Note

- All Codes of this assignment are available here

Question

Dining Philosophers Algorithm implemented using Monitors

Code

```
/**
 * @author YashKumarVerma / 19BCE2669
 * @for: Lab Digital Assignment 2
 * @description: Implementation of Dining Philosophers Problem using
Monitors
 */

#include<iostream>
#include<vector>

using namespace std;

/** Utility function to format display as requireds **/
class console{
    public:
    static void log(string status, string message){
        cout << "[" << status << "] :: " << message << endl;
    }

    static void log(string status, int message){
        cout << "[" << status << "] :: " << message << endl;
    }
};


class Philosopher {
    public:
        bool left;
        bool right;
        bool takenDinner;

    Philosopher(bool left = false, bool right = false){
        this->left = left;
        this->right = right;
        this->takenDinner = false;
    }

    bool justTakenTwoForks(){
        return this->left && this->right;
    }
```

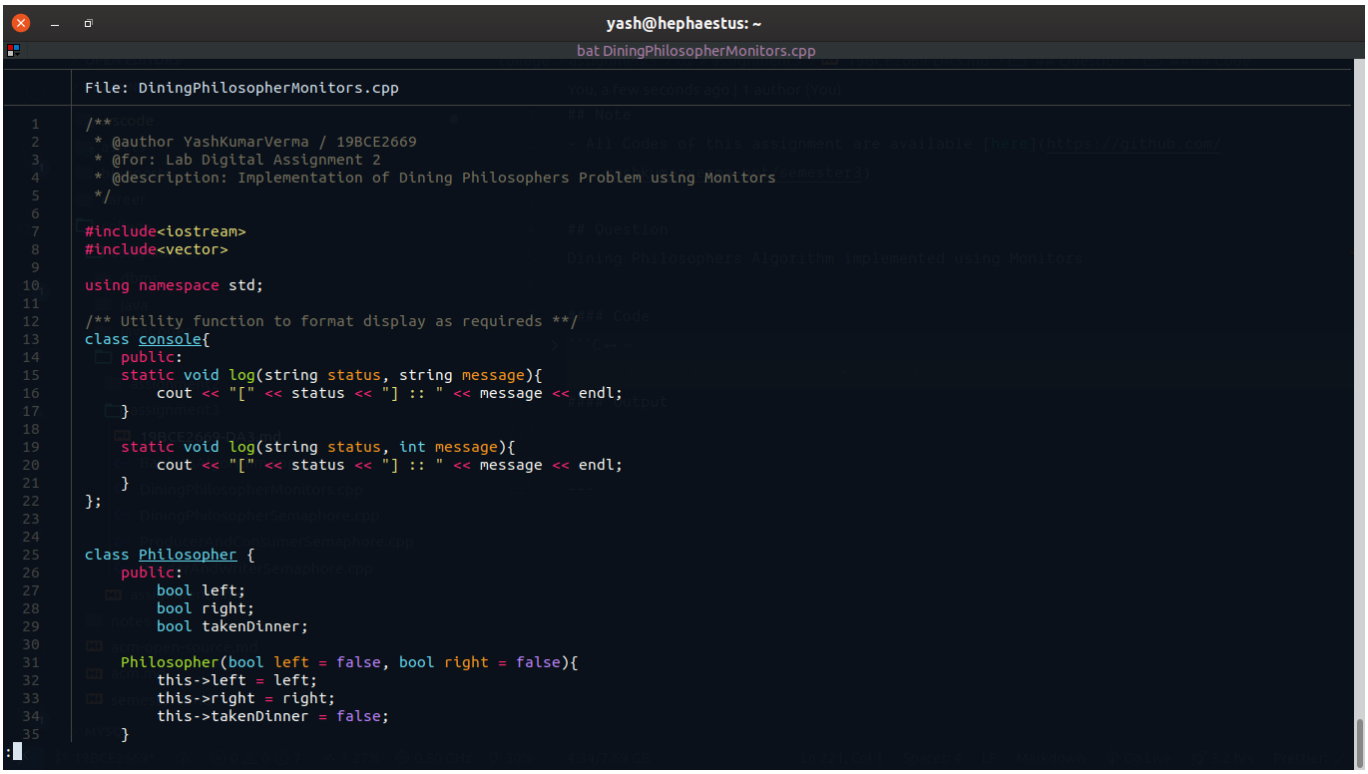
```
    bool hasLeftAndWantsRight(){
        return this->left && !this->right;
    }

    bool hasNoForks(){
        return !this->left && !this->right;
    }
};

/**
 * @class Fork
 * @description To handle all operating related to fork / chopstick
 **/

class Fork {
    private:
        bool taken;
    public:
    Fork(bool taken = false){
        this->taken = taken;
    }

    bool isTaken(){
        return this->taken;
    }

    void take(){
        this->taken = true;
    }

    void release(){
        this->taken = false;
    }
};

/**
 * @class DiningPhilosophers
 * @description To handle all operating related to problem
 **/
class DiningPhilosophers{
    private:
        vector<Philosopher> philosopher;
        vector<Fork> fork;

    public:
        int completedPhilosophers;
        int totalPhilosophers;

    /** set initial conditions **/
    DiningPhilosophers(int totalPhilosophers = 5){
        this->completedPhilosophers = 0;
        this->totalPhilosophers = totalPhilosophers;

        for(int i=0; i<this->totalPhilosophers; i++){
```

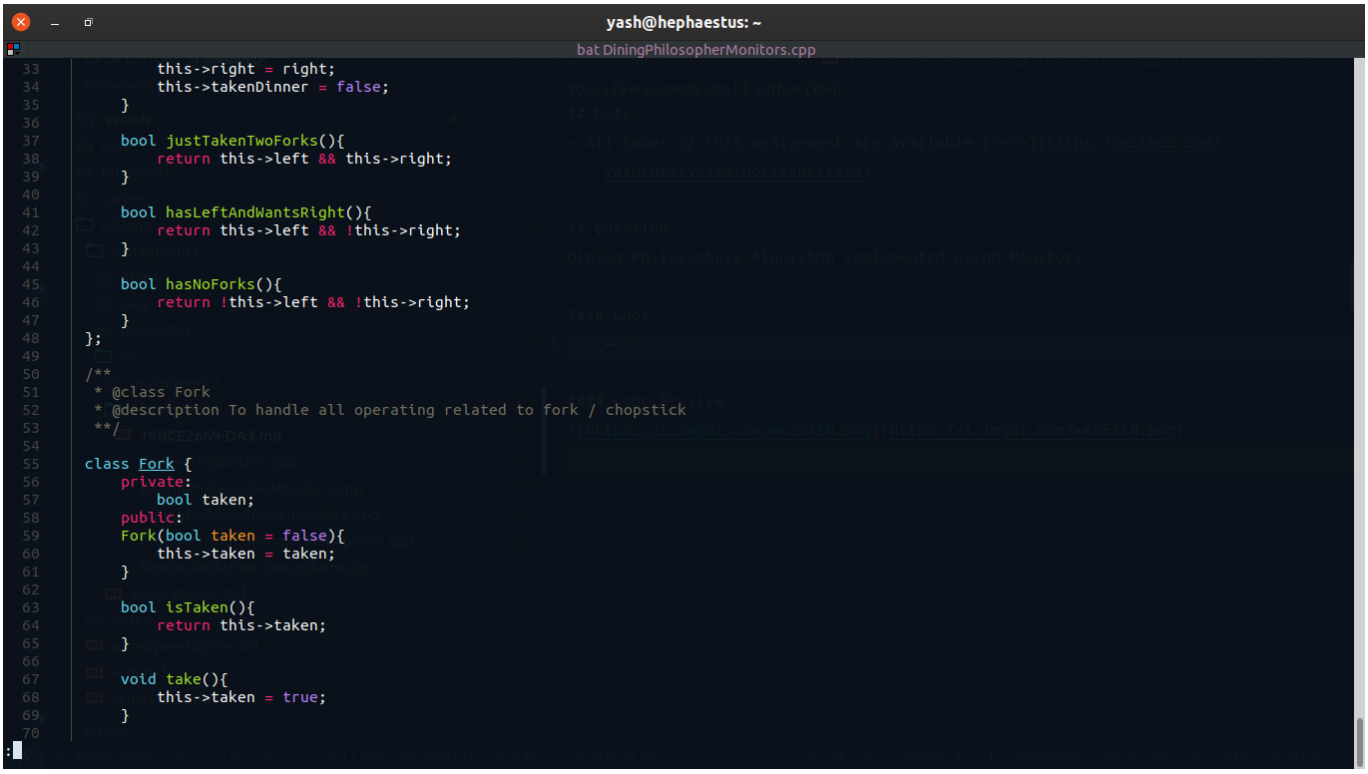
```
            this->philosopher.push_back(Philosopher());
            this->fork.push_back(Fork());
        }
    }

    /** start dinner by philosophers **/
    void startDinner(){
        console::log("starting", "meal");
        while(this->completedPhilosophers < this->totalPhilosophers){
            console::log("totalPhilosophers", this->totalPhilosophers);
            console::log("completedPhilosophers", this-
>completedPhilosophers);

            for(int philosopherId=0; philosopherId< this-
>totalPhilosophers; philosopherId += 1){
                console::log("starting dinner", philosopherId);
                this->attemptToEat(philosopherId);
            }
        }
        console::log("completing", "meal");
    }

    /** make attempts to eat **/
    void attemptToEat(int philosopherId){
        /** if dinner already taken **/
        if(this->philosopher[philosopherId].takenDinner){
            console::log("taken dinner", philosopherId);
        }

        /** then the philosopher can take dinner **/
        else if(this->philosopher[philosopherId].justTakenTwoForks()){
            console::log("taken dinner", philosopherId);
            this->philosopher[philosopherId].takenDinner = true;

            int otherFork = philosopherId - 1;
            if(otherFork == -1){
                otherFork = this->totalPhilosophers - 1;
            }

            /** releasing forks */
            this->fork[philosopherId].release();
            this->fork[otherFork].release();

            console::log("philosopher", philosopherId );
            console::log("released fork", philosopherId );
            console::log("released fork", otherFork );

            this->completedPhilosophers += 1;
        }

        else if(this->philosopher[philosopherId].hasLeftAndWantsRight()){
            if(philosopherId == (this->totalPhilosophers - 1)){
                if(!this->fork[philosopherId].isTaken()){
                    this->fork[philosopherId].take();
```

```
                    this->philosopher[philosopherId].right = true;
                    console::log("philosopher",philosopherId );
                    console::log("picks fork", philosopherId );
                }
                else{
                    console::log("philosopher",philosopherId );
                    console::log("waiting for fork", philosopherId );
                }
            }
            /** case of last philosopher **/
            else {
                int duplicatePhilosopherId = philosopherId;
                philosopherId -= 1;

                if(philosopherId == -1){
                    philosopherId = this->totalPhilosophers - 1;
                }

                if(!this->fork[philosopherId].isTaken()){
                    this->fork[philosopherId].take();
                    this->philosopher[duplicatePhilosopherId].right = true;
                    console::log("philosopher",philosopherId );
                    console::log("picks fork", duplicatePhilosopherId );
                }
                else{
                    console::log("philosopher",duplicatePhilosopherId );
                    console::log("waiting for fork", philosopherId );
                }
            }
        }
        else if(this->philosopher[philosopherId].hasNoForks()){
            if(philosopherId == this->totalPhilosophers - 1){
                if(!this->fork[philosopherId-1].isTaken()){
                    this->fork[philosopherId-1].take();
                    this->philosopher[philosopherId].left = true;
                    console::log("philosopher",philosopherId );
                    console::log("picks fork", philosopherId-1);
                }
                else {
                    console::log("philosopher",philosopherId );
                    console::log("waiting for fork", philosopherId-1);
                }
            }
            /** case of last philosopher **/
            else {
                if(!this->fork[philosopherId].isTaken()){
                    this->fork[philosopherId].take();
                    this->philosopher[philosopherId].left = true;
                    console::log("philosopher",philosopherId );
                    console::log("picks fork", philosopherId);
                }
                else{
                    console::log("philosopher",philosopherId );
                    console::log("waiting for fork", philosopherId);
```

```
                    }
                }
            }
        }
};

/** Driver code **/
int main(){
    DiningPhilosophers diningPhilosophers = DiningPhilosophers(5);
    diningPhilosophers.startDinner();
}
```

