Liya Norng

Project 3

3/5/16

I got that the number of ratio for completion for the first run is 1/1. The numerator is the producer over the denominator, which is consumer. Since there is one producer, and one consumer so this mean there shouldn’t be more than one item in the buffer, since the number of producer is equal to the consumer. So the number of the throughput for the first run in 25 sec is about the average of 10 completions since it’s also depending on the sleep time for each producer, and consumer.

Then for the second ran, I got that the ratio is 5/2. This mean that there should be more items in the buffer as time clicking, since there are more producer than the consumer. For the number of throughput in 25 sec is about the average of 24 completions. It should be about twice of the first run, because there is one extra consumer in the second run.

When we compare the two runs, I got that it is not 5 times as large as the first ran, because from the output ran I got that for the first ran there is about 10 completions, and for the second ran, I got 25 completions. So For it to be 5 times as large of the first ran. The second ran has to output at least around the average of 55, since the first ran complete about 10. Its doesn’t mean that if we have more producer we would be having bigger throughput, because it is also depend on the number of the consumer. So in this case there is only 1 consumer over the first ran. So that’s why the second ran is about twice as large of the first ran, not five times as large of the first ran. We can also used the ratio between first ran with the second ran on the consumer. So it would be true that it is 1 to 2 as a ratio. Which you can see that the second ran is twice as large of the first ran.

Script started on Thu 03 Mar 2016 02:35:07 PM PST

[uh8425@algebra ~]$ gcc v.c -lpthread -o s

[uh8425@algebra ~]$ ./s 25 1 1

Main thread beginning.

Creating producer thread with id 0.

Creating consumer thread with id 0.

Main thread sleeping for 25 seconds.

Producer thread 0 sleeping for 2 seconds.

Consumer thread 0 sleeping for 2 seconds.

Insert\_item inserted item 78 at position 0

[78][empty][empty][empty][empty]in = 1, out = 4.

Producer thread 0 inserted value 78.

Producer thread 0 sleeping for 2 seconds.

Remove\_item removed item 0 at position 0

[empty][empty][empty][empty][empty]in = 0, out = 5.

Consumer thread 0 removed value 78.

Consumer thread 0 sleeping for 3 seconds.

Insert\_item inserted item 36 at position 0

[36][empty][empty][empty][empty]in = 1, out = 4.

Producer thread 0 inserted value 36.

Producer thread 0 sleeping for 2 seconds.

Remove\_item removed item 78 at position 0

[empty][empty][empty][empty][empty]in = 0, out = 5.

Consumer thread 0 removed value 36.

Consumer thread 0 sleeping for 1 seconds.

Insert\_item inserted item 50 at position 0

[50][empty][empty][empty][empty]in = 1, out = 4.

Producer thread 0 inserted value 50.

Producer thread 0 sleeping for 2 seconds.

Remove\_item removed item 36 at position 0

[empty][empty][empty][empty][empty]in = 0, out = 5.

Consumer thread 0 removed value 50.

Consumer thread 0 sleeping for 3 seconds.

Insert\_item inserted item 28 at position 0

[28][empty][empty][empty][empty]in = 1, out = 4.

Producer thread 0 inserted value 28.

Producer thread 0 sleeping for 3 seconds.

Remove\_item removed item 50 at position 0

[empty][empty][empty][empty][empty]in = 0, out = 5.

Consumer thread 0 removed value 28.

Consumer thread 0 sleeping for 2 seconds.

Insert\_item inserted item 64 at position 0

[64][empty][empty][empty][empty]in = 1, out = 4.

Producer thread 0 inserted value 64.

Producer thread 0 sleeping for 2 seconds.

Remove\_item removed item 28 at position 0

[empty][empty][empty][empty][empty]in = 0, out = 5.

Consumer thread 0 removed value 64.

Consumer thread 0 sleeping for 1 seconds.

Insert\_item inserted item 27 at position 0

[27][empty][empty][empty][empty]in = 1, out = 4.

Producer thread 0 inserted value 27.

Producer thread 0 sleeping for 2 seconds.

Remove\_item removed item 64 at position 0

[empty][empty][empty][empty][empty]in = 0, out = 5.

Consumer thread 0 removed value 27.

Consumer thread 0 sleeping for 2 seconds.

Insert\_item inserted item 12 at position 0

[12][empty][empty][empty][empty]in = 1, out = 4.

