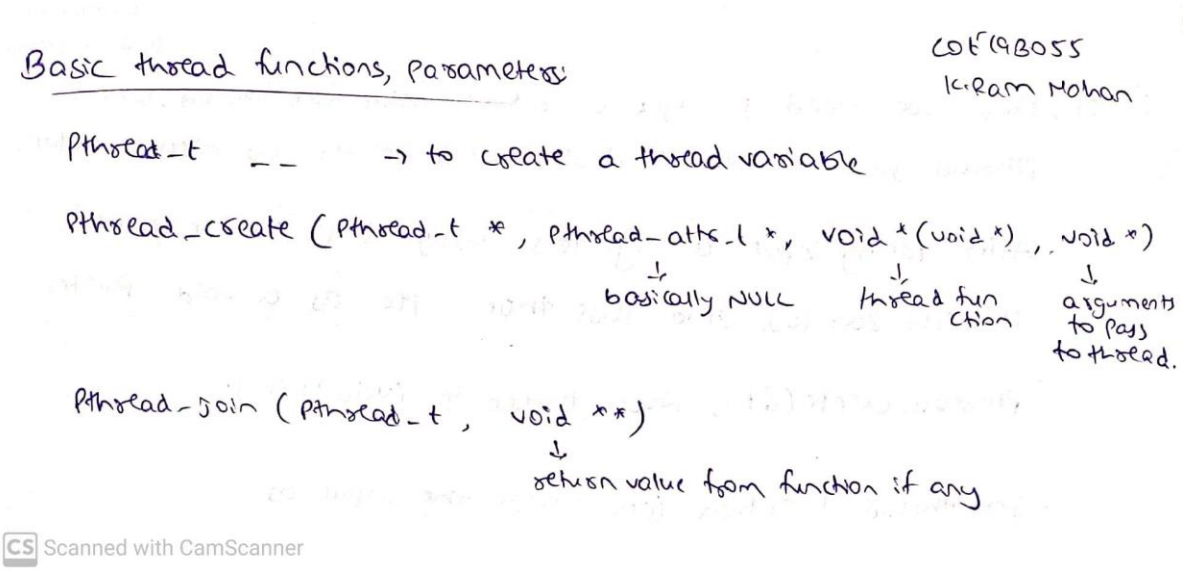
OS_Lab9

K. Ram Mohan

COE19B055



Q1)

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

int sum[5];

void *thread_fn(void *arg)
{
        int id= (int *) arg;
        int start = (id)*200;
        start = start+1;
        int i=0;
```

```
        while(i<200)
        {
                sum[id]+=(i+start);
                i++;
        }
        return NULL;
}

int main()
{
        pthread_t t1, t2, t3, t4, t5;

        pthread_create(&t1, NULL, thread_fn, (void *)0);
        pthread_create(&t2, NULL, thread_fn, (void *)1);
        pthread_create(&t3, NULL, thread_fn, (void *)2);
        pthread_create(&t4, NULL, thread_fn, (void *)3);
        pthread_create(&t5, NULL, thread_fn, (void *)4);

        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
        pthread_join(t3, NULL);
        pthread_join(t4, NULL);
        pthread_join(t5, NULL);

        printf("Sum of numbers from   1 to  200 calculated by thread 1 is %d\n", sum[0]);
        printf("Sum of numbers from 200 to  400 calculated by thread 2 is %d\n", sum[1]);
        printf("Sum of numbers from 400 to  600 calculated by thread 3 is %d\n", sum[2]);
        printf("Sum of numbers from 600 to  800 calculated by thread 4 is %d\n", sum[3]);
        printf("Sum of numbers from 800 to 1000 calculated by thread 5 is %d\n", sum[4]);
```

```
    printf("Sum of all 1000 numbers is %d\n", sum[0]+sum[1]+sum[2]+sum[3]+sum[4]);
    return 0;
}
```

K.Ram Mohan
COE19B055

1) Asked to create 5 threads and add 0 to 1000, 200 in each thread.

For that we can pass an argument in pthread_create which can say the starting number to add and in the function we can add the next 200 numbers and store result in a global variable. And print it in main function.

WKT Pthread function accepts only (void *) as inputs. we need to type cast it to integer.

eg:- Pthread_create (&t1, NULL, thread_fn, (void *) 0);

Output:

```
ram@ram:~/Documents/OS/Lab9$ ./lab9_q1
Sum of numbers from   1 to  200 calculated by thread 1 is 20100
Sum of numbers from 200 to  400 calculated by thread 2 is 60100
Sum of numbers from 400 to  600 calculated by thread 3 is 100100
Sum of numbers from 600 to  800 calculated by thread 4 is 140100
Sum of numbers from 800 to 1000 calculated by thread 5 is 180100
Sum of all 1000 numbers is 500500
ram@ram:~/Documents/OS/Lab9$
```

Q2)

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

void *thread_fn(void* arg)
{
        int *end = (int*) arg;
        int i=0;

        printf("In thread\n");
        while(i<=*end)
        {
                printf("%d\n", i);
                i++;
        }
        return NULL;
}

int main()
{
        int n;
        printf("In main- Enter n: ");
        scanf("%d", &n);

        pthread_t t1, t2, t3, t4;
```

```
    pthread_create(&t1, NULL, thread_fn, (void *)&n);

    pthread_join(t1, NULL);
    return 0;
}
```

COE19B055
K.Ram Mohan

2) Here also we need to pass 'n' value to thread function and print 0 to n in thread.

Since n is a variable we need to pass it as a reference void pointer

ie,

```
pthread_create(&t1, NULL, thread_fn, (void *) &n);
```

In thread

```
int *end = (int *) arg;
```

Now we can do a while (i<=*end) and print 'i'.

Output:

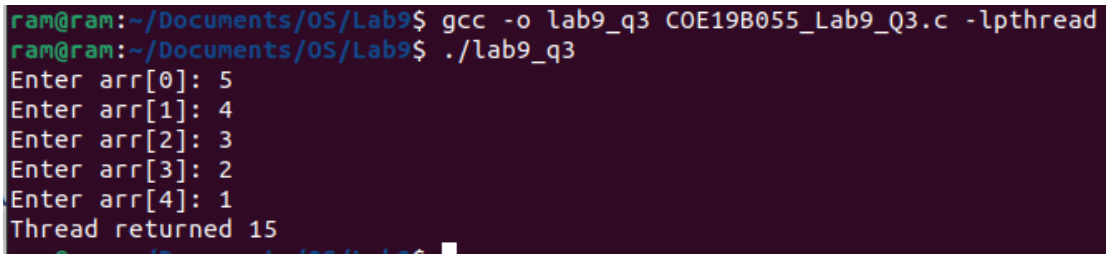
```
ram@ram:~/Documents/OS/Lab9$ gcc -o lab9_q2 COE19B055_Lab9_Q2.c -lpthread
ram@ram:~/Documents/OS/Lab9$ ./lab9_q2
In main- Enter n: 5
In thread
0
1
2
3
4
5
```

Q3)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<string.h>

void *thread_function(void *arg)
{
        int *ptr = (int *)arg;
        int sum=0, n ,i;
        int *result = malloc(sizeof(int));

        for(i=0; i<5; i++){
                sum = sum + (*(ptr+i));
        }

        *result = sum;
        pthread_exit((void *)result);
        //pthread_exit((void *)&sum);
        //we can't directly pass local variable refernece because it will be removed from stack once funcion is returned
}

int main()
{
        int arr[5], i, *ptr;
```

```
    for(i=0; i<5; i++){
        printf("Enter arr[%d]: ", i);
        scanf("%d", &arr[i]);
    }

    ptr = &arr[0];

    pthread_t a_thread;
    int *sum;

    pthread_create(&a_thread, NULL, thread_function, (void *)ptr);
    //pthread_join expects void ** so we are passing reference of sum which is a pointer results in double pointer, then type casting to void
    pthread_join(a_thread, (void **)&sum);

    printf("Thread returned %d\n", *(int *)sum);
    free(sum);
    return 0;
}
```

Output:



3). Here we need to get a return value from thread, where pthread_join accepts a double void pointer as return value.

- After taking input array. lets assign it to an int pointer.

  int *ptr = &arr[0]; And pass that ptr as a void pointer.

  pthread_create(&t1, NULL, thread_fn, (void *) ptr);

- In thread function lets accept the input as

  int *ptr = (int *) arg; and do computation and add all elements in ptr. And store it in " int sum".

- If we return pthread_exit((void *)&sum); . since sum is a reference, double void pointer

  is a local variable it will be removed from stack. once we move out from fn and when we access the reference in main nothing will be displayed.

- So we will assign sum to a malloc variable of int datatype,

