**OPERATING SYSTEM**



QUESTION 1:

**Multithreaded Sorting Application**

Write a multithreaded sorting program that works as follows: A list of integers is divided into two smaller lists of equal size. Two separate threads (which we will term ***sorting threads***) sort each sublist using a sorting algorithm of your choice. The two sublists are then merged by a third thread—a ***merging thread*** —which merges the two sublists into a single sorted list. Because global data are shared cross all threads, perhaps the easiest way to set up the data is to create a global array. Each sorting thread will work on one half of this array. A second global array of the same size as the unsorted integer array will also be established. The merging thread will then merge the two sublists into this second array. This programming project will require passing parameters to each of the sorting threads. In particular, it will be necessary to identify the starting index from which each thread is to begin sorting. The parent thread will output the sorted array once all sorting threads have exited.

CODE:

#include<stdio.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <pthread.h>

struct Params

{

int \*st;

size\_t l;

int d;

};

pthread\_mutex\_t mtx = PTHREAD\_MUTEX\_INITIALIZER;

void \*merge\_sort\_thread(void \*pv);

void merging(int \*st, int \*m, int \*end)

{

int \*ro = malloc((end - st)\*sizeof(\*ro));

int \*lahu = st, \*rhsa = m, \*duh = ro;

while (lahu != m && rhsa != end)

\*duh++ = (\*lahu <= \*rhsa) ? \*lahu++ : \*rhsa++;

while (lahu != m)

\*duh++ = \*lahu++;

while (rhsa != end)

\*duh++ = \*rhsa++;

memcpy(st, ro, (end - st)\*sizeof(\*ro));

free(ro);

}

void merge\_sort\_ns(int \*st, size\_t l, int d)

{

if (l < 2)

return;

if (d <= 0 || l < 4)

{

merge\_sort\_ns(st, l/2, 0);

merge\_sort\_ns(st+l/2, l-l/2, 0);

}

else

{

struct Params params = { st, l/2, d/2 };

pthread\_t thrd;

pthread\_mutex\_lock(&mtx);

printf("Starting subthread...\n");

pthread\_mutex\_unlock(&mtx);

pthread\_create(&thrd, NULL, merge\_sort\_thread, &params);

merge\_sort\_ns(st+l/2, l-l/2, d/2);

pthread\_join(thrd, NULL);

pthread\_mutex\_lock(&mtx);

printf("Finished subthread.\n");

pthread\_mutex\_unlock(&mtx);

}

merging(st, st+l/2, st+l);

}

void \*merge\_sort\_thread(void \*pv)

{

struct Params \*params = pv;

merge\_sort\_ns(params->st, params->l, params->d);

return pv;

}

void merge\_sort(int \*st, size\_t l)

{

merge\_sort\_ns(st, l, 4);

}

int main()

{

static const unsigned int N = 2048;

int \*e = malloc(N \* sizeof(\*e));

unsigned int i;

srand((unsigned)time(0));

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

{

e[i] = rand() % 1024;

printf("%4d ", e[i]);

if ((i+1)%8 == 0)

printf("\n");

}

printf("\n");

merge\_sort(e, N);

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

{

printf("%4d ", e[i]);

if ((i+1)%8 == 0)

printf("\n");