Producer thread 0 inserted value 12.

Producer thread 0 sleeping for 3 seconds.

Remove\_item removed item 27 at position 0

[empty][empty][empty][empty][empty]in = 0, out = 5.

Consumer thread 0 removed value 12.

Consumer thread 0 sleeping for 1 seconds.

Insert\_item inserted item 30 at position 0

[30][empty][empty][empty][empty]in = 1, out = 4.

Producer thread 0 inserted value 30.

Producer thread 0 sleeping for 3 seconds.

Remove\_item removed item 12 at position 0

[empty][empty][empty][empty][empty]in = 0, out = 5.

Consumer thread 0 removed value 30.

Consumer thread 0 sleeping for 3 seconds.

Insert\_item inserted item 63 at position 0

[63][empty][empty][empty][empty]in = 1, out = 4.

Producer thread 0 inserted value 63.

Producer thread 0 sleeping for 2 seconds.

Remove\_item removed item 30 at position 0

[empty][empty][empty][empty][empty]in = 0, out = 5.

Consumer thread 0 removed value 63.

Consumer thread 0 sleeping for 2 seconds.

Insert\_item inserted item 36 at position 0

[36][empty][empty][empty][empty]in = 1, out = 4.

Producer thread 0 inserted value 36.

Producer thread 0 sleeping for 3 seconds.

Remove\_item removed item 63 at position 0

[empty][empty][empty][empty][empty]in = 0, out = 5.

Consumer thread 0 removed value 36.

Consumer thread 0 sleeping for 1 seconds.

Main thread exiting.

[uh8425@algebra ~]$ ./s 25 5 2

Main thread beginning.

Creating producer thread with id 0.

Creating producer thread with id 1.

Creating producer thread with id 2.

Creating producer thread with id 3.

Creating producer thread with id 4.

Creating consumer thread with id 0.

Creating consumer thread with id 1.

Main thread sleeping for 25 seconds.

Producer thread 0 sleeping for 2 seconds.

Producer thread 1 sleeping for 2 seconds.

Producer thread 2 sleeping for 1 seconds.

Producer thread 3 sleeping for 2 seconds.

Producer thread 4 sleeping for 3 seconds.

Consumer thread 0 sleeping for 2 seconds.

Consumer thread 1 sleeping for 2 seconds.

Insert\_item inserted item 93 at position 0

[93][empty][empty][empty][empty]in = 1, out = 4.

Producer thread 2 inserted value 93.

Producer thread 2 sleeping for 1 seconds.

Insert\_item inserted item 22 at position 1

[93][22][empty][empty][empty]in = 2, out = 3.

Producer thread 0 inserted value 22.

Producer thread 0 sleeping for 3 seconds.

Insert\_item inserted item 28 at position 2

[93][22][28][empty][empty]in = 3, out = 2.

Producer thread 1 inserted value 28.

Producer thread 1 sleeping for 3 seconds.

Insert\_item inserted item 60 at position 3

[93][22][28][60][empty]in = 4, out = 1.

Producer thread 3 inserted value 60.

Producer thread 3 sleeping for 3 seconds.

Remove\_item removed item 0 at position 0

[empty][22][28][60][empty]in = 3, out = 2.

Consumer thread 0 removed value 93.

Consumer thread 0 sleeping for 2 seconds.

Insert\_item inserted item 41 at position 4

[empty][22][28][60][41]in = 4, out = 1.

Producer thread 2 inserted value 41.

Producer thread 2 sleeping for 1 seconds.

Remove\_item removed item 0 at position 1

[empty][empty][28][60][41]in = 3, out = 2.

Consumer thread 1 removed value 22.

Consumer thread 1 sleeping for 2 seconds.

Insert\_item inserted item 37 at position 0

[37][empty][28][60][41]in = 4, out = 1.

Producer thread 4 inserted value 37.

Producer thread 4 sleeping for 3 seconds.

Insert\_item inserted item 69 at position 1

[37][69][28][60][41]in = 5, out = 0.

Producer thread 2 inserted value 69.

Producer thread 2 sleeping for 1 seconds.

Remove\_item removed item 93 at position 2

[37][69][empty][60][41]in = 4, out = 1.

Consumer thread 0 removed value 28.

Consumer thread 0 sleeping for 1 seconds.

