  sgk               Templates
ubuntu@ubuntu-MS-7C89:~$ gcc ^C
ubuntu@ubuntu-MS-7C89:~$ gcc reader-writer.c
/usr/bin/ld: /tmp/ccai6Pdc.o: in function `writer':
reader-writer.c:(.text+0x18): undefined reference to `sem_wait'
/usr/bin/ld: reader-writer.c:(.text+0x51): undefined reference to `sem_post'
/usr/bin/ld: /tmp/ccai6Pdc.o: in function `reader':
reader-writer.c:(.text+0x96): undefined reference to `sem_wait'
/usr/bin/ld: reader-writer.c:(.text+0xf2): undefined reference to `sem_post'
/usr/bin/ld: /tmp/ccai6Pdc.o: in function `main':
reader-writer.c:(.text+0x146): undefined reference to `sem_init'
/usr/bin/ld: reader-writer.c:(.text+0x1e6): undefined reference to `pthread_create'
/usr/bin/ld: reader-writer.c:(.text+0x247): undefined reference to `pthread_create'
/usr/bin/ld: reader-writer.c:(.text+0x27d): undefined reference to `pthread_join'
/usr/bin/ld: reader-writer.c:(.text+0x2b6): undefined reference to `pthread_join'
/usr/bin/ld: reader-writer.c:(.text+0x2de): undefined reference to `sem_destroy'
collect2: error: ld returned 1 exit status
ubuntu@ubuntu-MS-7C89:~$ gcc reader-writer.c -gtthread
gcc: error: unrecognized debug output level ‘tthread’
ubuntu@ubuntu-MS-7C89:~$ gcc reader-writer.c -lpthread
ubuntu@ubuntu-MS-7C89:~$ ./a.out
Reader 1: read cnt as 1
Reader 5: read cnt as 1
Reader 2: read cnt as 1
Reader 3: read cnt as 1
Reader 4: read cnt as 1
Reader 6: read cnt as 1
Reader 8: read cnt as 1
Reader 7: read cnt as 1
Reader 9: read cnt as 1
Reader 10: read cnt as 1
Writer 1 modified cnt to 2
Writer 2 modified cnt to 4
Writer 3 modified cnt to 8
Writer 4 modified cnt to 16
Writer 5 modified cnt to 32
ubuntu@ubuntu-MS-7C89:~$ 
```

Bounded-Buffer:

```
#include <stdio.h>
#include <stdlib.h>

int mutex = 1;


int full = 0;


int empty = 10, x = 0;

void producer()
{

    --mutex;


    ++full;


    --empty;


    x++;
    printf("\nProducer produces"
           "item %d",
           x);


    ++mutex;
}


void consumer()
{

    --mutex;



    --full;


    ++empty;
    printf("\nConsumer consumes "
           "item %d",
           x);
    x--;


    ++mutex;
}


int main()
{
    int n, i;
    printf("\n1. Press 1 for Producer"
           "\n2. Press 2 for Consumer"
           "\n3. Press 3 for Exit");


#pragma omp critical

    for (i = 1; i > 0; i++) {

        printf("\nEnter your choice:");
        scanf("%d", &n);


        switch (n) {
        case 1:


            if ((mutex == 1)
                && (empty != 0)) {
                producer();
            }


            else {
                printf("Buffer is full!");
            }
            break;

        case 2:


            if ((mutex == 1)
                && (full != 0)) {
                consumer();
            }


            else {
                printf("Buffer is empty!");
            }
            break;
        case 3:
            exit(0);
            break;
        }
    }
}
```



```
ubuntu@ubuntu-MS-7C89:~$ gcc PoducerConsumer.c
ubuntu@ubuntu-MS-7C89:~$ ./a.out

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:1

Producer producesitem 1
Enter your choice:1

Producer producesitem 2
Enter your choice:1

Producer producesitem 3
Enter your choice:1

Producer producesitem 4
Enter your choice:1

Producer producesitem 5
Enter your choice:2

Consumer consumes item 5
Enter your choice:2

Consumer consumes item 4
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:1

Producer producesitem 2
Enter your choice:2
```