}

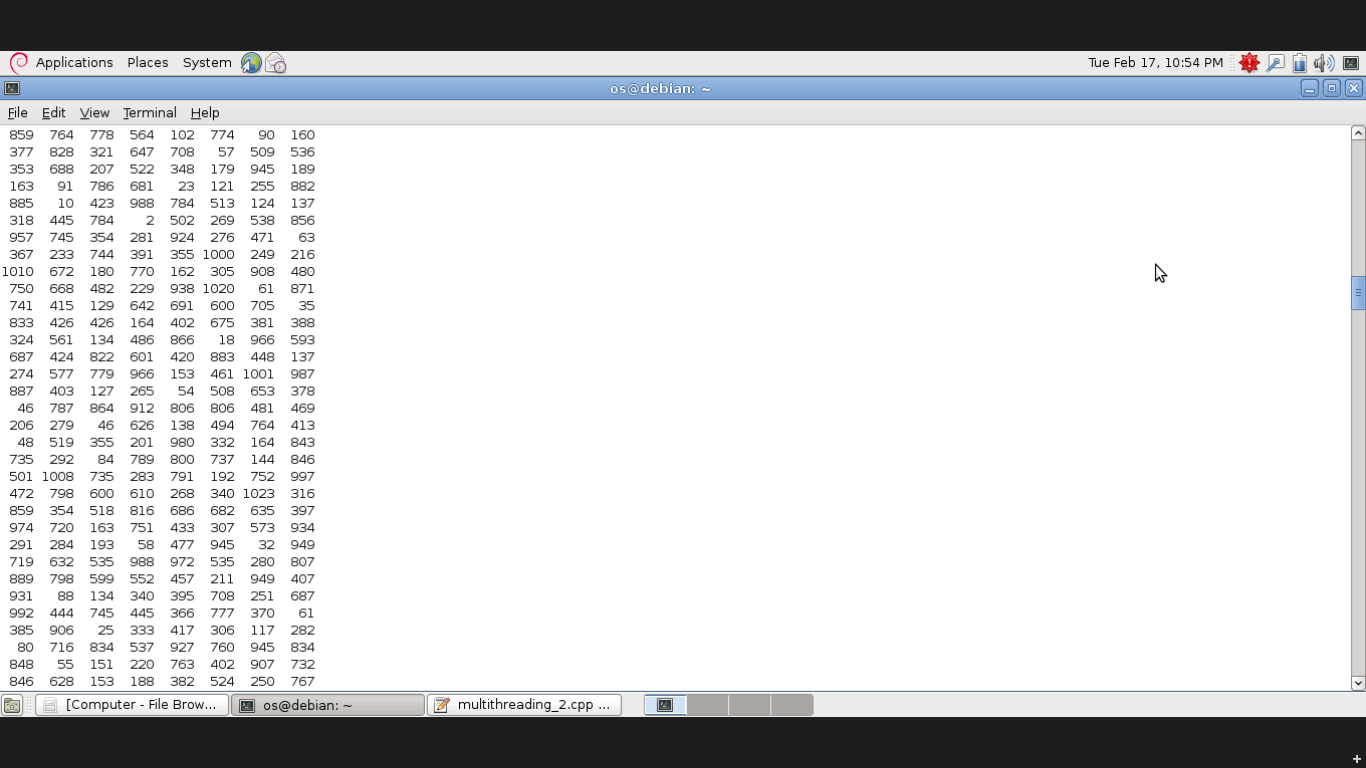
printf("\n");