Insert\_item inserted item 83 at position 2

[37][69][83][60][41]in = 5, out = 0.

Producer thread 2 inserted value 83.

Producer thread 2 sleeping for 3 seconds.

Remove\_item removed item 22 at position 3

[37][69][83][empty][41]in = 4, out = 1.

Consumer thread 1 removed value 60.

Consumer thread 1 sleeping for 3 seconds.

Insert\_item inserted item 24 at position 3

[37][69][83][24][41]in = 5, out = 0.

Producer thread 0 inserted value 24.

Producer thread 0 sleeping for 2 seconds.

Remove\_item removed item 28 at position 4

[37][69][83][24][empty]in = 4, out = 1.

Insert\_item inserted item 36 at position 4

[37][69][83][24][36]in = 5, out = 0.

Producer thread 1 inserted value 36.

Producer thread 1 sleeping for 1 seconds.

Consumer thread 0 removed value 41.

Consumer thread 0 sleeping for 1 seconds.

Remove\_item removed item 41 at position 0

[empty][69][83][24][36]in = 4, out = 1.

Consumer thread 0 removed value 37.

Consumer thread 0 sleeping for 1 seconds.

Insert\_item inserted item 30 at position 0

[30][69][83][24][36]in = 5, out = 0.

Producer thread 3 inserted value 30.

Producer thread 3 sleeping for 2 seconds.

Remove\_item removed item 37 at position 1

[30][empty][83][24][36]in = 4, out = 1.

Consumer thread 0 removed value 69.

Consumer thread 0 sleeping for 2 seconds.

Insert\_item inserted item 43 at position 1

[30][43][83][24][36]in = 5, out = 0.

Producer thread 2 inserted value 43.

Producer thread 2 sleeping for 1 seconds.

Remove\_item removed item 60 at position 2

[30][43][empty][24][36]in = 4, out = 1.

Insert\_item inserted item 70 at position 2

[30][43][70][24][36]in = 5, out = 0.

Producer thread 1 inserted value 70.

Producer thread 1 sleeping for 1 seconds.

Consumer thread 1 removed value 83.

Consumer thread 1 sleeping for 1 seconds.

Remove\_item removed item 83 at position 3

[30][43][70][empty][36]in = 4, out = 1.

Insert\_item inserted item 57 at position 3

[30][43][70][57][36]in = 5, out = 0.

Producer thread 0 inserted value 57.

Producer thread 0 sleeping for 3 seconds.

Consumer thread 1 removed value 24.

Consumer thread 1 sleeping for 3 seconds.

Remove\_item removed item 69 at position 4

[30][43][70][57][empty]in = 4, out = 1.

Consumer thread 0 removed value 36.

Consumer thread 0 sleeping for 3 seconds.

Insert\_item inserted item 59 at position 4

[30][43][70][57][59]in = 5, out = 0.

Producer thread 4 inserted value 59.

Producer thread 4 sleeping for 1 seconds.

Remove\_item removed item 24 at position 0

[empty][43][70][57][59]in = 4, out = 1.

Consumer thread 1 removed value 30.

Consumer thread 1 sleeping for 2 seconds.

Insert\_item inserted item 20 at position 0

[20][43][70][57][59]in = 5, out = 0.

Producer thread 3 inserted value 20.

Producer thread 3 sleeping for 3 seconds.

Remove\_item removed item 36 at position 1

[20][empty][70][57][59]in = 4, out = 1.

Consumer thread 0 removed value 43.

Consumer thread 0 sleeping for 3 seconds.

Insert\_item inserted item 85 at position 1

[20][85][70][57][59]in = 5, out = 0.

Producer thread 2 inserted value 85.

Producer thread 2 sleeping for 1 seconds.

Remove\_item removed item 30 at position 2

[20][85][empty][57][59]in = 4, out = 1.

Insert\_item inserted item 38 at position 2

[20][85][38][57][59]in = 5, out = 0.

Producer thread 1 inserted value 38.

Producer thread 1 sleeping for 3 seconds.

Consumer thread 1 removed value 70.

Consumer thread 1 sleeping for 2 seconds.

Remove\_item removed item 43 at position 3

[20][85][38][empty][59]in = 4, out = 1.

Consumer thread 0 removed value 57.

Consumer thread 0 sleeping for 1 seconds.