Code Preview


yash@hephaestus: ~
bat DiningPhilosopherMonitors.cpp
File: DiningPhilosopherMonitors.cpp
/**
* @author YashKumarVerma / 19BCE2669
* @for: Lab Digital Assignment 2
* @description: Implementation of Dining Philosophers Problem using Monitors
*/
#include<iostream>
#include<vector>
using namespace std;
/** Utility function to format display as requireds **/
class console{
public:
static void log(string status, string message){
cout << "[" << status << "] :: " << message << endl;
}
static void log(string status, int message){
cout << "[" << status << "] :: " << message << endl;
}
};
class Philosopher {
public:
bool left;
bool right;
bool takenDinner;
Philosopher(bool left = false, bool right = false){
this->left = left;
this->right = right;
this->takenDinner = false;
}


yash@hephaestus: ~
bat DiningPhilosopherMonitors.cpp
this->right = right;
this->takenDinner = false;
}
bool justTakenTwoForks(){
return this->left && this->right;
}
bool hasLeftAndWantsRight(){
return this->left && !this->right;
}
bool hasNoForks(){
return !this->left && !this->right;
}
};
/**
* @class Fork
* @description To handle all operating related to fork / chopstick
**/
class Fork {
private:
bool taken;
public:
Fork(bool taken = false){
this->taken = taken;
}
bool isTaken(){
return this->taken;
}
void take(){
this->taken = true;
}

yash@hephaestus: ~

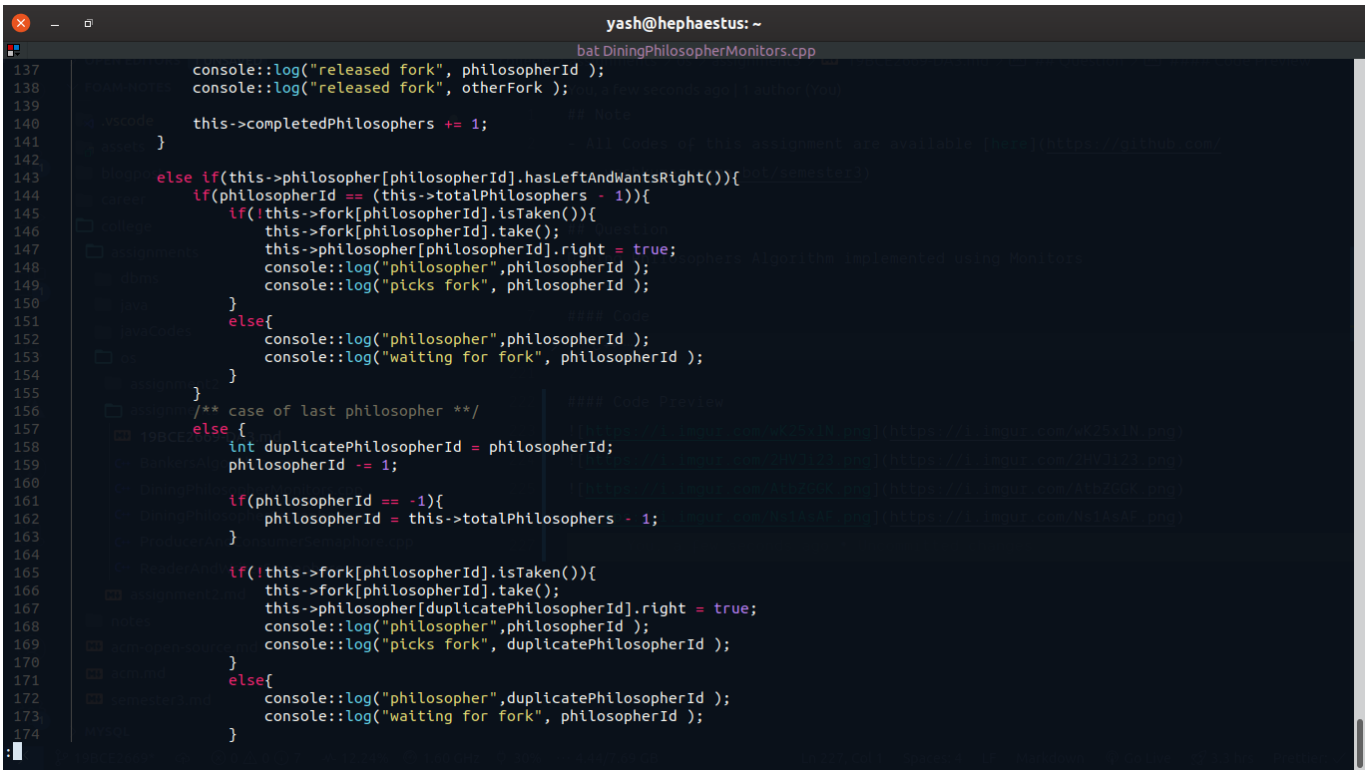
bat DiningPhilosopherMonitors.cpp

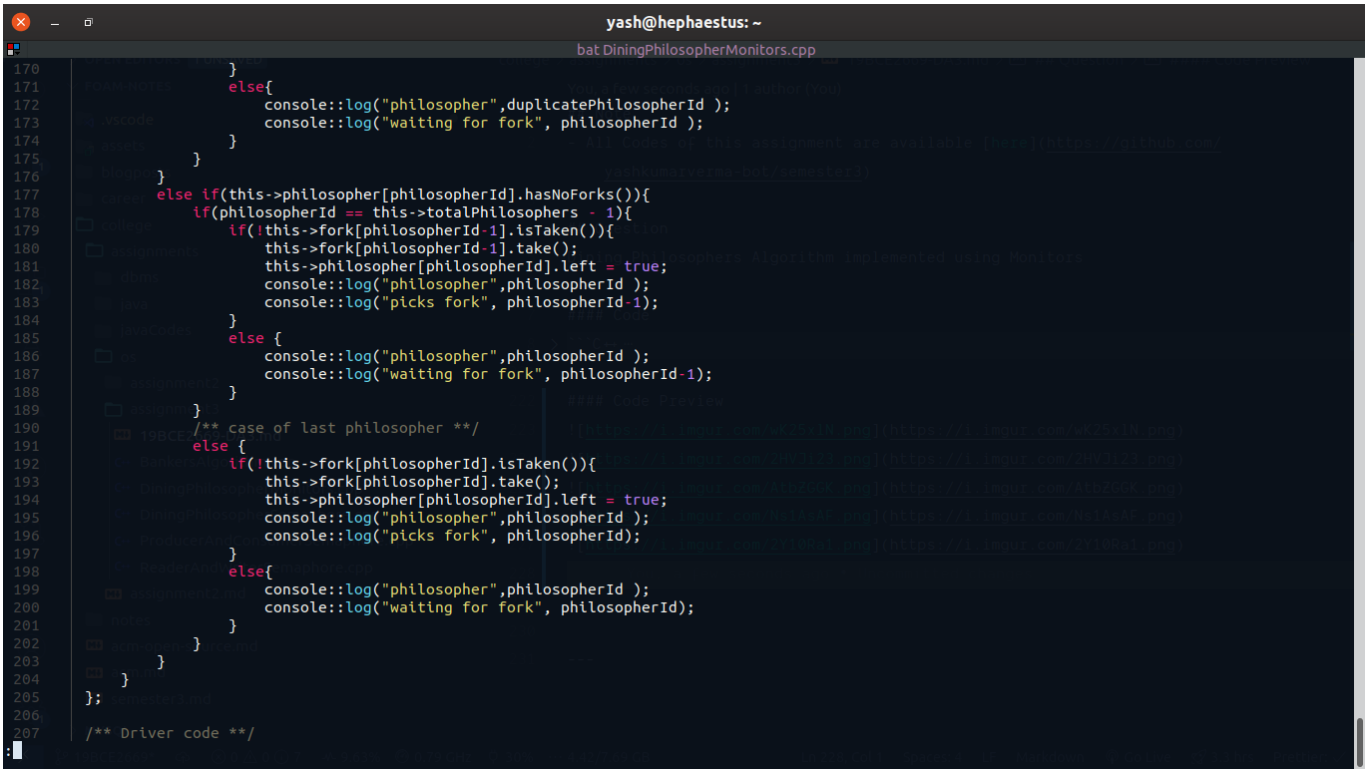
```
    void take(){
        this->taken = true;
    }
    void release(){
        this->taken = false;
    }
};

/**
 * @class DiningPhilosophers
 * @description To handle all operating related to problem
 **/
class DiningPhilosophers{
    private:
        vector<Philosopher> philosopher;
        vector<Fork> fork;

    public:
        int completedPhilosophers;
        int totalPhilosophers;

        /** set initial conditions **/
        DiningPhilosophers(int totalPhilosophers = 5){
            this->completedPhilosophers = 0;
            this->totalPhilosophers = totalPhilosophers;

            for(int i=0; i<this->totalPhilosophers; i++){
                this->philosopher.push_back(Philosopher());
                this->fork.push_back(Fork());
            }
        }

        /** start dinner by philosophers **/
        void startDinner(){
            console::log("starting", "meal");
            while(this->completedPhilosophers < this->totalPhilosophers){
                console::log("totalPhilosophers", this->totalPhilosophers);
```