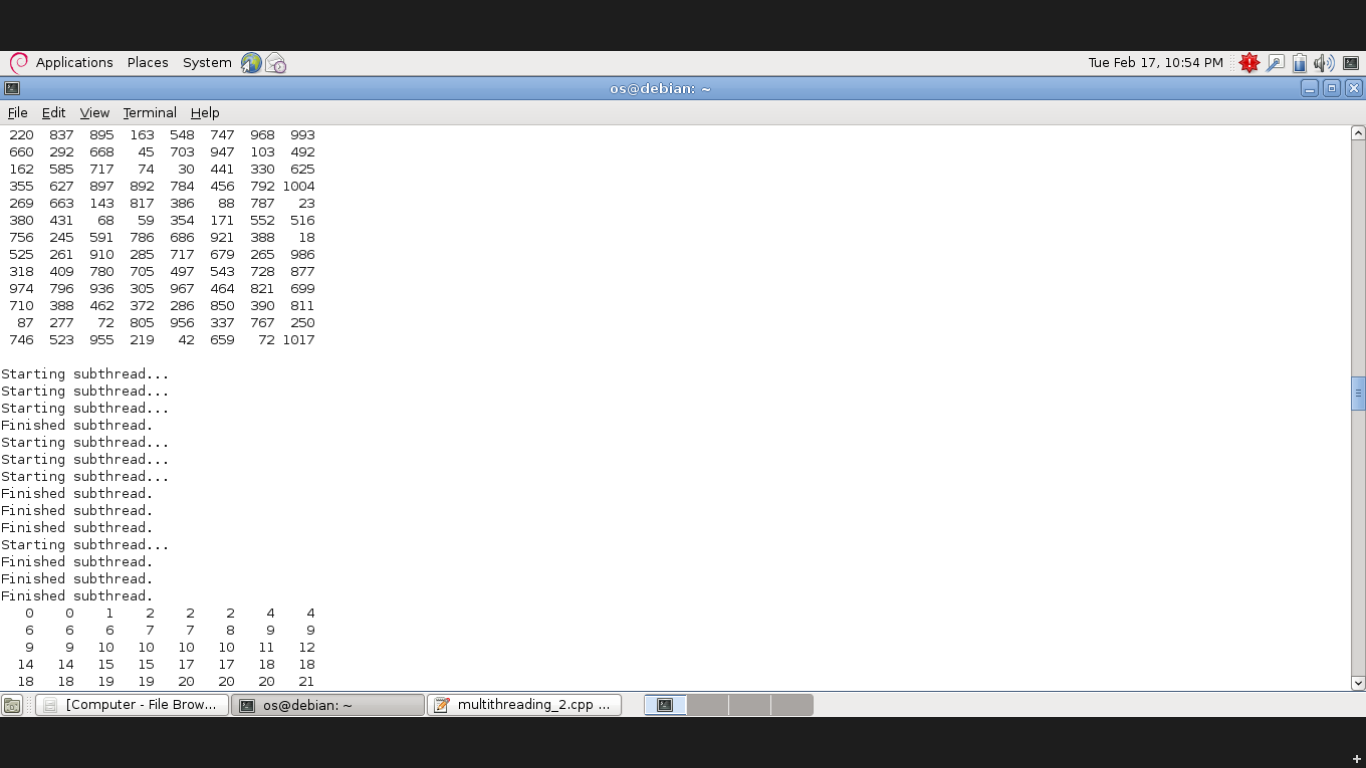
free(e);