Insert\_item inserted item 14 at position 3

[20][85][38][14][59]in = 5, out = 0.

Producer thread 4 inserted value 14.

Producer thread 4 sleeping for 1 seconds.

Remove\_item removed item 70 at position 4

[20][85][38][14][empty]in = 4, out = 1.

Insert\_item inserted item 27 at position 4

[20][85][38][14][27]in = 5, out = 0.

Producer thread 0 inserted value 27.

Producer thread 0 sleeping for 3 seconds.

Consumer thread 1 removed value 59.

Consumer thread 1 sleeping for 1 seconds.

Remove\_item removed item 57 at position 0

[empty][85][38][14][27]in = 4, out = 1.

Consumer thread 0 removed value 20.

Consumer thread 0 sleeping for 3 seconds.

Insert\_item inserted item 6 at position 0

[6][85][38][14][27]in = 5, out = 0.

Producer thread 4 inserted value 6.

Producer thread 4 sleeping for 2 seconds.

Remove\_item removed item 59 at position 1

[6][empty][38][14][27]in = 4, out = 1.

Consumer thread 1 removed value 85.

Consumer thread 1 sleeping for 2 seconds.

Insert\_item inserted item 82 at position 1

[6][82][38][14][27]in = 5, out = 0.

Producer thread 3 inserted value 82.

Producer thread 3 sleeping for 3 seconds.

Remove\_item removed item 85 at position 2

[6][82][empty][14][27]in = 4, out = 1.

Consumer thread 1 removed value 38.

Consumer thread 1 sleeping for 1 seconds.

Insert\_item inserted item 63 at position 2

[6][82][63][14][27]in = 5, out = 0.

Producer thread 2 inserted value 63.

Producer thread 2 sleeping for 1 seconds.

Remove\_item removed item 20 at position 3

[6][82][63][empty][27]in = 4, out = 1.

Insert\_item inserted item 30 at position 3

[6][82][63][30][27]in = 5, out = 0.

Producer thread 1 inserted value 30.

Producer thread 1 sleeping for 3 seconds.

Consumer thread 0 removed value 14.

Consumer thread 0 sleeping for 1 seconds.

Remove\_item removed item 38 at position 4

[6][82][63][30][empty]in = 4, out = 1.

Consumer thread 1 removed value 27.

Consumer thread 1 sleeping for 3 seconds.

Insert\_item inserted item 25 at position 4

[6][82][63][30][25]in = 5, out = 0.

Producer thread 4 inserted value 25.

Producer thread 4 sleeping for 1 seconds.

Remove\_item removed item 14 at position 0

[empty][82][63][30][25]in = 4, out = 1.

Insert\_item inserted item 96 at position 0

[96][82][63][30][25]in = 5, out = 0.

Producer thread 0 inserted value 96.

Producer thread 0 sleeping for 2 seconds.

Consumer thread 0 removed value 6.

Consumer thread 0 sleeping for 1 seconds.

Remove\_item removed item 6 at position 1

[96][empty][63][30][25]in = 4, out = 1.

Consumer thread 0 removed value 82.

Consumer thread 0 sleeping for 3 seconds.

Insert\_item inserted item 35 at position 1

[96][35][63][30][25]in = 5, out = 0.

Producer thread 3 inserted value 35.

Producer thread 3 sleeping for 3 seconds.

Remove\_item removed item 27 at position 2

[96][35][empty][30][25]in = 4, out = 1.

Consumer thread 1 removed value 63.

Consumer thread 1 sleeping for 3 seconds.

Insert\_item inserted item 51 at position 2

[96][35][51][30][25]in = 5, out = 0.

Producer thread 2 inserted value 51.

Producer thread 2 sleeping for 3 seconds.

Remove\_item removed item 82 at position 3

[96][35][51][empty][25]in = 4, out = 1.

Consumer thread 0 removed value 30.

Consumer thread 0 sleeping for 2 seconds.

Insert\_item inserted item 33 at position 3

[96][35][51][33][25]in = 5, out = 0.

Producer thread 3 inserted value 33.

Producer thread 3 sleeping for 1 seconds.

Main thread exiting.