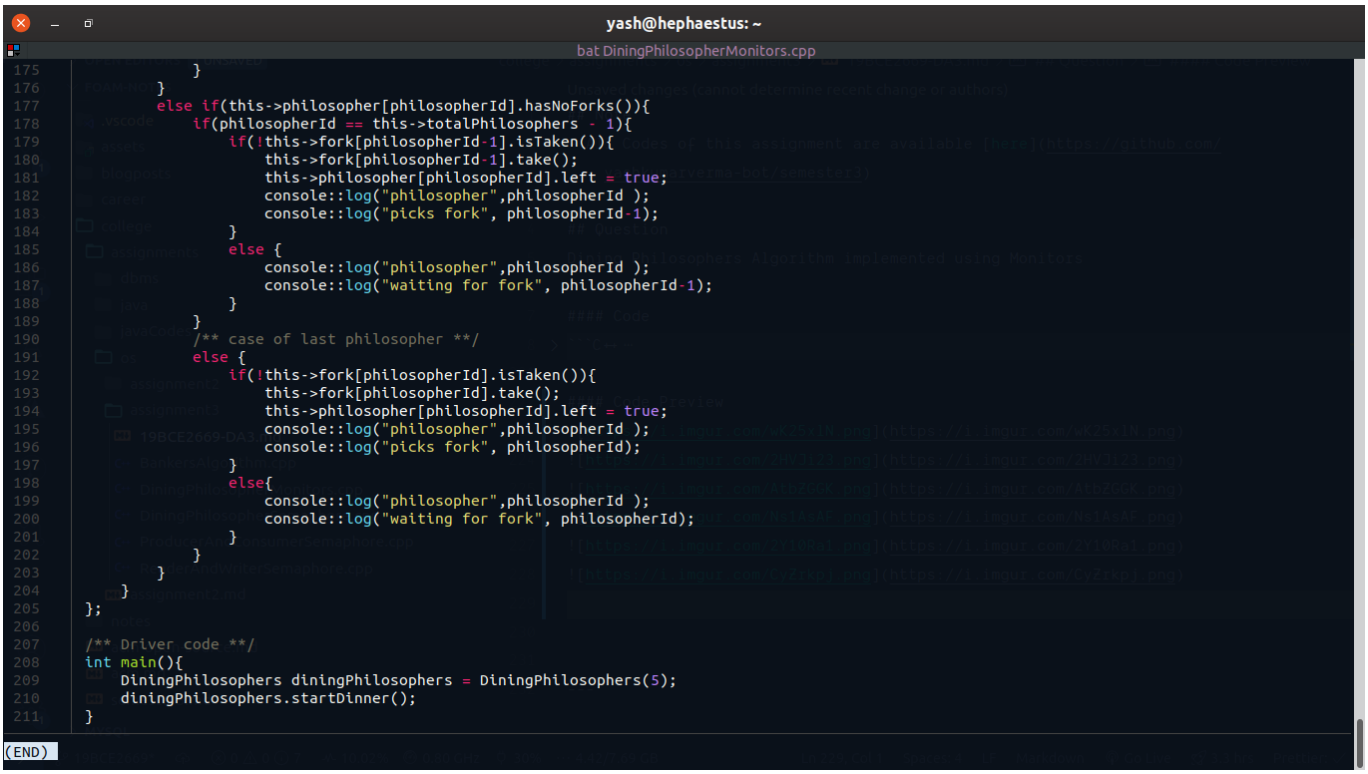
yash@hephaestus: ~

bat DiningPhilosopherMonitors.cpp

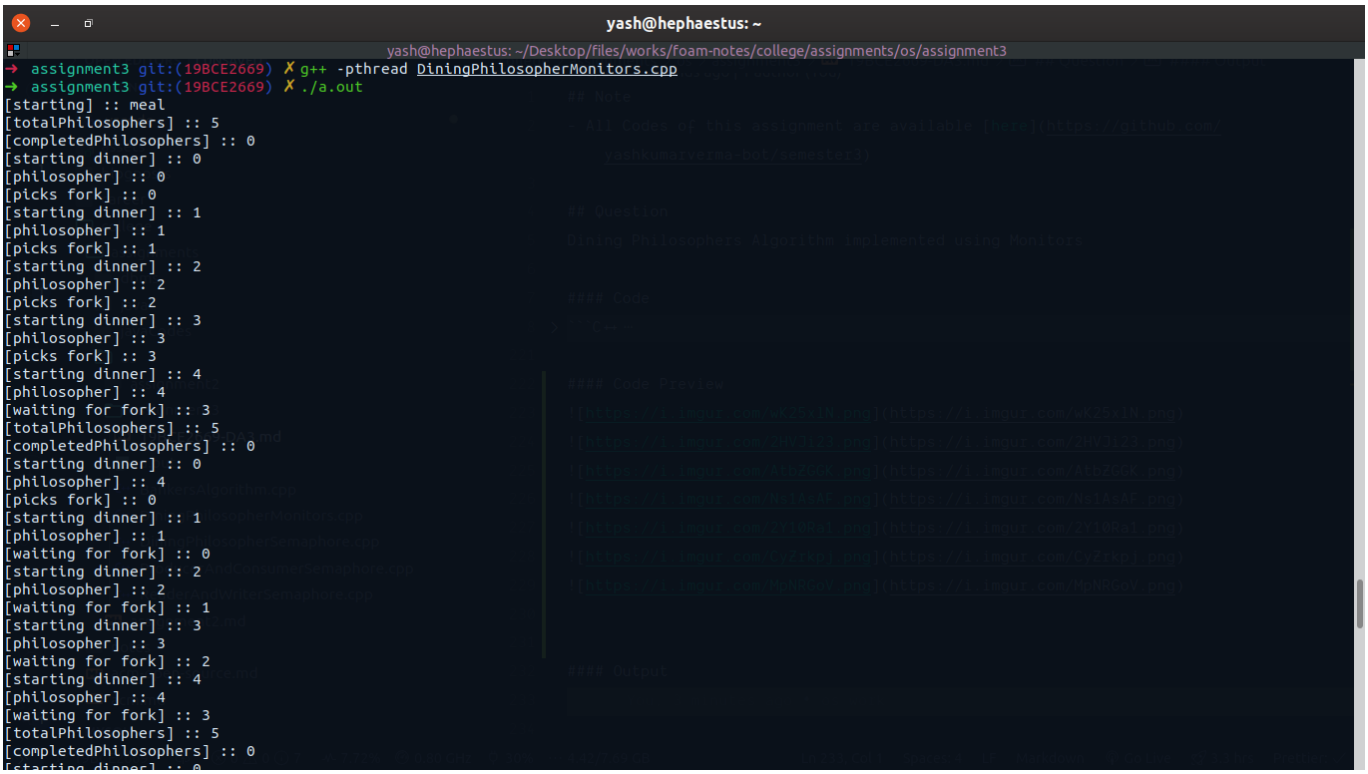
```
            console::log("starting", "meal");
            while(this->completedPhilosophers < this->totalPhilosophers){
                console::log("totalPhilosophers", this->totalPhilosophers);
                console::log("completedPhilosophers", this->completedPhilosophers);

                for(int philosopherId=0; philosopherId< this->totalPhilosophers; philosopherId += 1){
                    console::log("starting dinner", philosopherId);
                    this->attemptToEat(philosopherId);
                }
            }
            console::log("completing", "meal");
        }

        /** make attempts to eat **/
        void attemptToEat(int philosopherId){
            /** if dinner already taken **/
            if(this->philosopher[philosopherId].takenDinner){
                console::log("taken dinner", philosopherId);
            }

            /** then the philosopher can take dinner **/
            else if(this->philosopher[philosopherId].justTakenTwoForks()){
                console::log("taken dinner", philosopherId);
                this->philosopher[philosopherId].takenDinner = true;

                int otherFork = philosopherId - 1;
                if(otherFork == -1){
                    otherFork = this->totalPhilosophers - 1;
                }

                /** releasing forks */
                this->fork[philosopherId].release();
                this->fork[otherFork].release();

                console::log("philosopher", philosopherId );
                console::log("released fork", philosopherId );
                console::log("released fork", otherFork );
```