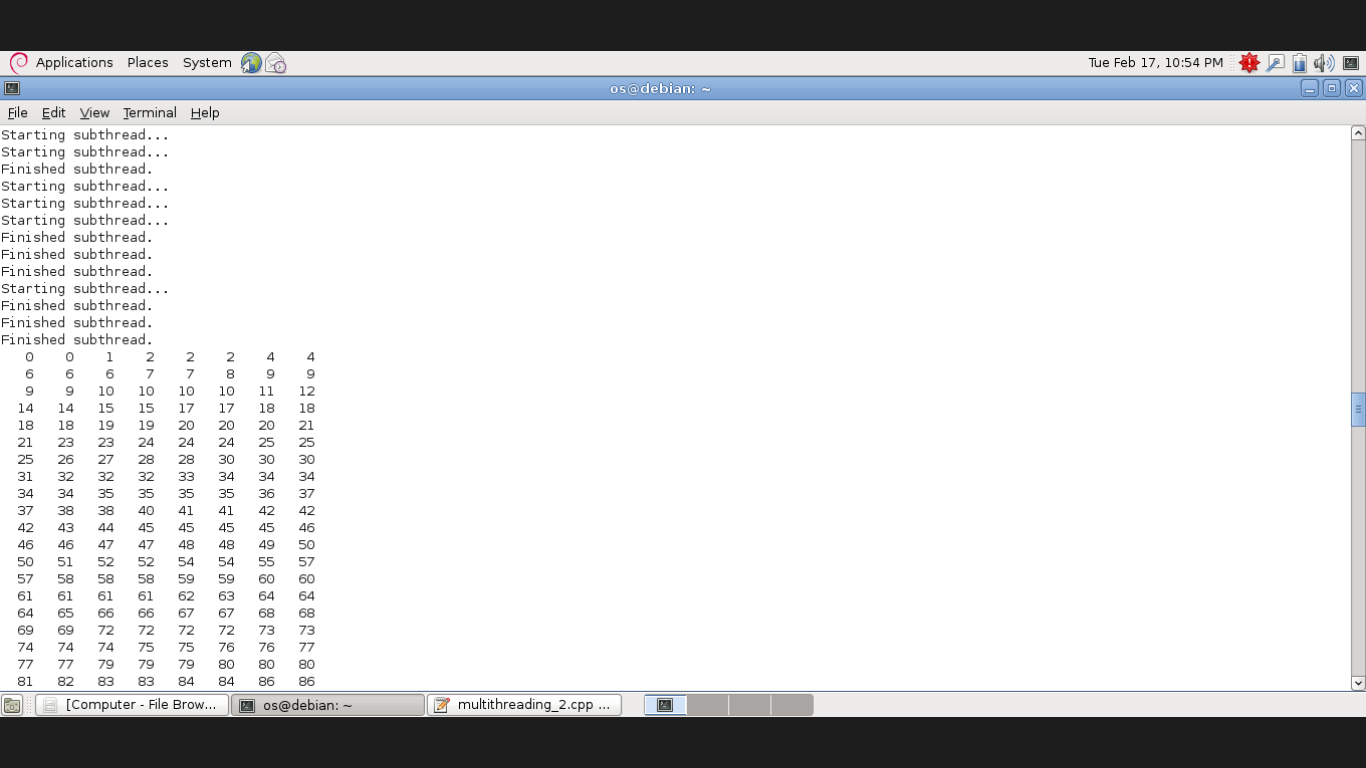
return 0;