[uh8425@algebra ~]$ exit

exit

Script done on Thu 03 Mar 2016 02:42:11 PM PST

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

/\* buffer.h \*/

const int max = 5; // this is just my constant variable

int buffer[5]; // this is my buffer, for storing items

int size = 0;

int In = 0; // this is for keeping track where at in the buffer is the next item to put in

int Out = 0; // this is for keeping track where at in the buffer is the next item to take out

int n = 0; // keeping track how many are in the buffer

// mutex lock

sem\_t mutex; // my mutex lock for keeping track who is in the buffer

// counting semaphore

sem\_t empty; // this is my variable for checking if it is empty

sem\_t full; // this is for keeping track if the buffer is full

// this is my consumer thread, where it will call to the remove method to get the item, and display it

void \* Consumer(void \* i)

{

int item = 0; // storing temporary item

while (1)

{

int sleepTime = rand() % 3 + 1; // generate random number for sleep

/\* sleep for a random period of time \*/

printf ("Consumer thread %ld sleeping for %d seconds.\n", (long) i, sleepTime);

sleep(sleepTime); // telling the thread to go to sleep for sleepTime

// this if statement is for check if it actually get to remove the item

if (remove\_Item(& item) < 0)

{

printf ("Consumer error\n");

}

else

{

printf ("Consumer thread %ld removed value %d.\n", (long) i , item);

}

}

}

// this is my producer thread, where it will generate a number, and it will call to the insert method to insert

// the item that it just generate.

void \* Producer(void \* i)

{

int item = 0; // this is my temporary storage for using to put as a parameter when calling insert method

while(1)

{

/\* generate random number \*/

int sleepTime = rand() % 3 + 1; // generate some random number

/\* sleep for random amount of time \*/

printf ("Producer thread %ld sleeping for %d seconds.\n", (long) i, sleepTime);

sleep(sleepTime); // thread is going to sleep

/\* generate a random number - this is the producer's product \*/

item = rand() % 100 + 1; // generate some number to put in the buffer, and store it in the item

if (insert\_Item(item) < 0) // this if statement is for checking if the producer actually get

{ // to insert the item

printf ("Producer error\n");

}

else

{

printf ("Producer thread %ld inserted value %d.\n", (long) i , item);

}

}

}

// this is my insert method, it will get the item in the parameter, and it will use to store it in the buffer if it not lock

int insert\_Item(int items)

{

int i = 0; // loop counter

int check = 0; // for checking if the item got remove from the buffer

int out = 0; // running total for how many available space left are in the buffer

int count = 0; // running total for how many item are in the buffer

do

{

/\* produce an item in next produced \*/

sem\_wait(&empty); // lock if buffer is empty

sem\_wait(&mutex); // lock if there is a process is in the buffer

/\* add next produced to the buffer \*/

if (n == 0) // this is for checking if the buffer is empty. if so, then reset the In, and Out so it won't

{ // just store in different slot when when the buffer is empty.

In = 0;

Out = 0;

}

printf ("Insert\_item inserted item %d at position %d\n",items, In);

buffer[In] = items; // putting the item in the buffer

In = (In + 1) % max; // this is for setting the next empty slot in the buffer

check++;

n++;

for (i = 0; i < 5; i++) // this for loop is for checking the buffer for empty, and getting the item to dislpay

{

if (buffer[i] == (int)NULL)

{

printf ("[empty]");

out++;

}

else if (buffer[i] != (int)NULL)

{

printf ("[%d]", buffer[i]);

count++;

}

}

printf ("in = %d, out = %d.\n", count, out);

i = 0; // reset the value for next thread

count = 0;

out = 0;

// release the lock for the next thead to come in

sem\_post(&mutex);

sem\_post(&full);

// returning the value that i get from above

if ( check == 0)

{

return -1;

}

else

{

check = 0;

return 0;

}

}while (1);

}

// this is my remove method, it would get the item from the buffer, and return it to the consumer, so it can display the item to the user

int remove\_Item(int \* items)

{

int i = 0; // loop counter

int out = 0; // running total for how many available space left are in the buffer

int count = 0; // running total for how many item are in the buffer

int check = 0; // for checking if the item got remove from the buffer

do

{ // locking if it is full, and setting mutex

sem\_wait(&full);

sem\_wait(&mutex);

/\* remove an item from buffer to next consumed \*/

printf ("Remove\_item removed item %d at position %d\n",\*items, Out);

\*items = buffer[Out]; // getting the item from the buffer

buffer[Out] = (int)NULL; // setting the item that took out to NULL, reset for the next item