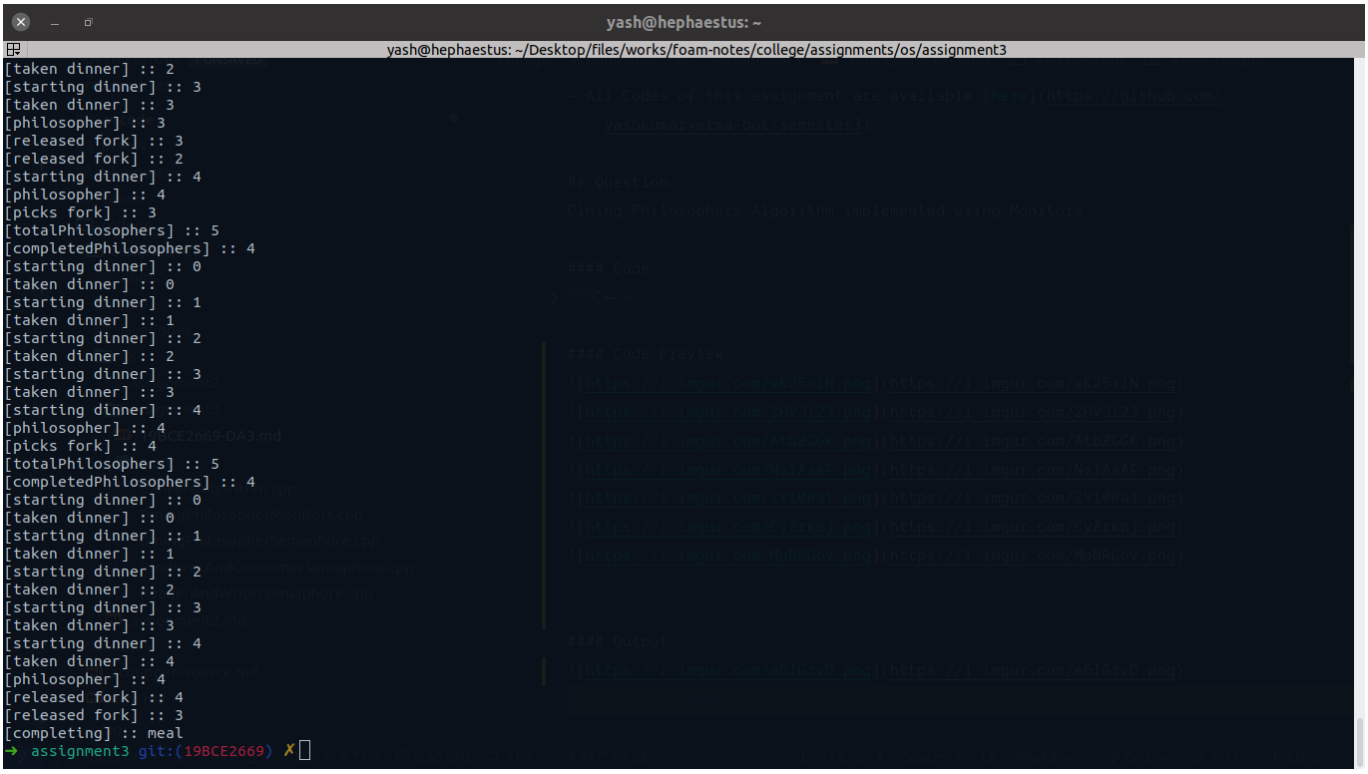

yash@hephaestus: ~
bat DiningPhilosopherMonitors.cpp
console::log("released fork", philosopherId);
console::log("released fork", otherFork);
this->completedPhilosophers += 1;
}
else if(this->philosopher[philosopherId].hasLeftAndWantsRight()){
if(philosopherId == (this->totalPhilosophers - 1)){
if(!this->fork[philosopherId].isTaken()){
this->fork[philosopherId].take();
this->philosopher[philosopherId].right = true;
console::log("philosopher",philosopherId);
console::log("picks fork", philosopherId);
}
else{
console::log("philosopher",philosopherId);
console::log("waiting for fork", philosopherId);
}
}
/** case of last philosopher **/
else {
int duplicatePhilosopherId = philosopherId;
philosopherId -= 1;
if(philosopherId == -1){
philosopherId = this->totalPhilosophers - 1;
}
if(!this->fork[philosopherId].isTaken()){
this->fork[philosopherId].take();
this->philosopher[duplicatePhilosopherId].right = true;
console::log("philosopher",philosopherId);
console::log("picks fork", duplicatePhilosopherId);
}
else{
console::log("philosopher",duplicatePhilosopherId);
console::log("waiting for fork", philosopherId);
}


yash@hephaestus: ~
bat DiningPhilosopherMonitors.cpp
}
else{
console::log("philosopher",duplicatePhilosopherId);
console::log("waiting for fork", philosopherId);
}
}
}
else if(this->philosopher[philosopherId].hasNoForks()){
if(philosopherId == this->totalPhilosophers - 1){
if(!this->fork[philosopherId-1].isTaken()){
this->fork[philosopherId-1].take();
this->philosopher[philosopherId].left = true;
console::log("philosopher",philosopherId);
console::log("picks fork", philosopherId-1);
}
else {
console::log("philosopher",philosopherId);
console::log("waiting for fork", philosopherId-1);
}
}
/** case of last philosopher **/
else {
if(!this->fork[philosopherId].isTaken()){
this->fork[philosopherId].take();
this->philosopher[philosopherId].left = true;
console::log("philosopher",philosopherId);
console::log("picks fork", philosopherId);
}
else{
console::log("philosopher",philosopherId);
console::log("waiting for fork", philosopherId);
}
}
}
}
};
/** Driver code **/


yash@hephaestus: ~
bat DiningPhilosopherMonitors.cpp
}
}
else if(this->philosopher[philosopherId].hasNoForks()){
if(philosopherId == this->totalPhilosophers - 1){
if(!this->fork[philosopherId-1].isTaken()){
this->fork[philosopherId-1].take();
this->philosopher[philosopherId].left = true;
console::log("philosopher",philosopherId);
console::log("picks fork", philosopherId-1);
}
else {
console::log("philosopher",philosopherId);
console::log("waiting for fork", philosopherId-1);
}
}
/** case of last philosopher **/
else {
if(!this->fork[philosopherId].isTaken()){
this->fork[philosopherId].take();
this->philosopher[philosopherId].left = true;
console::log("philosopher",philosopherId);
console::log("picks fork", philosopherId);
}
else{
console::log("philosopher",philosopherId);
console::log("waiting for fork", philosopherId);
}
}
}
}
};
/** Driver code **/
int main(){
DiningPhilosophers diningPhilosophers = DiningPhilosophers(5);
diningPhilosophers.startDinner();
}
(END)

Output





Question

Dining Philosophers Algorithm implemented using Semaphores

Code

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

```c
#include <semaphore.h>

typedef struct {
  int position;
  int count;
  sem_t *forks;
  sem_t *lock;
} params_t;

void initialize_semaphores(sem_t *lock, sem_t *forks, int num_forks);
void run_all_threads(pthread_t *threads, sem_t *forks, sem_t *lock, int
num_philosophers);

void *philosopher(void *params);
void think(int position);
void eat(int position);

int main(int argc, char *args[])
{
  int num_philosophers = 5;

  sem_t lock;
  sem_t forks[num_philosophers];
  pthread_t philosophers[num_philosophers];

  initialize_semaphores(&lock, forks, num_philosophers);
  run_all_threads(philosophers, forks, &lock, num_philosophers);
  pthread_exit(NULL);
}

void initialize_semaphores(sem_t *lock, sem_t *forks, int num_forks)
{
  int i;
  for(i = 0; i < num_forks; i++) {
    sem_init(&forks[i], 0, 1);
  }

  sem_init(lock, 0, num_forks - 1);
}

void run_all_threads(pthread_t *threads, sem_t *forks, sem_t *lock, int
num_philosophers)
{
  int i;
  for(i = 0; i < num_philosophers; i++) {
    params_t *arg = malloc(sizeof(params_t));
    arg->position = i;
    arg->count = num_philosophers;
    arg->lock = lock;
    arg->forks = forks;

    pthread_create(&threads[i], NULL, philosopher, (void *)arg);
  }
}
```

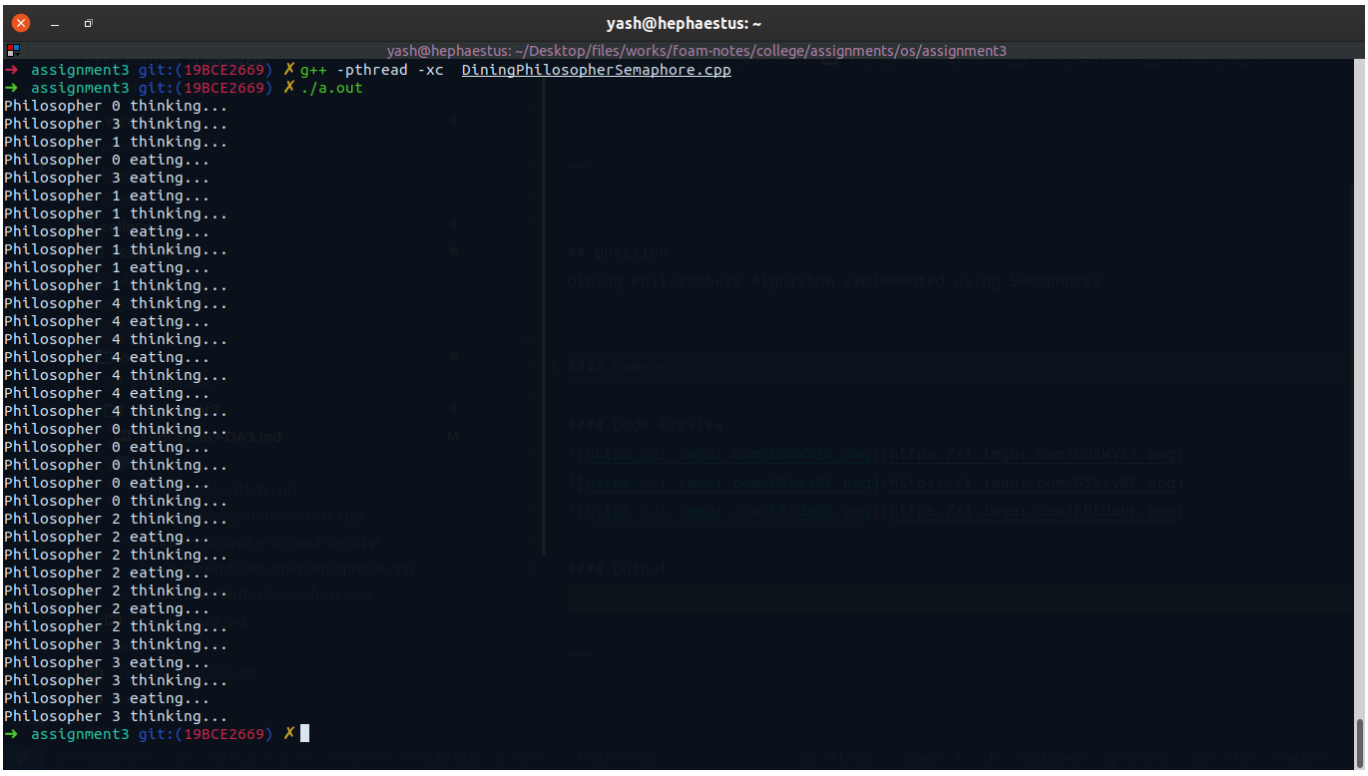
```
void *philosopher(void *params)
{
  int i;
  params_t self = *(params_t *)params;

  for(i = 0; i < 3; i++) {
    think(self.position);

    sem_wait(self.lock);
    sem_wait(&self.forks[self.position]);
    sem_wait(&self.forks[(self.position + 1) % self.count]);
    eat(self.position);
    sem_post(&self.forks[self.position]);
    sem_post(&self.forks[(self.position + 1) % self.count]);
    sem_post(self.lock);
  }

  think(self.position);
  pthread_exit(NULL);
}

void think(int position)
{
  printf("Philosopher %d thinking...\n", position);
}

void eat(int position)
{
  printf("Philosopher %d eating...\n", position);
}
```

Code Preview


yash@hephaestus: ~
bat DiningPhilosopherSemaphore.cpp
File: DiningPhilosopherSemaphore.cpp
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
typedef struct {
int position;
int count;
sem_t *forks;
sem_t *lock;
} params_t;
void initialize_semaphores(sem_t *lock, sem_t *forks, int num_forks);
void run_all_threads(pthread_t *threads, sem_t *forks, sem_t *lock, int num_philosophers);
void *philosopher(void *params);
void think(int position);
void eat(int position);
int main(int argc, char *args[])
{
int num_philosophers = 5;
sem_t lock;
sem_t forks[num_philosophers];
pthread_t philosophers[num_philosophers];
initialize_semaphores(&lock, forks, num_philosophers);
run_all_threads(philosophers, forks, &lock, num_philosophers);
pthread_exit(NULL);
}
void initialize_semaphores(sem_t *lock, sem_t *forks, int num_forks)
{
int i;


yash@hephaestus: ~
bat DiningPhilosopherSemaphore.cpp
void initialize_semaphores(sem_t *lock, sem_t *forks, int num_forks)
{
int i;
for(i = 0; i < num_forks; i++) {
sem_init(&forks[i], 0, 1);
}
sem_init(lock, 0, num_forks - 1);
}
void run_all_threads(pthread_t *threads, sem_t *forks, sem_t *lock, int num_philosophers)
{
int i;
for(i = 0; i < num_philosophers; i++) {
params_t *arg = malloc(sizeof(params_t));
arg->position = i;
arg->count = num_philosophers;
arg->lock = lock;
arg->forks = forks;
pthread_create(&threads[i], NULL, philosopher, (void *)arg);
}
}
void *philosopher(void *params)
{
int i;
params_t self = *(params_t *)params;
for(i = 0; i < 3; i++) {
think(self.position);
sem_wait(self.lock);
sem_wait(&self.forks[self.position]);
sem_wait(&self.forks[(self.position + 1) % self.count]);
eat(self.position);
sem_post(&self.forks[self.position]);