```
int *result = malloc(sizeof(int));
*result = sum;
pthread_exit((void *) result);
```

result: itself a pointer; (void *): double void pointer

  Now we can use this one reference in main as malloc will be in heap

- We must free the malloc variable once we used it. in main.

Q4)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>
#include<time.h>

sem_t full;
sem_t empty;
sem_t s;
int buffer[5];
int in=0, out=0;

void *producer()
{
        int item, i;
        for(i=0; i<5; i++)
        {
                item = rand()%10;
                sem_wait(&empty);
                sem_wait(&s);

                buffer[in]=item;
                printf("Producer inserted item %d at %d\n", item, in);
                in = (in+1)%5;

                sem_post(&s);
```

```
                sem_post(&full);
        }
        return 0;
}

void *consumer()
{
        int item;
        for(int i=0; i<5; i++)
        {
                sem_wait(&full);
                sem_wait(&s);

                item = buffer[out];
                printf("Consumer inserted item %d from position %d\n", item, out);
                out = (out+1)%5;

                sem_post(&s);
                sem_post(&empty);
        }
}

int main()
{
        srand(time(NULL));
        sem_init(&empty, 0, 5);
        sem_init(&full, 0, 0);
        sem_init(&s, 0, 1);
```

```
    pthread_t p, c;

    pthread_create(&p, NULL, producer, NULL);
    pthread_create(&c, NULL, consumer, NULL);

    pthread_join(p, NULL);
    pthread_join(c, NULL);
}
```

CSE19B055
K. Ram Mohan

4) We need to create a produce and consumer.
let us assume size of buffer as '5'.

. To create locks for critical section let us use semaphore.

→ Sem_t name :- is used to create semaphore variable

. initially empty will be "5"
full will be "0"
and a variable for allowing either producer or consumer only one at a time into critical section let us keep that initially values as "1".

. we will create two threads
i) one for producer function → which will add items to buffer
ii) one for consumer function → which will remove items from buffer

Output:

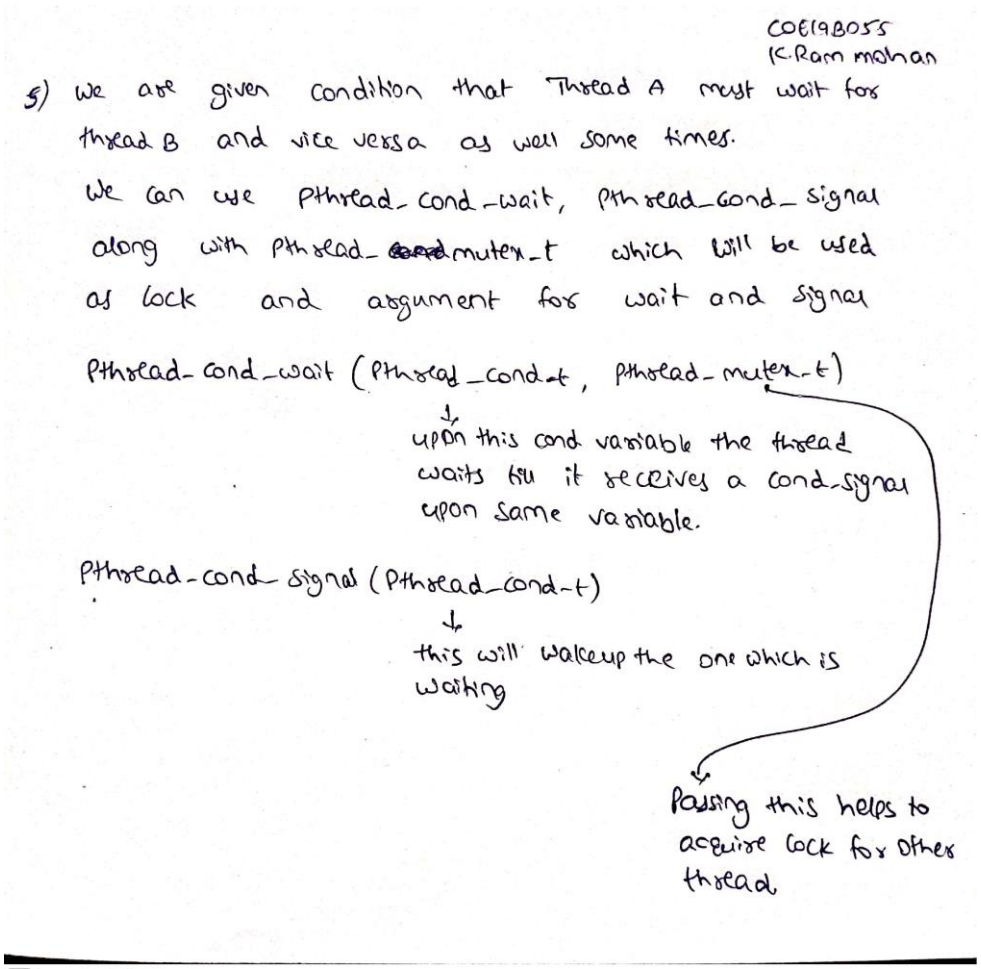
```
ram@ram:~/Documents/OS/Lab9$ gcc -o lab9_q4 CSE19B055_Lab9_Q4.c -lpthread
ram@ram:~/Documents/OS/Lab9$ ./lab9_q4
Producer inserted item 4 at 0
Producer inserted item 4 at 1
Producer inserted item 1 at 2
Producer inserted item 5 at 3
Producer inserted item 6 at 4
Consumer inserted item 4 from position 0
Consumer inserted item 4 from position 1
Consumer inserted item 1 from position 2
Consumer inserted item 5 from position 3
Consumer inserted item 6 from position 4
```

