East West University

CSE325 (Section: 2)

PROJECT REPORT

Amherst Candy Factory Problem

Prepared for

Tanni Mittra

Senior Lecturer,

Department of CSE

Prepared by

Group-3

Name	ID
K. M. Safin Kamal	2020-1-60-235
Rokeya Jahan Chowdhury Ettifa	2020-1-60-232
Md. Arash Kursy Sourov	2020-1-60-177

Date of Submission: 21th January 2022

Project Title: Amherst Candy Factory Problem

Project Description:

Amherst candy factory is preparing up for Halloween and has implemented an assembly line to ramp up Halloween candy production. The assembly line will be implemented using a bounded buffer producer and consumers. The factory contains two types of worker threads: producers and consumers. Each producer thread produces a certain type of candy, while each consumer creates boxes of assorted candies using ones produced by producers. After producing each candy, the producer deposits into the bounded buffer. If the buffer is full, it must wait until one slot becomes available. A consumer extracts candy from the bounded buffer, one at a time, and when 1 candy items have been extracted, it fills up a box of assorted candy. Implement a synchronization method to solve the problem.

System Requirements:

Here we used Intel i5 7500 processor with 16GB DDR4 RAM. Our operating system is Ubuntu which is required to run the project and we used Codeblocks IDE for the code. To run the code, we need to use terminal of Ubuntu. People can also run this project in low end devices.

System design:

In this project, we have created two type of thread, one is producer and another one is consumer.

(A thread is a flow of execution accomplishing a given task within a process. Thread is the unit of cpu utilization. All modern cpu are multi-threaded. Creating threads are lightweight compared to creating processes. We will use POSIX standard Pthread library to create and manipulate threads.)

We also created a bufferlist array to put candy that is created by producers. This work is done in Produce function. The consumers consumes candy from bufferlist and put it in Boxlist array to create boxes. We have also used thread Synchronization semaphore and Mutex lock.

Code Implementation:

Let's start from the main function.

```
int main(){

printf("Enter Bounded Buffer Size :");
    scanf("%d",&buffer);
 full=buffer;

for(int k=0; k<buffer; k++)
    {
        bufferlist[k]=-1;//to make every element of buffer -1
    }


printf("Enter the number of candy type:");
    scanf("%d",&candytype);


for(cx=1;cx<=candytype;cx++)
    {
        printf("Candy %d: ",cx);
        scanf("%s",candylist[cx]);//enter candy names
    }

 for(int t=0; t<100; t++)
    {
        for(int q=0; q<100; q++)
        {
            boxlist[t][q]=-1;//make -1 to all boxlist elements
        }
    }


printf("Enter capacity in a box: ");
    scanf("%d",&box);//highest number of candy can be put in a box
```

→*In main function we take input of necessary variables like buffer size, candy type and name, box capacity etc. we also initialize all the elements of every array to "-1" for user purpose.

```
pthread_t producer[99],consumer[99];
pthread_mutex_init(&mutex, NULL);
sem_init(&sema,0,1);
```

→*Here we initialized the producer and consumer thread. We also initialized the mutex and semaphore.

```
while(1)
    {
        printf("\n1. Produce candy\n2. Consume candy\n3. Show candy in box\n4. Show empty buffer space\n5. Show candy list\n6. exit");
        printf("\nEnter your choice:");
        scanf("%d",&n);

        if(n==1){//to produce candy

          int h,pPro;
            printf("How many candy you want to produce: ");
            scanf("%d",&pPro);
            for(h=0; h<pPro; h++)
            {
            if(buffer>0){//if buffer has any space to put candy

               printallcandy();
               printf("Enter the candy you want to produce:");
               scanf("%d",&item);//select the candy that want to produce

               pthread_create(&producer[p],NULL, (void*)produce,NULL);//producer thread that send to produce funtion

               int prono=p+1;
               printf("Producer %d produced candy ", prono);
               printcandy(item);//print the candy that want to produce
               printf(" in buffer %d\n",in);

               pthread_join(producer[p], NULL);//start to run the thread

            }
            else
                {
                    printf("###Buffer is full.###\n");
                }

        }
        }
```

→*Here we take input of how much we want to produce candy and then we create that number of producer thread. Here we also check if buffer has any empty space or not. This thread produce candy and put in buffer. This work is done in produce function.

```
else if(n==2)
{
    int v,pCon;
    printf("How many candy you want to consume: ");
    scanf("%d",&pCon);
    for(v=0; v<pCon; v++)
    {
        if(buffer<full)//if buffer is not totally empty
        {

            pthread_create(&consumer[c], NULL, (void *)consume, NULL);//consumer thread that send to consume function

            printf("Consumer %d consumed candy ", c+1);
            printcandy(bufferlist[out]);//print the candy that is consumed by the consumer
            printf(" from buffer %d\n",bn);
            bn = (bn+1)%full;//to count the buffer number

            pthread_join(consumer[c], NULL);//start the thread
        }


        else
        {
            printf("###Buffer is empty.###\n");
        }
    }

}
```