Output



Question

Implementation of Bankers Algorithm

Code

```
/**
 * @author YashKumarVerma / 19BCE2669
 * @for: Lab Digital Assignment 3
 * @description: Implementation of Bankers Algorithm
 */

#include <bits/stdc++.h>

using namespace std;

/** Class to handle all operating and keep data **/
class BankerAlgorithm {
public:
  int p;
  bool come;
  int keepOn;
  int Max[10][3];
  int current[3];
  int need[10][3];
  bool executed[10];
  int available[10][3];
  int allocation[10][3];

  BankerAlgorithm() { int keepOn = 1; }

  void TakeInput() {
    cout << "Enter No. of processes: ";
    cin >> p;
    cout << "\n";
    cout << "Enter the current resources: ";
    cin >> current[0] >> current[1] >> current[2];

    for (int i = 0; i < p; ++i) {
      cout << "[19BCE2669] \t\t > Process P" << i + 1 << " Details" <<
endl;
      cout << "Enter Allocation : ";
      cin >> allocation[i][0] >> allocation[i][1] >> allocation[i][2];
      cout << "Enter Max :";
      cin >> Max[i][0] >> Max[i][1] >> Max[i][2];
      need[i][0] = Max[i][0] - allocation[i][0];
      need[i][1] = Max[i][1] - allocation[i][1];
      need[i][2] = Max[i][2] - allocation[i][2];
    }

    cout << "[19BCE2669] \t\t > Table for Bankers Algo" << endl << endl;
    cout << "Initial Resources: " << current[0] << " " << current[1] << " "
         << current[2] << endl
         << endl;
    cout << "Process    Max    Allocation    Need" << endl;

    for (int i = 0; i < p; ++i) {
      cout << "  P" << i + 1 << "    "
           << "  " << Max[i][0] << " " << Max[i][1] << " " << Max[i][2]
```

```
            << "     "
            << " " << allocation[i][0] << " " << allocation[i][1] << " "
            << allocation[i][2] << "     "
            << " " << need[i][0] << " " << need[i][1] << " " << need[i][2]
            << endl;
    }
  }

  void ProcessData() {
    Calculate();
    while (keepOn) {
      int val, pro;
      cout << "\n\nSelect Below oprations:" << endl << endl;
      cout << "1.Change Max of process: " << endl;
      cout << "2.Change Allocation of process" << endl;
      cout << "3.Change Initial Resources" << endl;
      cout << "4.Exit" << endl;
      cin >> val;

      if (val == 1) {
        cout << "\n\nEnter Process No: ";
        cin >> pro;
        cout << "\nEnter New Max: ";
        cin >> Max[pro - 1][0] >> Max[pro - 1][1] >> Max[pro - 1][2];
      } else if (val == 2) {
        cout << "\n\nEnter Process No: ";
        cin >> pro;
        cout << "\nEnter New Allocation: ";
        cin >> allocation[pro - 1][0] >> allocation[pro - 1][1] >>
            allocation[pro - 1][2];
      } else if (val == 3) {
        cout << "\nEnter Initial Resources: ";
        cin >> current[0] >> current[1] >> current[2];
      } else {
        break;
      }
    }
    Calculate();
  }

  void Calculate() {
    IMP();
    int i, j;
    for (i = 0; i < p; ++i) {
      for (j = 0; j < p; ++j) {
        while (executed[j] && j < p - 1) {
          j++;
        }
        if (need[j][0] <= current[0] && need[j][1] <= current[1] &&
            need[j][2] <= current[2]) {
          if (!executed[j]) {
            executed[j] = true;
            current[0] += allocation[j][0];
            current[1] += allocation[j][1];
```

```
            current[2] += allocation[j][2];
            cout << "\nProcess P" << j + 1;
            cout << "\nCurrent: " << current[0] << " " << current[1] << " "
                 << current[2] << "\n";
            cout << "\nProcess executed without deadlock";
            come = true;
            break;
          }
        }
      }
      if (!come) {
        cout << "\n\t\t\tDead lock\n\n";
        break;
      } else {
        come = false;
      }
    }
  }

  void IMP() {
    come = false;
    for (int i = 0; i < 10; ++i) {
      executed[i] = false;
    }
  }
};

int main() {
    BankerAlgorithm handler = BankerAlgorithm();
    handler.TakeInput();
    handler.ProcessData();
    return 0; }
```

Code Preview

yash@hephaestus: ~

bat BankersAlgorithm.cpp

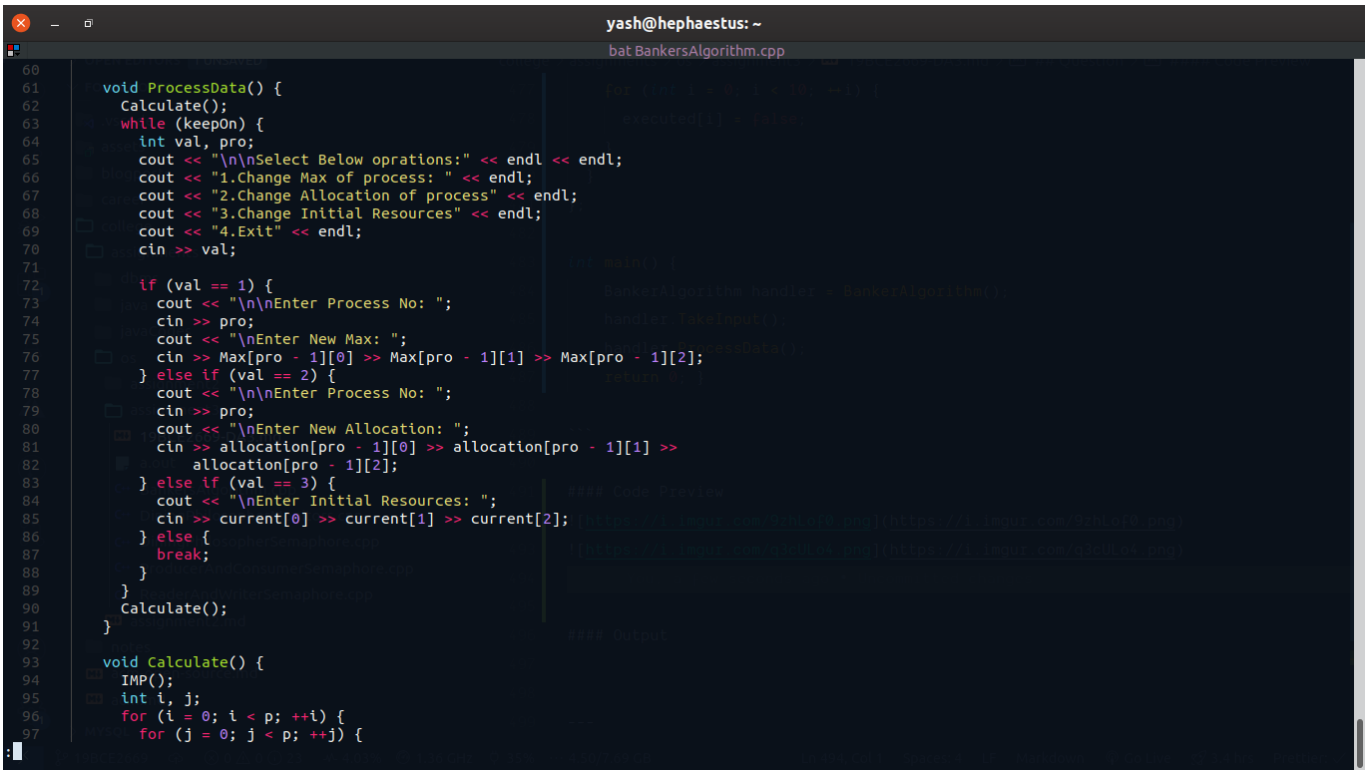
File: BankersAlgorithm.cpp

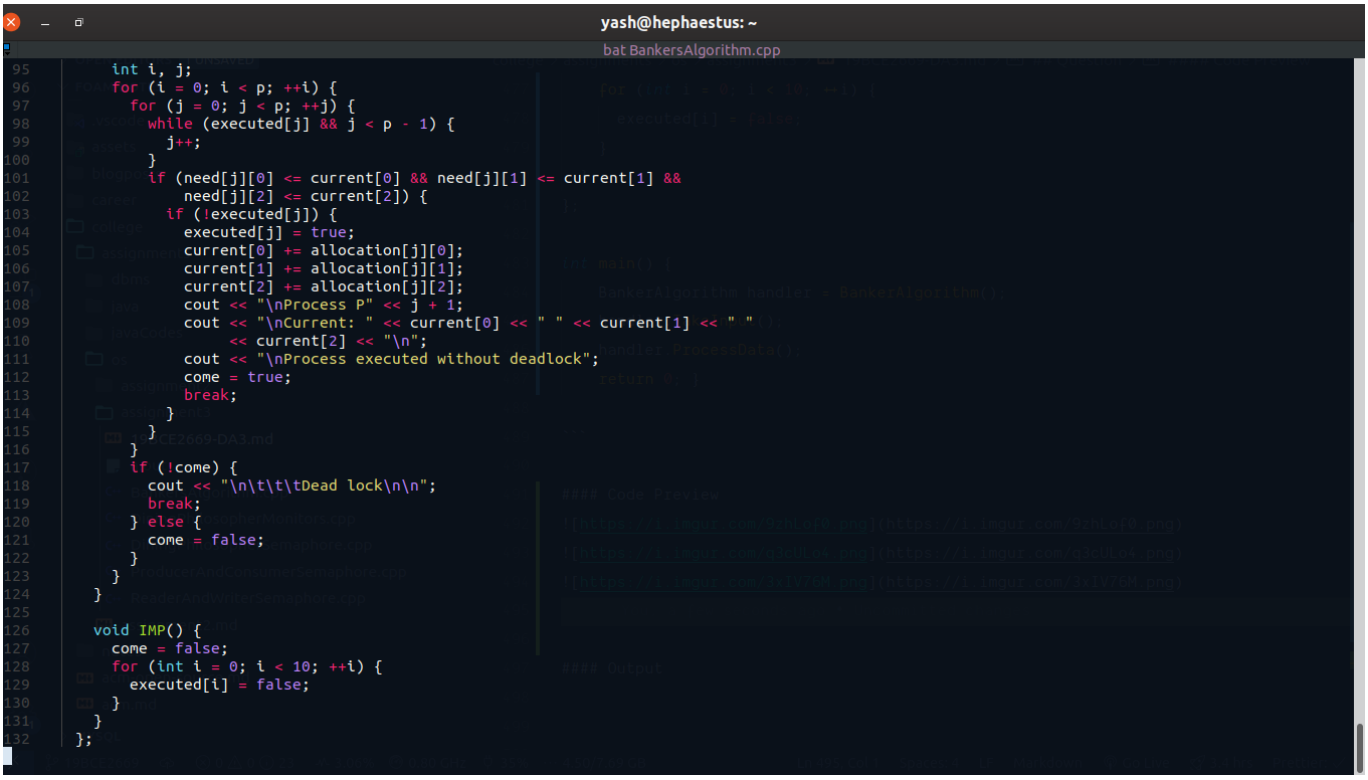
```cpp
/**
 * @author YashKumarVerma / 19BCE2669
 * @for: Lab Digital Assignment 3
 * @description: Implementation of Bankers Algorithm
 */

#include <bits/stdc++.h>
using namespace std;

/** Class to handle all operating and keep data **/
class BankerAlgorithm {
public:
  int p;
  bool come;
  int keepOn;
  int Max[10][3];
  int current[3];
  int need[10][3];
  bool executed[10];
  int available[10][3];
  int allocation[10][3];

  BankerAlgorithm() { int keepOn = 1; }

  void TakeInput() {
    cout << "Enter No. of processes: ";
    cin >> p;
    cout << "\n";
    cout << "Enter the current resources: ";
    cin >> current[0] >> current[1] >> current[2];

    for (int i = 0; i < p; ++i) {
      cout << "[19BCE2669] \t\t > Process P" << i + 1 << " Details" << endl;
      cout << "Enter Allocation : ";
```