Q5)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>
#include<time.h>

sem_t s;
int x=1;

pthread_cond_t cond1 = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond2 = PTHREAD_COND_INITIALIZER;

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void *fun1(void * arg)
{
        int *t=(int *)arg;
        printf("Statement a1\n");
        //sem_wait(&s);
        pthread_mutex_lock(&lock);
        x++;
        printf("Thread1 - X updated to %d\n", x);

        if(*t%2==0)
        {
                pthread_cond_wait(&cond1, &lock);
```

```
        }

        x++;
        printf("Statemetn a2\n");
        printf("Thread 1 - X value %d\n", x);

        if(*t%2==1)
        {
                pthread_cond_signal(&cond2);
        }

        //sem_post(&s);
        pthread_mutex_unlock(&lock);
}

void *fun2(void* arg)
{
        int *t = (int *)arg;
        //sem_wait(&s);
        pthread_mutex_lock(&lock);
        printf("Statement b1 \n");
        x--;
        printf("Thraed 2 - X values updated to %d\n", x);

        if(*t%2==1)
        {
                pthread_cond_wait(&cond2, &lock);
        }
```

```
	printf("Statement b2\n");
	printf("Thread 2 - X value updated to %d\n", x);

	if(*t==0)
	{
		pthread_cond_signal(&cond1);
	}
	//sem_post(&s);
	pthread_mutex_unlock(&lock);
}

int main()
{
	srand(time(NULL));
	int t = rand()%10 + 1;

	pthread_t p1, p2;
	sem_init(&s, 0, 1);

	//printf("%d %d\n", t, t%2);
	printf("Initially x is %d\n", x);

	pthread_create(&p1, NULL, fun1, (void *)&t);
	pthread_create(&p2, NULL, fun2, (void *)&t);

	pthread_join(p1, NULL);
	pthread_join(p2, NULL);
}
```



Output:

```
ram@ram:~/Documents/OS/Lab9$ gcc lab9_q5.c -lpthread
ram@ram:~/Documents/OS/Lab9$ ./a.out
Initially x is 1
Statement a1
Statement b1
Thraed 2 - X values updated to 0
Thread1 - X updated to 1
Statemetn a2
Thread 1 - X value 2
Statement b2
Thread 2 - X value updated to 2
```

```
ram@ram:~/Documents/OS/Lab9$ gcc lab9_q5.c -lpthread
ram@ram:~/Documents/OS/Lab9$ ./a.out
Initially x is 1
Statement b1
Thraed 2 - X values updated to 0
Statement a1
Thread1 - X updated to 1
Statemetn a2
Thread 1 - X value 2
Statement b2
Thread 2 - X value updated to 2
```