DHRUV GARG 16 BCE 1190

DEADLOCKS AND SYNCHRONIZATION

LAB 8

SIMULATION 1: Banker's algorithm

CODE

```
#include<stdio.h>
void main()
{
   int k=0,output[10],d=0,t=0,ins[5],i,avail[5];
   int allocated[10][5],need[10][5],MAX[10][5];
   int pno,P[10],j,rz, count=0;

   printf("\n Enter the number of resources : ");
   scanf("%d", &rz);
   printf("\n enter the max instances of each
resources\n");
   for(i=0; i<rz; i++)
   {
      avail[i]=0;
      printf("%c= ",(i+97));
      scanf("%d",&ins[i]);
   }

   printf("\n Enter the number of processes : ");
   scanf("%d", &pno);
   printf("\n Enter the allocation matrix \n     ");

   for(i=0; i<rz; i++)
      printf(" %c",(i+97));
   printf("\n");

   for(i=0; i <pno; i++)
   {
      P[i]=i;
      printf("P[%d]  ",P[i]);
      for(j=0; j<rz; j++)
      {
         scanf("%d",&allocated[i][j]);
         avail[j]+=allocated[i][j];
      }
   }

   printf("\nEnter the MAX matrix \n     ");
   for(i=0; i<rz; i++)
   {
      printf(" %c",(i+97));
      avail[i]=ins[i]-avail[i];
   }
   printf("\n");
   for(i=0; i <pno; i++)
   {
      printf("P[%d]  ",i);
      for(j=0; j<rz; j++)
         scanf("%d", &MAX[i][j]);
   }

   printf("\n");
A:
   d=-1;
   for(i=0; i <pno; i++)
   {
      count=0;
```

```
    t=P[i];
    for(j=0; j<rz; j++)
    {
      need[t][j] = MAX[t][j]-allocated[t][j];
      if(need[t][j]<=avail[j])
        count++;
    }
    if(count==rz)
    {
      output[k++]=P[i];
      for(j=0; j<rz; j++)
        avail[j]+=allocated[t][j];
    }
    else
        P[++d]=P[i];
    }

    if(d!=-1)
    {
      pno=d+1;
      goto A;
    }
    printf("\t <");
    for(i=0; i<k; i++)
      printf(" P[%d] ",output[i]);
    printf(">\n\n");
  }
```

SCREENSHOT

```
vmdhruv@ubuntu:~/Documents$ gcc -o banker banker.c
vmdhruv@ubuntu:~/Documents$ ./banker

 Enter the number of resources : 3

 enter the max instances of each resources
a= 10
b= 7
c= 7

 Enter the number of processes : 3

 Enter the allocation matrix
      a b c
P[0]  3 2 1
P[1]  1 1 2
P[2]  4 1 2

Enter the MAX matrix
      a b c
P[0]  4 4 4
P[1]  3 4 5
P[2]  5 2 4

         < P[2]  P[0]  P[1] >

vmdhruv@ubuntu:~/Documents$
```

```
vmdhruv@ubuntu:~/Documents$ gcc -o banker banker.c
vmdhruv@ubuntu:~/Documents$ ./banker

 Enter the number of resources : 3

 enter the max instances of each resources
a= 12
b= 7
c= 8

 Enter the number of processes : 4

 Enter the allocation matrix
      a b c
P[0]  0 1 2
P[1]  2 3 2
P[2]  3 0 1
P[3]  0 2 2

Enter the MAX matrix
      a b c
P[0]  2 2 2
P[1]  4 5 3
P[2]  4 0 2
P[3]  1 3 3

         < P[0]  P[1]  P[2]  P[3] >

vmdhruv@ubuntu:~/Documents$
```