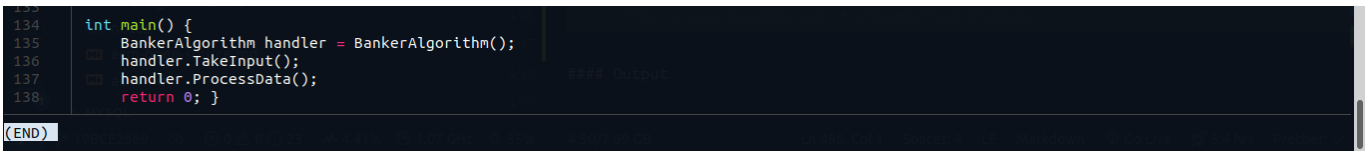
yash@hephaestus: ~

bat BankersAlgorithm.cpp

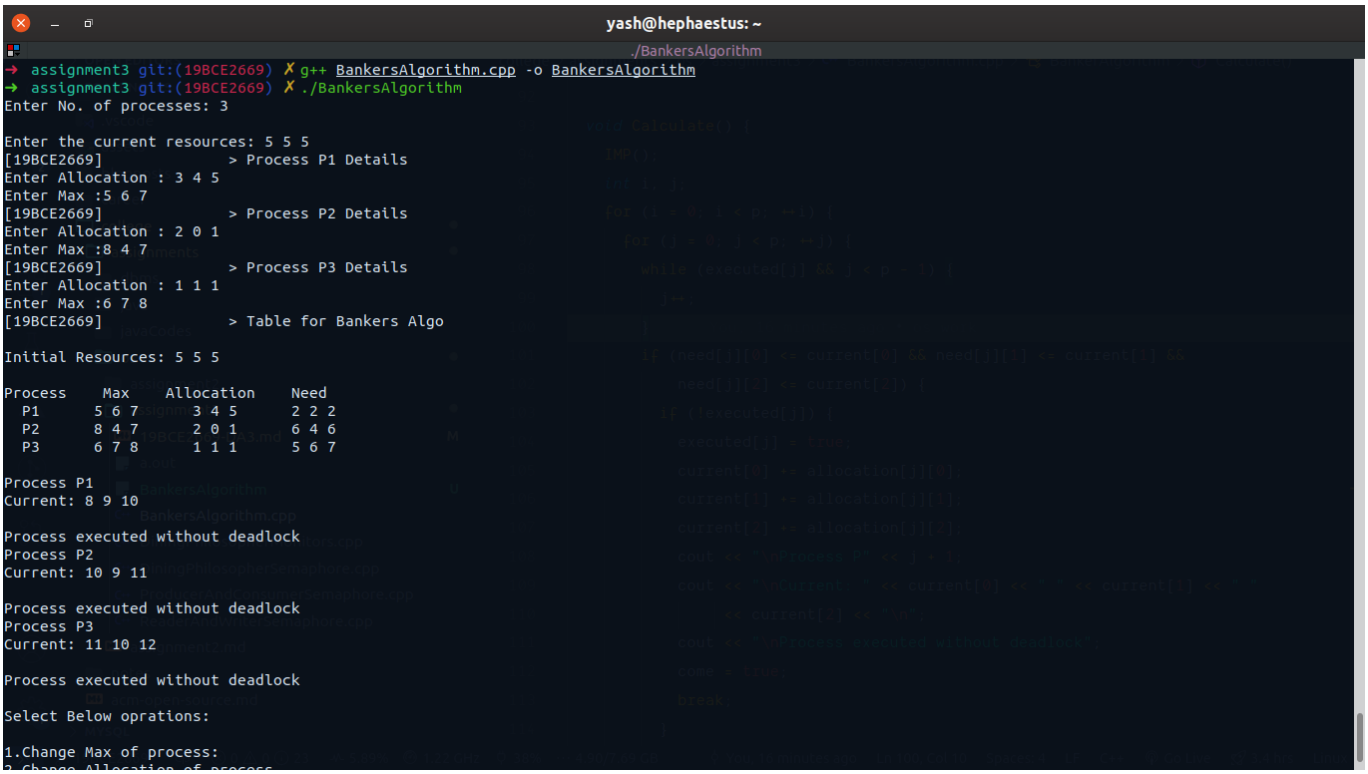
```cpp
      cout << "[19BCE2669] \t\t > Process P" << i + 1 << " Details" << endl;
      cout << "Enter Allocation : ";
      cin >> allocation[i][0] >> allocation[i][1] >> allocation[i][2];
      cout << "Enter Max :";
      cin >> Max[i][0] >> Max[i][1] >> Max[i][2];
      need[i][0] = Max[i][0] - allocation[i][0];
      need[i][1] = Max[i][1] - allocation[i][1];
      need[i][2] = Max[i][2] - allocation[i][2];
    }

    cout << "[19BCE2669] \t\t > Table for Bankers Algo" << endl << endl;
    cout << "Initial Resources: " << current[0] << " " << current[1] << " "
         << current[2] << endl
         << endl;
    cout << "Process    Max    Allocation    Need" << endl;

    for (int i = 0; i < p; ++i) {
      cout << "  P" << i + 1 << "    "
           << "  " << Max[i][0] << " " << Max[i][1] << " " << Max[i][2]
           << "      "
           << " " << allocation[i][0] << " " << allocation[i][1] << " "
           << allocation[i][2] << "      "
           << " " << need[i][0] << " " << need[i][1] << " " << need[i][2]
           << endl;
    }
  }

  void ProcessData() {
    Calculate();
    while (keepOn) {
      int val, pro;
      cout << "\n\nSelect Below oprations:" << endl << endl;
      cout << "1.Change Max of process: " << endl;
      cout << "2.Change Allocation of process" << endl;
      cout << "3.Change Initial Resources" << endl;
      cout << "4.Exit" << endl;
      cin >> val;
```


yash@hephaestus: ~
bat BankersAlgorithm.cpp
void ProcessData() {
Calculate();
while (keepOn) {
int val, pro;
cout << "\n\nSelect Below oprations:" << endl << endl;
cout << "1.Change Max of process: " << endl;
cout << "2.Change Allocation of process" << endl;
cout << "3.Change Initial Resources" << endl;
cout << "4.Exit" << endl;
cin >> val;
if (val == 1) {
cout << "\n\nEnter Process No: ";
cin >> pro;
cout << "\nEnter New Max: ";
cin >> Max[pro - 1][0] >> Max[pro - 1][1] >> Max[pro - 1][2];
} else if (val == 2) {
cout << "\n\nEnter Process No: ";
cin >> pro;
cout << "\nEnter New Allocation: ";
cin >> allocation[pro - 1][0] >> allocation[pro - 1][1] >>
allocation[pro - 1][2];
} else if (val == 3) {
cout << "\nEnter Initial Resources: ";
cin >> current[0] >> current[1] >> current[2];
} else {
break;
}
}
Calculate();
}
void Calculate() {
IMP();
int i, j;
for (i = 0; i < p; ++i) {
for (j = 0; j < p; ++j) {


yash@hephaestus: ~
bat BankersAlgorithm.cpp
int i, j;
for (i = 0; i < p; ++i) {
for (j = 0; j < p; ++j) {
while (executed[j] && j < p - 1) {
j++;
}
if (need[j][0] <= current[0] && need[j][1] <= current[1] &&
need[j][2] <= current[2]) {
if (!executed[j]) {
executed[j] = true;
current[0] += allocation[j][0];
current[1] += allocation[j][1];
current[2] += allocation[j][2];
cout << "\nProcess P" << j + 1;
cout << "\nCurrent: " << current[0] << " " << current[1] << " "
<< current[2] << "\n";
cout << "\nProcess executed without deadlock";
come = true;
break;
}
}
}
if (!come) {
cout << "\n\t\t\tDead lock\n\n";
break;
} else {
come = false;
}
}
}
void IMP() {
come = false;
for (int i = 0; i < 10; ++i) {
executed[i] = false;
}
}
};


int main() {
BankerAlgorithm handler = BankerAlgorithm();
handler.TakeInput();
handler.ProcessData();
return 0; }
(END)

