Mirette Boules

ID=5515

Grp2

**Lab3**

**Explanation of the lab:**

3 functions made

1.function mcounter:

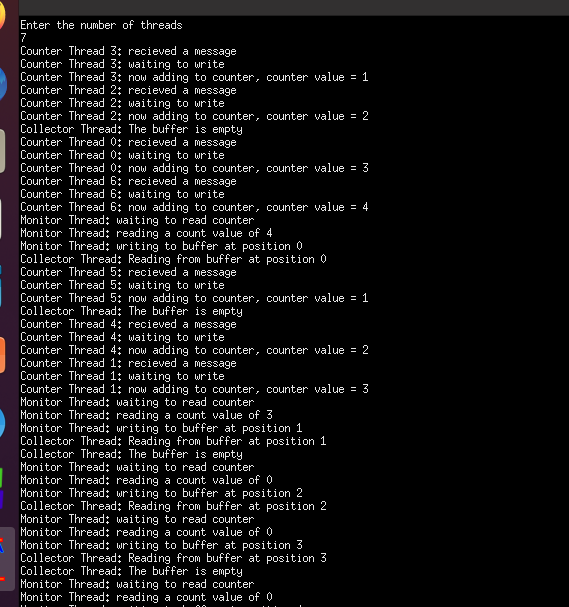
* Count number of thread via count which is a shared int ..so we apply semwait and sempost before and after it to be sure of mutual exclusion

2.function mmonitor:

* Read the value of the count and enqueue it
* Before enqueuing the value of counter we have to semwait on semaphore empty in order to be sure that the buffer isn’t full and semwait on bufferacess and countacess then enqueue it
* After enqueuing the value we sempost on bufferacess and countacess and on semaphore full

3.function mcollector:

* Collect the value of the counter from queue
* Before dequeuing we have to semwait on semaphore full to be sure that the queue isnot empty and semwait on bufferacess
* After dequeuing we sempost on bufferacess and sempost on semaphore empty to let the monitor to enqueue in the buffer

**Output**

Thread 3 increse count to 1

Thread 2 increase count to 2

Collecter doesn’t collect because buffer is empty

Thread 0 increase count to 3

Thread 6 increase count to 4

Monitor enqueue vakue 4

Then make count =0

So thread 5 increase count to 1

And etc…….

The datastructure to make a buffer is a Queue

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

typedef struct

{

int front, rear, num\_of\_elements;

int items[QUEUESIZE];

} Queue;

Queue q;

void initialize(Queue \*q)

{

q->front = 0;

q->rear = -1;

q->num\_of\_elements = 0;

}

int isFull(Queue \*q)

{

if (q->num\_of\_elements == QUEUESIZE)

return 1;

else

return 0;

}

int isEmpty(Queue \*q)

{

if (q->num\_of\_elements == 0)

return 1;

else

return 0;

}

void enqueue(Queue \*q, int value)

{

if (!isFull(q))

{

q->rear=((q->rear+1)%QUEUESIZE);

q->items[q->rear]=value;

q->num\_of\_elements++;

//printf("qqqqqqqqqqqqqqqqqqqqqqq++++++++++++++== %d \n",q->num\_of\_elements);

}

else

printf("Queue is Full \n");

}

int dequeue(Queue \*q)

{

if(isEmpty(q))

{

printf("Queue is Empty \n");

return -1;

}

else

{

q->front=((q->front+1)%QUEUESIZE);

int value = q->items[q->front];

q->num\_of\_elements--;

// printf("qqqqqqqqqqqqqqqqqqqqqqq------------------------== %d \n",q->num\_of\_elements);

return value;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**//function gets the count of threads at a time**

**//count is a shared resource must be putten in a semaphore to guarante mutual exclusion )**

void \*mCounter(void \*x){

int randvar = rand()%5;

sleep(randvar);

printf("Counter Thread %d: recieved a message\n",x);

sem\_wait(&counteracces);

printf("Counter Thread %d: waiting to write\n",x);

count++;

printf("Counter Thread %d: now adding to counter, counter value = %d\n",x,count);

sem\_post(&counteracces);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**//function to read the count and access it in the buffer**

**//(buffer and count are a shared resources )**

void \*mMonitor(void \*y){

while(1){

int randvar = rand()%3;

sleep(randvar);

if(q.num\_of\_elements==BUFFERSIZE)

{printf("Buffer is full!\n");}

printf("Monitor Thread: waiting to read counter\n");

sem\_wait(&empty);

sem\_wait(&bufferaccess);

sem\_wait(&counteracces);

enqueue(&q,count);

printf("Monitor Thread: reading a count value of %d\n",count);

printf("Monitor Thread: writing to buffer at position %d\n",q.rear);

count=0;

sem\_post(&counteracces);

sem\_post(&bufferaccess);

sem\_post(&full);

}

pthread\_exit(NULL);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void \*mCollector(void \*z){

while(1)

{

int randvar = rand()%8;

sleep(randvar);

if(q.num\_of\_elements==0)

{printf("Collector Thread: The buffer is empty \n");}

sem\_wait(&full);

sem\_wait(&bufferaccess);

printf("Collector Thread: Reading from buffer at position %d\n",q.front);

int temp=dequeue(&q);

sem\_post(&bufferaccess);

sem\_post(&empty);

}

pthread\_exit(NULL);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**//main**

int main()

{

sem\_init(&empty,0,BUFFERSIZE);

sem\_init(&full,0,0);

sem\_init(&counteracces,0,1);

sem\_init(&bufferaccess,0,1);

initialize(&q);

printf("Enter the number of threads \n");

scanf("%d",&n);

pthread\_t thread1,thread2,thread3[n];

pthread\_create(&thread1, NULL, &mMonitor, NULL);

pthread\_create(&thread2, NULL, &mCollector, NULL);

for(int i = 0 ; i < n ; i++){

pthread\_create(&thread3[i], NULL, &mCounter, (void \*)i);

}

pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

for(int i = 0 ; i < n ; i++){

pthread\_join(thread3[i], NULL);

}

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*