SIMULATION 2: Reader writer problem using SEMAPHORE

CODE

```
#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>


sem_t readCountAccess;

sem_t databaseAccess;

int readCount=0;


void *Reader(void *arg);

void *Writer(void *arg);

int main()

{

   int
i=0,NumberofReaderThread=0,NumberofWriterT
hread;

   sem_init(&readCountAccess,0,1);

   sem_init(&databaseAccess,0,1);

   pthread_t Readers_thr[100],Writer_thr[100];

   printf("\nEnter number of Readers
thread(MAX 10) : ");

   scanf("%d",&NumberofReaderThread);

   printf("\nEnter number of Writers thread(MAX
10) : ");

   scanf("%d",&NumberofWriterThread);

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

   {

pthread_create(&Readers_thr[i],NULL,Reader,(v
oid *)i);

   }

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

   {
pthread_create(&Writer_thr[i],NULL,Writer,(voi
d *)i);
```

```
    }
    for(i=0; i<NumberofWriterThread; i++)
    {
        pthread_join(Writer_thr[i],NULL);
    }
    for(i=0; i<NumberofReaderThread; i++)
    {
        pthread_join(Readers_thr[i],NULL);
    }
    sem_destroy(&databaseAccess);
    sem_destroy(&readCountAccess);
    return 0;
}

void * Writer(void *arg)
{
    sleep(1);
    int temp=(int)arg;
    printf("\nWRITER %d : WANTS TO ACCESS the database.",temp);
    sem_wait(&databaseAccess);
    printf("\nWRITER %d : WRITING to database.",temp);
    printf("\nWRITER %d : LEAVING the database.\n\n");
    sem_post(&databaseAccess);
}

void *Reader(void *arg)
{
    sleep(1);
    int temp=(int)arg;
    printf("\nREADER %d : WANTS TO ACCESS the database.",temp);
    sem_wait(&readCountAccess);
    readCount++;
    if(readCount==1)
    {
        sem_wait(&databaseAccess);
        printf("\nREADER %d : READING the database.",temp);
    }
    sem_post(&readCountAccess);
    sem_wait(&readCountAccess);
    readCount--;
    if(readCount==0)
    {
        printf("\nREADER %d : LEAVING the database.\n\n",temp);
        sem_post(&databaseAccess);
    }
    sem_post(&readCountAccess);
}
```

SCREENSHOTS

```
vmdhruv@ubuntu:~/Documents$ gcc -o readerWriter readerWriter.c -lpthread
readerWriter.c: In function 'main':
readerWriter.c:22:52: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
         pthread_create(&Readers_thr[i],NULL,Reader,(void *)i);
                                                    ^
readerWriter.c:26:51: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
         pthread_create(&Writer_thr[i],NULL,Writer,(void *)i);
                                                   ^
readerWriter.c: In function 'Writer':
readerWriter.c:42:5: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
     sleep(1);
     ^
readerWriter.c:43:14: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
     int temp=(int)arg;
              ^
readerWriter.c:47:12: warning: format '%d' expects a matching 'int' argument [-Wformat=]
     printf("\nWRITER %d : LEAVING the database.\n\n");
            ^
readerWriter.c: In function 'Reader':
readerWriter.c:53:14: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
     int temp=(int)arg;
              ^
vmdhruv@ubuntu:~/Documents$ ./readerWriter
```

```
vmdhruv@ubuntu:~/Documents$ ./readerWriter

Enter number of Readers thread(MAX 10) : 4

Enter number of Writers thread(MAX 10) : 5

WRITER 1 : WANTS TO ACCESS the database.
WRITER 1 : WRITING to database.
WRITER 0 : LEAVING the database.


WRITER 2 : WANTS TO ACCESS the database.
WRITER 2 : WRITING to database.
WRITER 0 : LEAVING the database.


WRITER 0 : WANTS TO ACCESS the database.
WRITER 0 : WRITING to database.
WRITER 0 : LEAVING the database.


WRITER 3 : WANTS TO ACCESS the database.
WRITER 3 : WRITING to database.
WRITER 0 : LEAVING the database.


WRITER 4 : WANTS TO ACCESS the database.
WRITER 4 : WRITING to database.
WRITER 0 : LEAVING the database.


READER 3 : WANTS TO ACCESS the database.
READER 3 : READING the database.
READER 3 : LEAVING the database.
```

```
READER 3 : WANTS TO ACCESS the database.
READER 3 : READING the database.
READER 3 : LEAVING the database.


READER 2 : WANTS TO ACCESS the database.
READER 2 : READING the database.
READER 2 : LEAVING the database.


READER 1 : WANTS TO ACCESS the database.
READER 1 : READING the database.
READER 1 : LEAVING the database.


READER 0 : WANTS TO ACCESS the database.
READER 0 : READING the database.
READER 0 : LEAVING the database.

vmdhruv@ubuntu:~/Documents$
```