Output



Question

Implementing Producers and Consumers Problem using Semaphore

Code

```
/**
 * @author YashKumarVerma / 19BCE2669
 * @for: Lab Digital Assignment 3
 * @description: Implementation of producer consumer problem using
semaphores
 */

/** defining constants for application **/
#define BUFFER_SIZE 10

/** Including Required headers **/
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <stdio.h>

using namespace std;

/**
 * @class Config
 * @description To store code configurations and local state
 **/
class Config {
```

```
public:
    static int MaxItems;
    static int BufferSize;
    static sem_t empty;
    static sem_t full;
    static int in;
    static int out;
    static int buffer[BUFFER_SIZE];
    static pthread_mutex_t mutex;
};

/** Initializing All App state **/
int Config::MaxItems = 10;
int Config::BufferSize = BUFFER_SIZE;
sem_t Config::empty;
sem_t Config::full;
int Config::in = 0;
int Config::out = 0;
int Config::buffer[BUFFER_SIZE];
pthread_mutex_t Config::mutex;

/**
 * @class Semaphore
 * @description Class to contain definitions for Producer and Consumer
 **/
class Semaphore {
public:
    /* comsumer process */
    static void* Consumer(void* cno) {
        for (int i = 0; i < Config::MaxItems; i++) {
            sem_wait(&Config::full);
            pthread_mutex_lock(&Config::mutex);
            int item = Config::buffer[Config::out];
            printf("Consumer %d: Remove Item %d from %d\n", *((int*)cno),
item, Config::out);
            Config::out = (Config::out + 1) % Config::BufferSize;
            pthread_mutex_unlock(&Config::mutex);
            sem_post(&Config::empty);
        }
    }

    /* producer process */
    static void* Producer(void* pno) {
        int item;
        for (int i = 0; i < Config::MaxItems; i++) {
            item = rand(); // Produce an random item
            sem_wait(&Config::empty);
            pthread_mutex_lock(&Config::mutex);
            Config::buffer[Config::in] = item;
            printf("Producer %d: Insert Item %d at %d\n", *((int*)pno),
Config::buffer[Config::in], Config::in);
            Config::in = (Config::in + 1) % Config::BufferSize;
            pthread_mutex_unlock(&Config::mutex);
            sem_post(&Config::full);
```

```
            }
        }
    };

    /* main caller function */
    int main(){

        pthread_t pro[5], con[5];
        pthread_mutex_init(&Config::mutex, NULL);
        sem_init(&Config::empty, 0, Config::BufferSize);
        sem_init(&Config::full, 0, 0);

        /* Just used for numbering the producer and consumer */
        int a[5] = { 1, 2, 3, 4, 5 };
        for (int i = 0; i < 5; i++) {
            pthread_create(&pro[i], NULL, Semaphore::Producer, (void*)&a[i]);
        }
        for (int i = 0; i < 5; i++) {
            pthread_create(&con[i], NULL, Semaphore::Consumer, (void*)&a[i]);
        }

        for (int i = 0; i < 5; i++) {
            pthread_join(pro[i], NULL);
        }
        for (int i = 0; i < 5; i++) {
            pthread_join(con[i], NULL);
        }

        pthread_mutex_destroy(&Config::mutex);
        sem_destroy(&Config::empty);
        sem_destroy(&Config::full);

        return 0;
    }
```

Code Preview

yash@hephaestus: ~

bat ProducerAndConsumerSemaphore.cpp

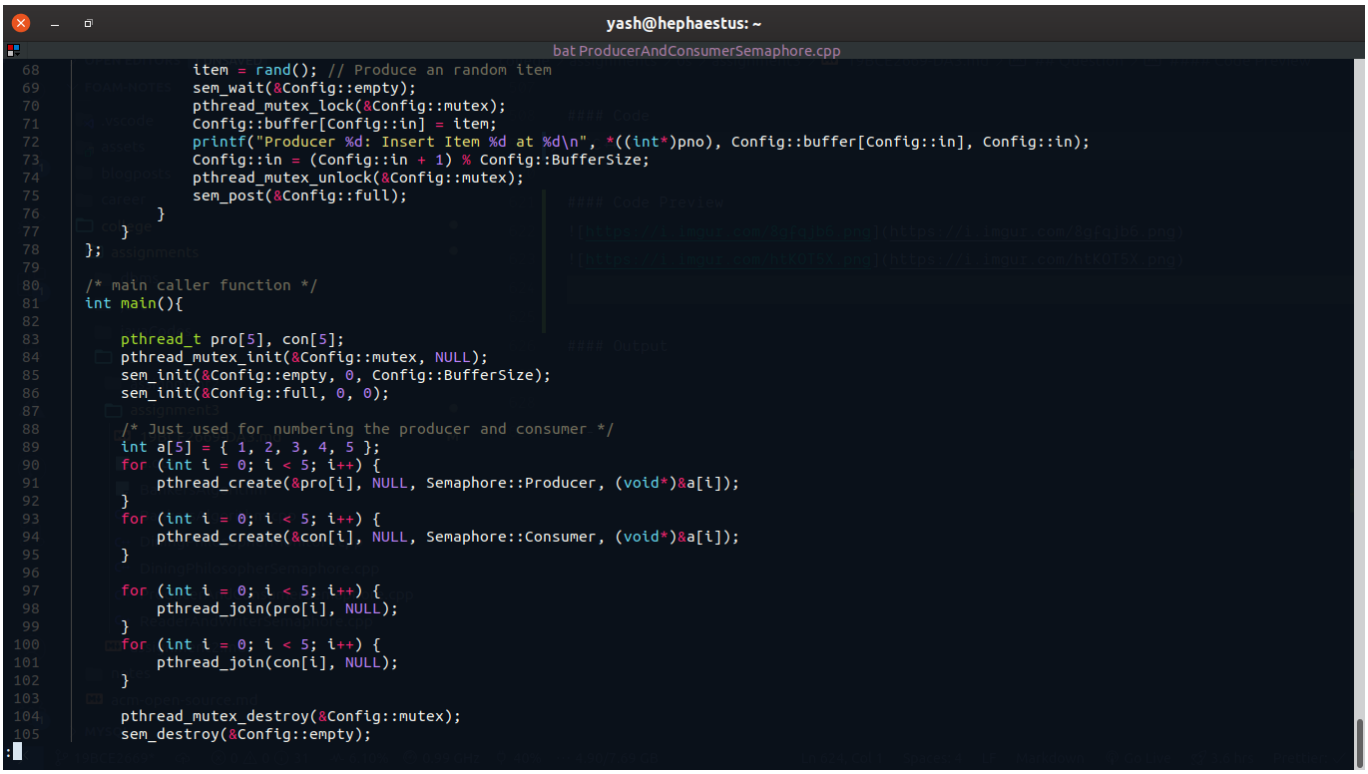
File: ProducerAndConsumerSemaphore.cpp

```cpp
/**
 * @author YashKumarVerma / 19BCE2669
 * @for: Lab Digital Assignment 3
 * @description: Implementation of producer consumer problem using semaphores
 */

/** defining constants for application **/
#define BUFFER_SIZE 10

/** Including Required headers **/
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <stdio.h>

using namespace std;

/**
 * @class Config
 * @description To store code configurations and local state
 **/
class Config {
public:
    static int MaxItems;
    static int BufferSize;
    static sem_t empty;
    static sem_t full;
    static int in;
    static int out;
    static int buffer[BUFFER_SIZE];
    static pthread_mutex_t mutex;
};

/** Initializing All App state **/
```