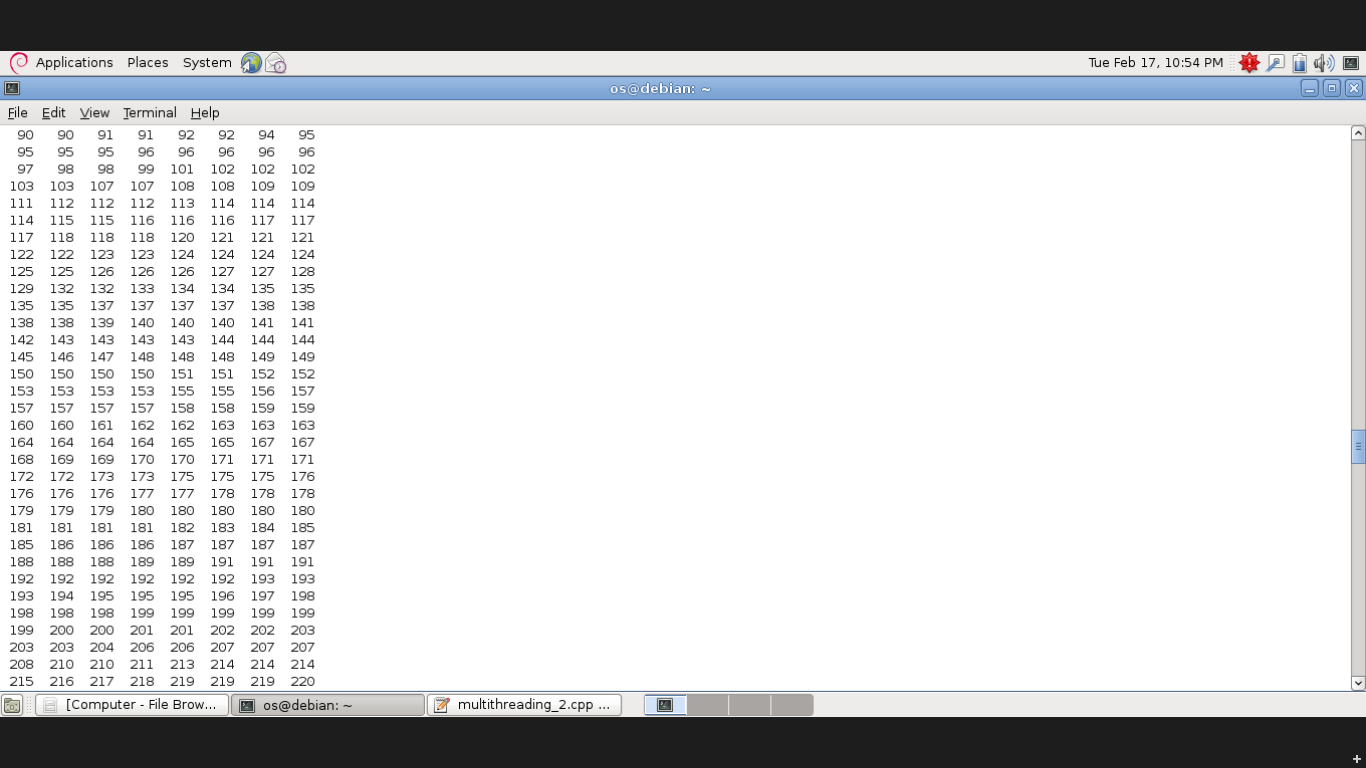
}

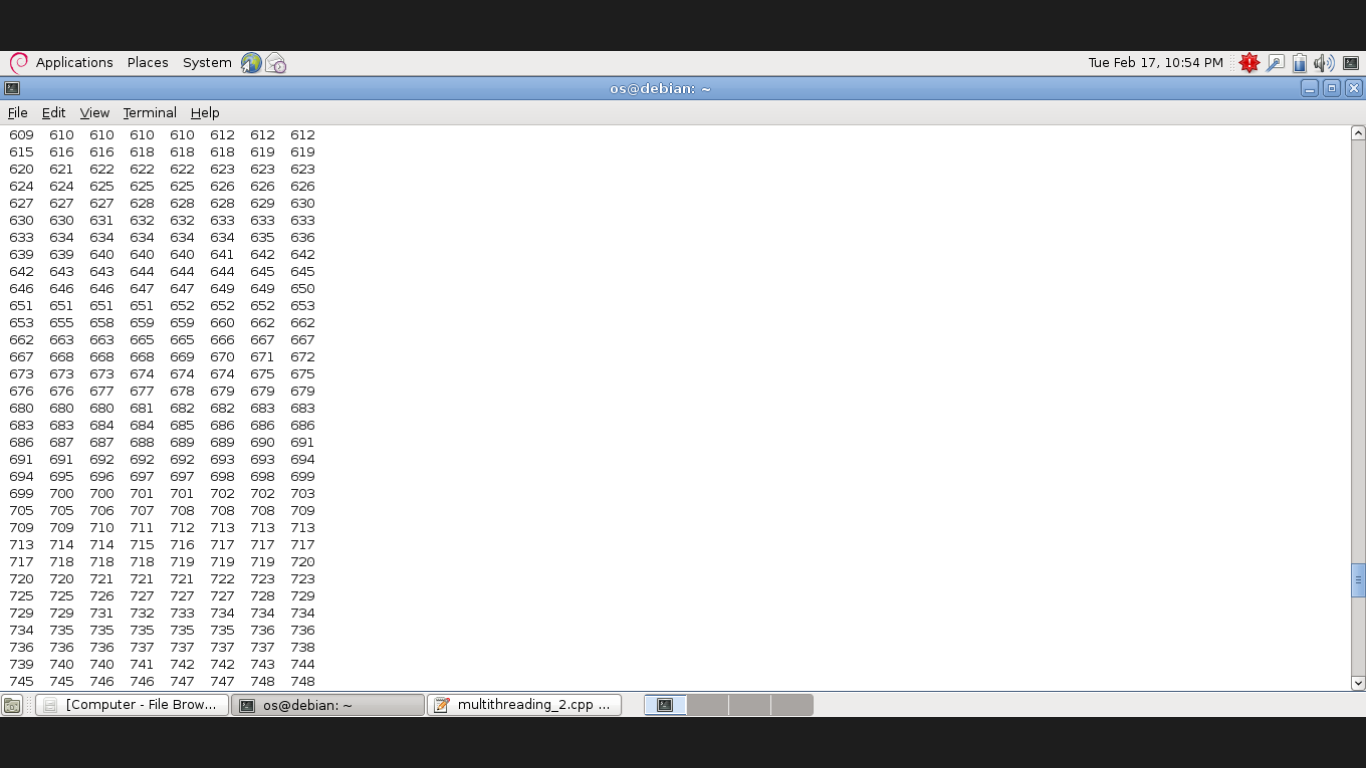
OUTPUT:

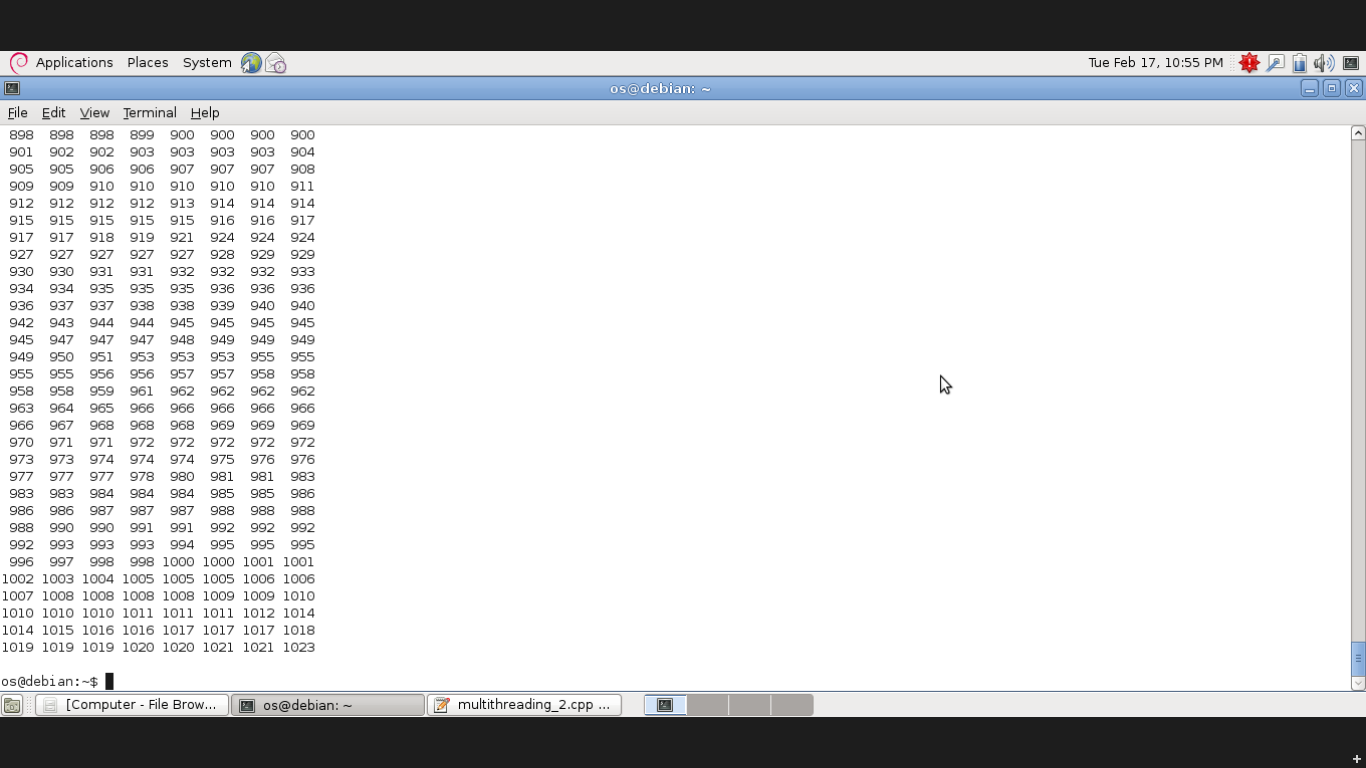












QUESTION 2:

Assume that a finite number of resources of a single resource type must be managed. Processes may ask for a number of these resources and will return them once finished. As an example, many commercial software packages provide a given number of ***licenses***, indicating the number of applications that may run concurrently.When the application is started, the license count is decremented.When the application is terminated, the license count is incremented. If all licenses are in use, requests to start the application are denied. Such requests will only be granted when an existing license holder terminates the application and a license is

returned. The following program segment is used to manage a finite number of instances of an available resource. The maximum number of resources and the number of available resources are declared as follows:

#define MAX RESOURCES 5

int available resources = MAX RESOURCES;

When a process wishes to obtain a number of resources, it invokes the

decrease count() function:

/\* decrease available resources by count resources \*/

/\* return 0 if sufficient resources available, \*/

/\* otherwise return -1 \*/

int decrease count(int count) *{*

if (available resources *<* count)

return -1;

else *{*

available resources -= count;

return 0;

*}*

*}*

When a process wants to return a number of resources, it calls the

increase count() function:

/\* increase available resources by count \*/

int increase count(int count) *{*

available resources += count;

return 0;

*}*

The preceding program segment produces a race condition.

The decrease count() function in the given exercise currently

returns 0 if sufficient resources are available and −1 otherwise. This leads to awkward programming for a process that wishes to obtain a number of resources:

while (decrease count(count) == -1);

Rewrite the resource-manager code segment using a monitor and condition variables so that the decrease count() function suspends the process until sufficient resources are available. This will allow a process to invoke decrease count() by simply calling

decrease count(count);

The process will return from this function call only when sufficient resources are available.

CODE:

#include <unistd.h>

#include <unistd.h>

#include <sys/types.h>

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <string.h>

#include <semaphore.h>

int available\_resources=5;

sem\_t resources\_avail;

void\* decrease\_count(void\* count)

{int \*a=(int \*) count;

if(available\_resources < \*a)

{

printf("\n request denied \n");

sem\_wait(&resources\_avail);

}

else{

available\_resources -= \*a;

}}

void\* increase\_count(void\* count)

{int \*d=(int \*) count;

available\_resources += \*d;

sem\_post(&resources\_avail);

printf("available resources are = %d \n",available\_resources);

}

void main()

{

pthread\_t w1,r1;

int g,h;

sem\_init(&(resources\_avail),0,0);

printf("enter the no. of resources to be obtained=");

scanf("%d",&g);

pthread\_create(&w1,NULL,decrease\_count,(void \*)&g);

printf("enter the no. of resources to be returned=");

scanf("%d",&h);

pthread\_create(&r1,NULL,increase\_count,(void \*)&h);

pthread\_join(w1,NULL);

pthread\_join(r1,NULL);

}

OUTPUT:

