Read Me

Assignment3-Ques1

Part 1

DiningPhilosopher using Semaphores

Global Variables & Thread Creation

```
pthread_t philosophers[5];
sem_t sema[5];
int global[5]={0,0,0,0,0};
int forks[5]={0,1,2,3,4};
int phil[5]={1,2,3,4,5};
int left(int a){
    return forks[a-1];
}
```

Global[5] represents the shared global variable we wish to control the access for.

```
int main(int argc,char **argv){
    for(int i=0;i<5;i++){
        sem_init(&sema[i],0,1);
    }
    for(int i=0;i<5;i++){
        if(pthread_create(&philosophers[i],NULL,&Philosopher,&phil[i])!=0){
            perror("Error creating thread");
        }
    }
    for(int i=0;i<5;i++){
        if(pthread_join(philosophers[i],NULL)!=0){
            perror("Error joining thread");
        }
    }
```

Solution using semaphores

```
int take_fork(int a){
    if(a==5){
        sem_wait(&sema[right(a)]);
        sem_wait(&sema[left(a)]);
    }
    else{
        sem_wait(&sema[left(a)]);
        sem_wait(&sema[right(a)]);
    }
    printf("%d philosopher takes forks %d and %d\n",a,left(a),right(a));
```

Output

```
philosopher 4 is eating with forks 3 and 4
1 philosopher takes forks 0 and 1
philosopher 1 is eating with forks 0 and 1
4 philosopher keeps forks 3 and 4
3 philosopher takes forks 2 and 3
philosopher 3 is eating with forks 2 and 3
5 philosopher keeps forks 4 and 0
1 philosopher keeps forks 0 and 1
3 philosopher keeps forks 2 and 3
4 philosopher takes forks 3 and 4
philosopher 4 is eating with forks 3 and 4
4 philosopher keeps forks 3 and 4
2 philosopher takes forks 1 and 2
philosopher 2 is eating with forks 1 and 2
4 philosopher takes forks 3 and 4
philosopher 4 is eating with forks 3 and 4
2 philosopher keeps forks 1 and 2
4 philosopher keeps forks 3 and 4
3 philosopher takes forks 2 and 3
philosopher 3 is eating with forks 2 and 3
5 philosopher takes forks 4 and 0
philosopher 5 is eating with forks 4 and 0
3 philosopher keeps forks 2 and 3
1 philosopher takes forks 0 and 1
philosopher 1 is eating with forks 0 and 1
5 philosopher keeps forks 4 and 0
1 philosopher keeps forks 0 and 1
5 philosopher takes forks 4 and 0
philosopher 5 is eating with forks 4 and 0
2 philosopher takes forks 1 and 2
philosopher 2 is eating with forks 1 and 2
2 philosopher keeps forks 1 and 2
5 philosopher keeps forks 4 and 0
5 philosopher takes forks 4 and 0
philosopher 5 is eating with forks 4 and 0
5 philosopher keeps forks 4 and 0
Use of the forks 10 10 10 10 10
[artix Assignment3 Ques1]#
```

Output shows how two philosophers are unable to access the same fork together.

Part2

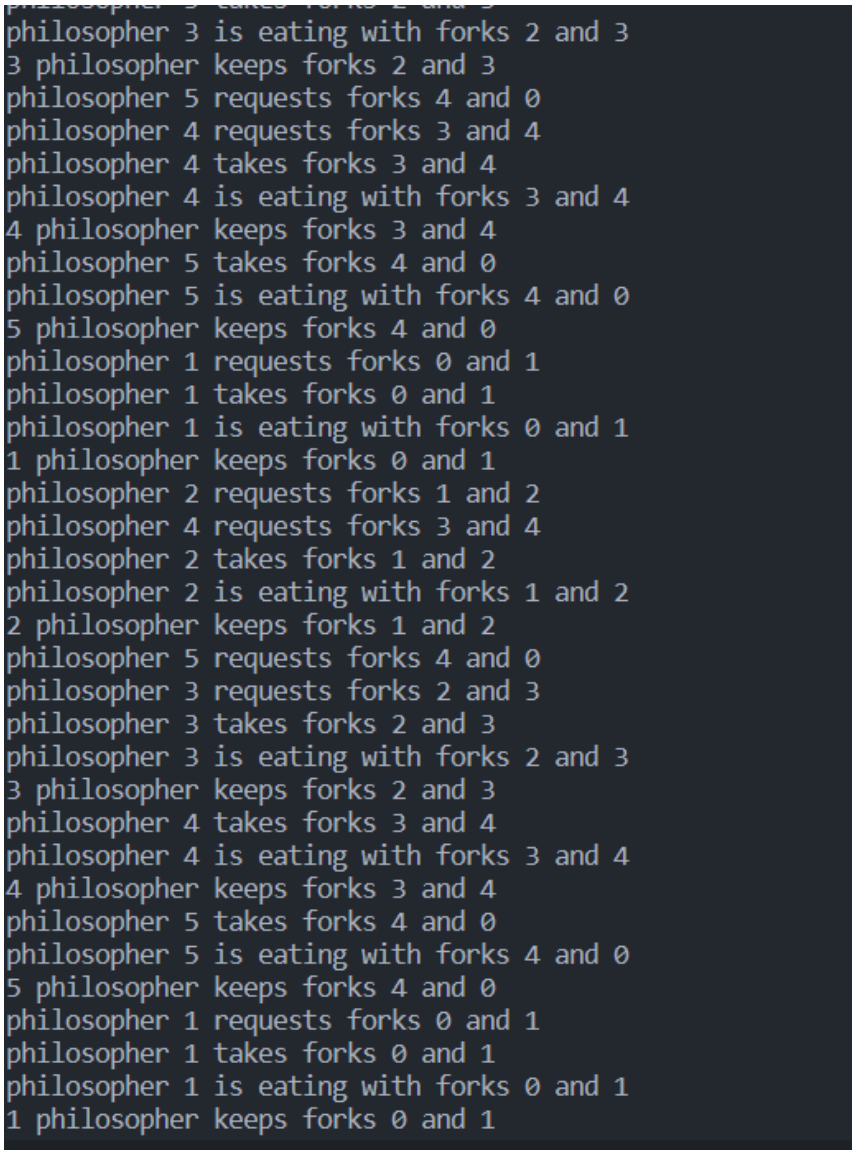
Dining Philosopher using Strict Ordering of Resource Requests

```
bool calcCount(int a){
    for(int i=0;i<a-1;i++){
        if(count[i]<=count[a-1]){
            return false;
        }
    }
    return true;
}
```

This function models a spin lock and ensures that a philosopher eats only if a philosopher of lower number has eaten at least as much as it has. Thus it places philosophers with lower numbers at higher priority. Thus the philosophers always eat strictly in the order 1,2,3,4,5.

```
    int i=0;
    while(i<5){
        think();
        printf("philosopher %d requests forks %d and %d\n",num,left(num),right(num));
        while(!calcCount(num)){
            sleep(0);
        }
        printf("philosopher %d takes forks %d and %d\n",num,left(num),right(num));
        eat(num);
        i++;
        printf("%d philosopher keeps forks %d and %d\n",num,left(num),right(num));
    }
}
```

Output:



While a higher number philosopher may request for a fork the request will not be granted unless the philosopher below it has eaten strictly. This allows us to order all philosophers.

Part 3

Sauce Problem

```
}
int main(int argc,char **argv){
    sem_init(&sauce,0,2);
    for(int i=0;i<5;i++){
        sem_init(&sema[i],0,1);
    }
```

The semaphore sauce is a counting semaphore initialized to 2 allowing two threads to access it at once. This semaphore symbolizes the 2 sauce bowls.Sauce is taken after forks are acquired and released before forks are released. This is to ensure that there are deadlocks.

```
    int i=0;
    while(i<5){
        think();
        take_fork(a);
        take_sauce(a);
        eat(a);
        keep_sauce(a);
        keep_fork(a);
        i++;
    }
```

Output

```
philosopher 3 is eating with forks 2 and 3 with sauce
3 philosopher keeps sauce
3 philosopher keeps forks 2 3
philosopher 1 is eating with forks 0 and 1 with sauce
1 philosopher keeps sauce
1 philosopher keeps forks 0 1
5 philosopher takes forks 4 and 0
5 philosopher takes sauce
2 philosopher takes forks 1 and 2
2 philosopher takes sauce
philosopher 2 is eating with forks 1 and 2 with sauce
2 philosopher keeps sauce
2 philosopher keeps forks 1 2
philosopher 5 is eating with forks 4 and 0 with sauce
5 philosopher keeps sauce
5 philosopher keeps forks 4 0
1 philosopher takes forks 0 and 1
1 philosopher takes sauce
philosopher 1 is eating with forks 0 and 1 with sauce
1 philosopher keeps sauce
1 philosopher keeps forks 0 1
2 philosopher takes forks 1 and 2
2 philosopher takes sauce
5 philosopher takes forks 4 and 0
5 philosopher takes sauce
philosopher 2 is eating with forks 1 and 2 with sauce
```

We can see two threads eating at the same time by accessing the sauce semaphore at the same time.