yash@hephaestus: ~

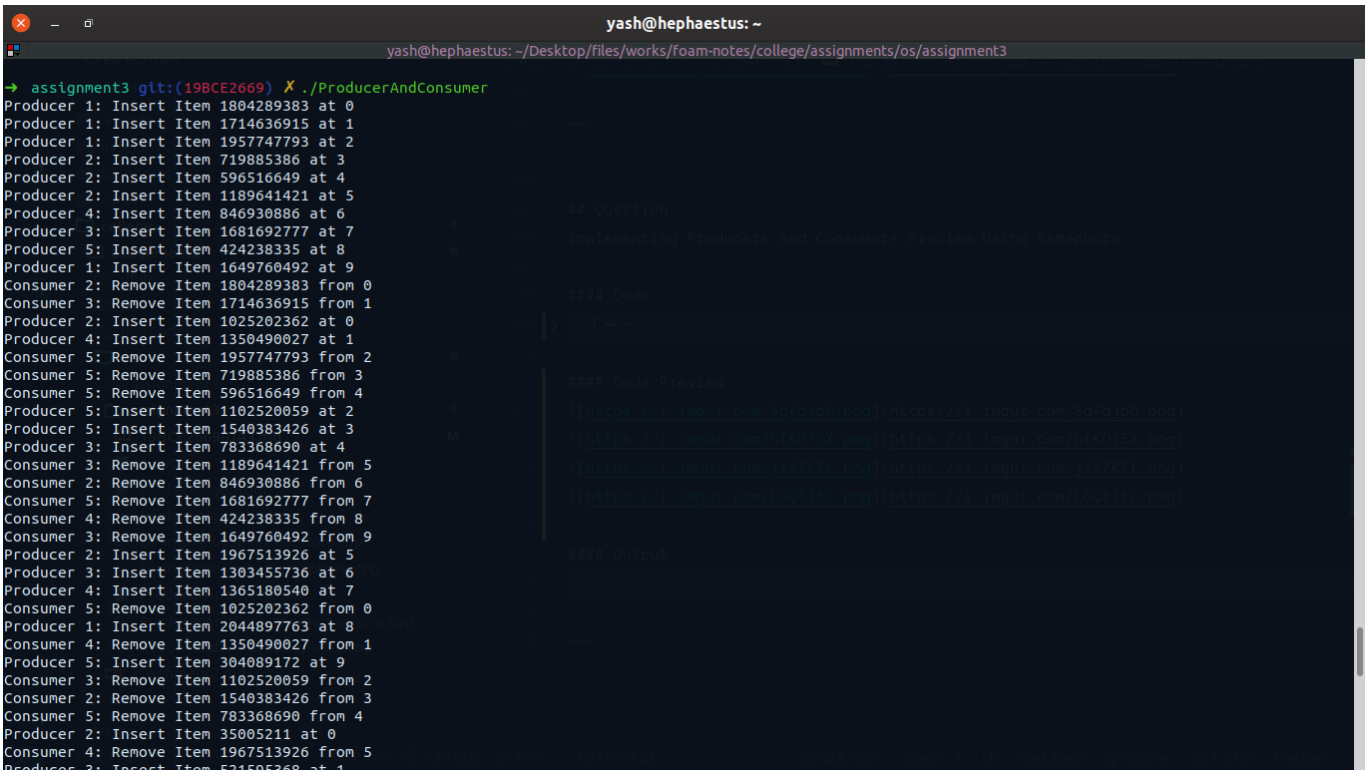
bat ProducerAndConsumerSemaphore.cpp

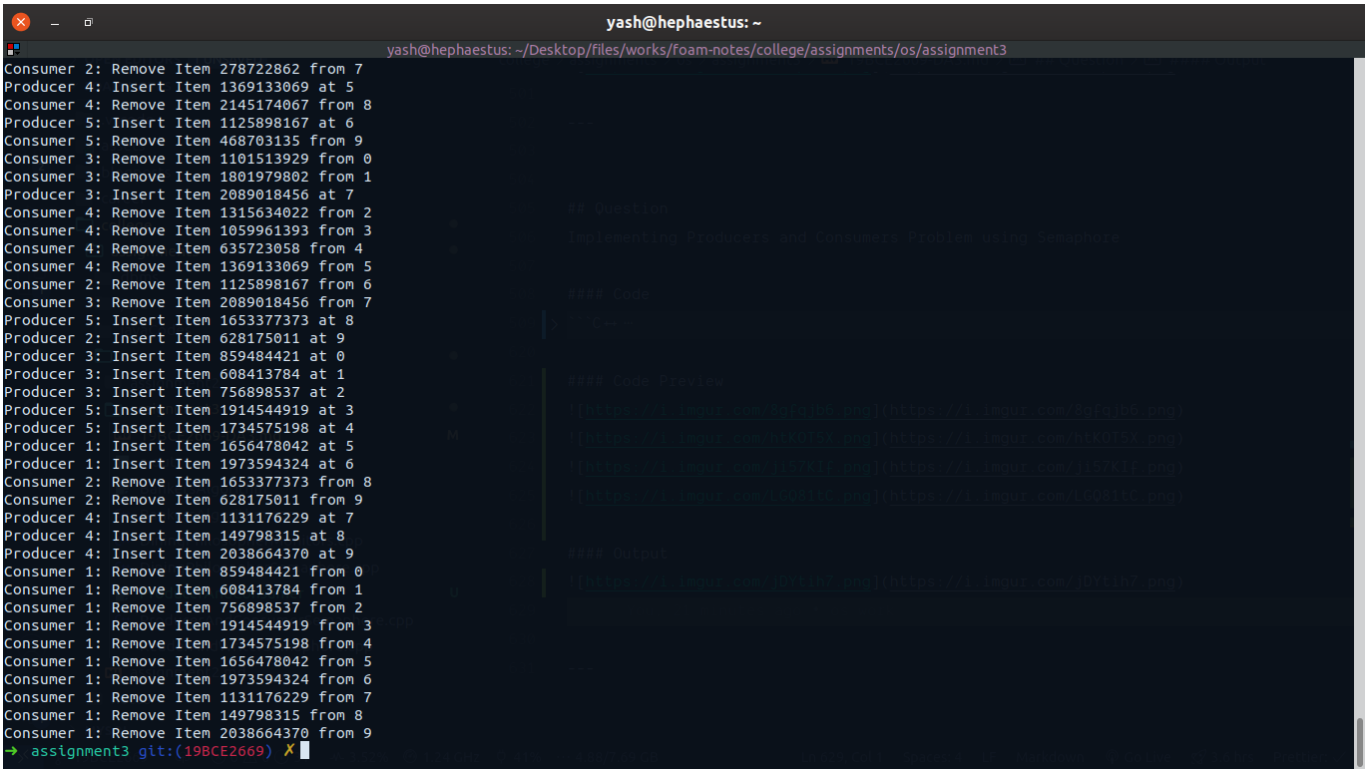
```cpp

/** Initializing All App state **/
int Config::MaxItems = 10;
int Config::BufferSize = BUFFER_SIZE;
sem_t Config::empty;
sem_t Config::full;
int Config::in = 0;
int Config::out = 0;
int Config::buffer[BUFFER_SIZE];
pthread_mutex_t Config::mutex;

/**
 * @class Semaphore
 * @description Class to contain definitions for Producer and Consumer
 **/
class Semaphore {
public:
    /* comsumer process */
    static void* Consumer(void* cno) {
        for (int i = 0; i < Config::MaxItems; i++) {
            sem_wait(&Config::full);
            pthread_mutex_lock(&Config::mutex);
            int item = Config::buffer[Config::out];
            printf("Consumer %d: Remove Item %d from %d\n", *((int*)cno), item, Config::out);
            Config::out = (Config::out + 1) % Config::BufferSize;
            pthread_mutex_unlock(&Config::mutex);
            sem_post(&Config::empty);
        }
    }

    /* producer process */
    static void* Producer(void* pno) {
        int item;
        for (int i = 0; i < Config::MaxItems; i++) {
            item = rand(); // Produce an random item
            sem_wait(&Config::empty);
            pthread_mutex_lock(&Config::mutex);
            Config::buffer[Config::in] = item;
```


yash@hephaestus: ~
bat ProducerAndConsumerSemaphore.cpp
item = rand(); // Produce an random item
sem_wait(&Config::empty);
pthread_mutex_lock(&Config::mutex);
Config::buffer[Config::in] = item;
printf("Producer %d: Insert Item %d at %d\n", *((int*)pno), Config::buffer[Config::in], Config::in);
Config::in = (Config::in + 1) % Config::BufferSize;
pthread_mutex_unlock(&Config::mutex);
sem_post(&Config::full);
}
}
};
/* main caller function */
int main(){
pthread_t pro[5], con[5];
pthread_mutex_init(&Config::mutex, NULL);
sem_init(&Config::empty, 0, Config::BufferSize);
sem_init(&Config::full, 0, 0);
/* Just used for numbering the producer and consumer */
int a[5] = { 1, 2, 3, 4, 5 };
for (int i = 0; i < 5; i++) {
pthread_create(&pro[i], NULL, Semaphore::Producer, (void*)&a[i]);
}
for (int i = 0; i < 5; i++) {
pthread_create(&con[i], NULL, Semaphore::Consumer, (void*)&a[i]);
}
for (int i = 0; i < 5; i++) {
pthread_join(pro[i], NULL);
}
for (int i = 0; i < 5; i++) {
pthread_join(con[i], NULL);
}
pthread_mutex_destroy(&Config::mutex);
sem_destroy(&Config::empty);


};
/* main caller function */
int main(){
pthread_t pro[5], con[5];
pthread_mutex_init(&Config::mutex, NULL);
sem_init(&Config::empty, 0, Config::BufferSize);
sem_init(&Config::full, 0, 0);
/* Just used for numbering the producer and consumer */
int a[5] = { 1, 2, 3, 4, 5 };
for (int i = 0; i < 5; i++) {
pthread_create(&pro[i], NULL, Semaphore::Producer, (void*)&a[i]);
}
for (int i = 0; i < 5; i++) {
pthread_create(&con[i], NULL, Semaphore::Consumer, (void*)&a[i]);
}
for (int i = 0; i < 5; i++) {
pthread_join(pro[i], NULL);
}
for (int i = 0; i < 5; i++) {
pthread_join(con[i], NULL);
}
pthread_mutex_destroy(&Config::mutex);
sem_destroy(&Config::empty);
sem_destroy(&Config::full);
return 0;
}
(END)

Output





Question

Implement Readers and Writers Problem using Semaphores

Code

```
/**
 * @author YashKumarVerma / 19BCE2669
 * @for: Lab Digital Assignment 3
```

```
 * @description: Implementation of readers-writers problem using semaphores
 */

#include<iostream>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

using namespace std;

sem_t mutex;
sem_t wrt;
int data = 0;
int readCount = 0;

/**
 * @class Semaphore
 * @description: to define reader and writer behavior
 */
class Semaphore{
    public:


    static void *Reader(void *arg){
        int f = ((intptr_t)arg);

        sem_wait(&mutex);
        readCount += 1;
        if(readCount == 1){
            sem_wait(&wrt);
        }

        sem_post(&mutex);
        std::cout << "Data read by the reader : " << f << " is " << data <<
std::endl;
        sleep(1);

        sem_wait(&mutex);
        readCount -= 1;
        if(readCount == 0){
            sem_post(&wrt);
        }
        sem_post(&mutex);
    }

    static void *Writer(void *arg){
        int f = ((intptr_t)arg);
        sem_wait(&wrt);
        data++;
        std::cout << "Data written by the writer : " << f << " is " << data
<< std::endl;
        sleep(1);
        sem_post(&wrt);
```

```
    }
};

/** Main caller function */
int main(){

    int i, b;
    pthread_t readerThreadId[8], writerThreadId[8];
    sem_init(&mutex, 0, 1);
    sem_init(&wrt, 0, 1);
    for (i=0;i<=5;i++){
        pthread_create(&writerThreadId[i], NULL, Semaphore::Writer, (void
*)(intptr_t)i);
        pthread_create(&readerThreadId[i], NULL, Semaphore::Reader, (void
*)(intptr_t)i);
    }
    for (i=0;i<=5;i++){
        pthread_join(writerThreadId[i], NULL);
        pthread_join(readerThreadId[i], NULL);
    }
    return 0;
}
```

Code Preview

yash@hephaestus: ~

bat ReaderAndWriterSemaphore.cpp

File: ReaderAndWriterSemaphore.cpp

```cpp
/**
 * @author YashKumarVerma / 19BCE2669
 * @for: Lab Digital Assignment 3
 * @description: Implementation of readers-writers problem using semaphores
 */

#include<iostream>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

using namespace std;

sem_t mutex;
sem_t wrt;
int data = 0;
int readCount = 0;

/**
 * @class Semaphore
 * @description: to define reader and writer behavior
 */
class Semaphore{
    public:


    static void *Reader(void *arg){
        int f = ((intptr_t)arg);

        sem_wait(&mutex);
        readCount += 1;
        if(readCount == 1){
            sem_wait(&wrt);
        }
```

yash@hephaestus: ~

bat ReaderAndWriterSemaphore.cpp

```cpp
            sem_wait(&wrt);
        }

        sem_post(&mutex);
        std::cout << "Data read by the reader : " << f << " is " << data << std::endl;
        sleep(1);

        sem_wait(&mutex);
        readCount -= 1;
        if(readCount == 0){
            sem_post(&wrt);
        }
        sem_post(&mutex);
    }

    static void *Writer(void *arg){
        int f = ((intptr_t)arg);
        sem_wait(&wrt);
        data++;
        std::cout << "Data written by the writer : " << f << " is " << data << std::endl;
        sleep(1);
        sem_post(&wrt);
    }
};

/** Main caller function */
int main(){

    int i, b;
    pthread_t readerThreadId[8], writerThreadId[8];
    sem_init(&mutex, 0, 1);
    sem_init(&wrt, 0, 1);
    for (i=0;i<=5;i++){
        pthread_create(&writerThreadId[i], NULL, Semaphore::Writer, (void *)(intptr_t)i);
        pthread_create(&readerThreadId[i], NULL, Semaphore::Reader, (void *)(intptr_t)i);
    }
    for (i=0;i<=5;i++){
        pthread_join(writerThreadId[i], NULL);
```

```cpp
/** Main caller function */
int main(){

    int i, b;
    pthread_t readerThreadId[8], writerThreadId[8];
    sem_init(&mutex, 0, 1);
    sem_init(&wrt, 0, 1);
    for (i=0;i<=5;i++){
        pthread_create(&writerThreadId[i], NULL, Semaphore::Writer, (void *)(intptr_t)i);
        pthread_create(&readerThreadId[i], NULL, Semaphore::Reader, (void *)(intptr_t)i);
    }
    for (i=0;i<=5;i++){
        pthread_join(writerThreadId[i], NULL);
        pthread_join(readerThreadId[i], NULL);
    }
    return 0;
}
```

(END)

Output

Time dependent Execution