→*Here we take input of how much we want to consume candy and then we create that number of consumer thread. Here we also check if buffer is totally empty or not. If buffer is fully empy, it cannot consume any candy. This thread consume candy and put it in box from buffer. This work is done in produce function.

```
else if(n==3)
{
    if(boxf!=0)//if box created
    {
        printbox();
    }
    else
    {
        printf("###No box has been created yet###\n");
    }

}

else if(n==4)
{
    printf("Empty buffer space : %d\n",buffer);
}

else if(n==5)
{
    printallcandy();
}


else if(n==6)
{
    exit(0);
}
else
printf("###Wrong Choice.###\n");
```

```
    pthread_mutex_destroy(&mutex);
    return 0;

}
```

→*In the last part of the main function there is some other necessary option so that we can monitor the project. Lastly we destroy the mutex lock.

```
void *produce(void *pro)
{
   pthread_mutex_lock(&mutex);
   sem_wait(&sema);

   bufferlist[in]=item;//put the selected candy in buffer
   buffer--;// one buffer space is used
   in=(in+1)%full;//to count the buffer no
   p++;//to count the producer thread number

   sem_post(&sema);
   pthread_mutex_unlock(&mutex);
   pthread_exit(NULL);

}
```

→*This produce function is created for producer thread. Here producer thread produces a candy and put it in buffer.

```
void *consume(void *con)
{
  pthread_mutex_lock(&mutex);

  buffer++;//one buffer space is free
  boxlist[i][j]=bufferlist[out];//put candy from buffer to box
    j++;
    if(j==box)//to update the index of boxlist array
    {
        i++;
        j=0;
    }

    bufferlist[out]=-1;//put -1 to free buffer space
    out = (out+1)%full;//to count the buffer no
    c++;//to count the consumer thread number
    boxf++;//to count box number

    pthread_mutex_unlock(&mutex);
    pthread_exit(NULL);
}
```

→*This Consume function is created for consumer thread. Here each consumer thread consumes a candy from buffer and put it in box.

```
void printallcandy()//to print all candy
{
 for(cx=1;cx<=candytype;cx++)
    {
   printf("%d %s \n",cx,candylist[cx]);
    }

}
void printcandy(int x)//to print a selected candy
{
   printf(" %s ",candylist[x]);

}
void printbox()//to print the boxes
{
  for(int t=0; t<i; t++)
    {
    printf("Box %d:",t+1);
    for(int q=0; q<box; q++)
        {
            if(boxlist[t][q]==-1)
              {printf("EMPTY");}
            else{
              printcandy(boxlist[t][q]);
            }
            printf(" , ");
        }
        printf("\n");
    }
}
```

→*This are some necessary functions to print candies and box elements. In printbox function we print the box elements depending on box capacity (“box” variable in here).

Testing result

```
safin@kamal:~/Desktop/candy$ gcc candy1.c -pthread -o candy1
safin@kamal:~/Desktop/candy$ ./candy1
Enter Bounded Buffer Size :4
Enter the number of candy type:3
Candy 1: kitkat
Candy 2: dairymilk
Candy 3: hersheys
Enter capacity in a box: 2

1. Produce candy
2. Consume candy
3. Show candy in box
4. Show empty buffer space
5. Show candy list
6. exit
Enter your choice:1
How many candy you want to produce: 5
1 kitkat
2 dairymilk
3 hersheys
Enter the candy you want to produce:1
Producer 1 produced candy  kitkat  in buffer 0
1 kitkat
2 dairymilk
3 hersheys
Enter the candy you want to produce:2
Producer 2 produced candy  dairymilk  in buffer 1
1 kitkat
2 dairymilk
3 hersheys
Enter the candy you want to produce:3
Producer 3 produced candy  hersheys  in buffer 2
1 kitkat
2 dairymilk
3 hersheys
Enter the candy you want to produce:2
Producer 4 produced candy  dairymilk  in buffer 3
###Buffer is full.###
```

Here we wanted to produce 5 candy, but buffer size is 4. That's why the 5th producer canot produce candy because of the full buffer (last line showing "Buffer is full").

```
1. Produce candy
2. Consume candy
3. Show candy in box
4. Show empty buffer space
5. Show candy list
6. exit
Enter your choice:2
How many candy you want to consume: 5
Consumer 1 consumed candy  kitkat  from buffer 0
Consumer 2 consumed candy  dairymilk  from buffer 1
Consumer 3 consumed candy  hersheys  from buffer 2
Consumer 4 consumed candy  dairymilk  from buffer 3
###Buffer is empty.###

1. Produce candy
2. Consume candy
3. Show candy in box
4. Show empty buffer space
5. Show candy list
6. exit
Enter your choice:3
Box 1: kitkat  ,  dairymilk  ,
Box 2: hersheys  ,  dairymilk  ,

1. Produce candy
2. Consume candy
3. Show candy in box
4. Show empty buffer space
5. Show candy list
6. exit
Enter your choice:4
Empty buffer space : 4

1. Produce candy
2. Consume candy
3. Show candy in box
4. Show empty buffer space
5. Show candy list
6. exit
Enter your choice:5
1 kitkat
2 dairymilk
3 hersheys
```