Out = (Out + 1) % max; // settitng the Out to the next available space which one up

check++; // increment check since it already remove

n--; // this is keeping count how many are in the buffer

// this for loop is for checking for item in the buffer, and display them

for (i = 0; i < 5; i++)

{

// this if statement is checking if it is empty

if (buffer[i] == (int)NULL)

{

printf ("[empty]");

out++;

} // so if the buffer is not empty then it will run this if statement

else if (buffer[i] != (int)NULL)

{

printf ("[%d]", buffer[i]);

count++;

}

} // this will print out the result getting from the for loop. Which i use out for empty, and count for

// item in the buffer

printf ("in = %d, out = %d.\n", count, out);

i = 0; // resetting the counter value to 0

count = 0; // resetting the running total of item in the buffer value to 0

out = 0; // resetting the running total for empty to 0

sem\_post(&mutex); // release the lock for mutex

sem\_post(&empty); // release the lock for empty

// this will return the result from removing

if ( check == 0)

{

return -1;

}

else

{

check = 0;

return 0;

}

}while (1);

}

int main(int argc, char \*argv[])

{

// this is for storing value return from creating pthread

int producers, consumers;

// this is use to store the amount of producer, and consumer

int run = 0, numProducer = 0, numConsumer = 0;

// doing if statment to check to make sure the user give the right format that it suppose to be

if (argc != 4)

{

printf ("Sorry that's not the right format.\n");

printf ("Please use this format ./name (int)seconds (int)producer (int)consumer.\n");

exit (1);

}

else

{

// if the top if statement is false, then the else will run, and turing the value of string into int.

run = atoi(argv[1]); // how long the main will sleep

numProducer = atoi(argv[2]); // how many produer to make

numConsumer = atoi(argv[3]); // how many consumer to make

}

/\* 2. Initialize buffer \*/

// creating the link to the pthread

pthread\_t producerT[numProducer];

pthread\_t consumerT[numConsumer];

printf ("Main thread beginning.\n");

// initialize the semaphore for full, empty, and mutex

// counting semaphore for full, and empty

int em = sem\_init(&empty, 0, max); // MAX buffers are empty to begin with...

int ful = sem\_init(&full, 0, 0); // ... and 0 are full

// mutex locks

int mut = sem\_init(&mutex, 0, 1); // mutex = 1 because it is a lock (NEW LINE)

// setting the buffer to null so easier to do if statement to check for empty

int i = 0; // loop counter

for (i = 0; i < max; i++)

{

buffer[i] = (int)NULL;

}

/// this is for checking if the initialization of the mutex, full, and empty is fail

if (em == -1 || ful == -1 || mut == -1)

{

printf("Fail to initialize the semaphore.\n");

exit(1); // this would exit if it fail

}

/\* 3. Create producer threads. \*/

// creating the producer threads by doing for loop

for (i = 0; i < numProducer; i++)

{

printf ("Creating producer thread with id %d.\n", i);

fflush(stdout); // want the OS to flush the statement above

producers = pthread\_create(&producerT[i], NULL, Producer, (void\*)(long)i);

// this is for checking if the pthread is fail

if (producers)

{

printf ("Error:unable to create thread, %d.\n", producers);

exit(1); // this would exit if it fail

}

}

/\* 4. Create consumer threads. \*/

// creating the consumer thread by doing for loop

for (i = 0; i < numConsumer; i++)

{

printf ("Creating consumer thread with id %d.\n", i);

fflush(stdout); // want the OS to flush the statement above

consumers = pthread\_create(&consumerT[i], NULL , Consumer, (void\*)(long)i);

// this is for checking if the pthread is fail

if (consumers)

{

printf ("Error:unable to create thread, %d.\n", consumers);

exit(1); // this would exit if it fail

}

}

/\* 5. Sleep. \*/

printf ("Main thread sleeping for %d seconds.\n", run);

// main sleep for run time

sleep(run);

/\* 6. Kill threads and exit. \*/

// killing all threads, and exit

for (i = 0; i < numProducer; i++)

{ // killing the producer thread

pthread\_cancel(producerT[i]);

}

for (i = 0; i < numConsumer; i++)

{ // killing the consumer thread

pthread\_cancel(consumerT[i]);

}

printf ("Main thread exiting.\n");

exit (1);

}