DHRUV GARG 16 BCE 1190

SIMULATION 3: Dining philosopher's problem using MONITORS

CODE

NOTE: *This program required multiple files to be saved as a single project. I did not know how to compile a project using GCC on linux. Hence I used CodeBlocks on Windows.*

File 1: chopsticks.c

```
#define _XOPEN_SOURCE 500

#define _REENTRANT

#include <unistd.h>

#include <ctype.h>

#include <pthread.h>

#include "chopsticks.h"


#define LEFT (i+NTHREADS-1)%NTHREADS /* philo to the left */

#define RIGHT (i+1)%NTHREADS        /* philo to the right */

#define THINKING 1                  /* assign values to states */

#define HUNGRY 2

#define EATING 3


/* local state and synchronization variables */


pthread_cond_t  CV[NTHREADS];    /* one per philosopher */

pthread_mutex_t M;               /* mutual exclusion for the monitor */

int          state[NTHREADS]; /* state of each philosopher */


int

update_state (int i)

{

   if (state[i] == HUNGRY

         && state[LEFT] != EATING

         && state[RIGHT] != EATING)

   {

      state[i] = EATING;

      pthread_cond_signal (&CV[i]);

   }

   return 0;

}


void

chopsticks_init ()

{

   int i;

   pthread_mutex_init (&M, NULL);

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

   {

      pthread_cond_init (&CV[i], NULL);

      state[i] = THINKING;

   }

}


void

chopsticks_take (int i)

{

   pthread_mutex_lock (&M);     /* enter cs, lock mutex */

   state[i] = HUNGRY;          /* set philosopher's state to HUNGRY */

   update_state(i);            /* update_state philosopher */

   while (state[i] == HUNGRY)    /* loop while philosopher is hungry */
```

```
    pthread_cond_wait (&CV[i],&M);
  pthread_mutex_unlock (&M); /* exit cs, unlock mutex */
}

void
chopsticks_put (int i)
{
  pthread_mutex_lock (&M);    /* enter cs, lock mutex */
  state[i] = THINKING;
  update_state (LEFT);        /* update_state neighbors */
  update_state (RIGHT);
  pthread_mutex_unlock (&M);  /* exit cs, unlock mutex */
}
```

File 2: chopsticks.h

```
#define NTHREADS 5
/* the number of philosophers */

extern void chopsticks_init();
extern void chopsticks_take(int i);
extern void chopsticks_put(int i);
extern void chopsticks_finalize();
extern void chopsticks_emergency_stop();
```

File 3: philosophers_t.c

```
#define _XOPEN_SOURCE 500
#define _REENTRANT
#include <unistd.h>
#include <pthread.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
#include "chopsticks.h"

pthread_mutex_t outlock;   /* protects against output interleaving */
int nsteps, maxsteps = 0;  /* number of steps to run this test */
int eat_count[NTHREADS];    /* number of steps for each thread */

void trace(int i, char *s)
{
  pthread_mutex_lock(&outlock);
  if (strcmp (s, "eating") == 0)
  {
    fprintf(stdout,"\n   PHILOSOPHER %d is eating\n", i+1);
    fflush(stdout);
    eat_count [i]++;
  }
```

```
   if (nsteps++ > maxsteps)
   {
      /* don't exit while we are holding any
chopsticks */
      if (strcmp(s,"thinking") == 0)
      {
         pthread_mutex_unlock(&outlock);
         pthread_exit(0);
      }
   }
   pthread_mutex_unlock(&outlock);
}

void * philosopher_body(void *arg)
{
   int self = *(int *) arg;
   for (;;)
   {
      trace(self,"thinking");
      chopsticks_take(self);
      trace(self,"eating");
      chopsticks_put(self);
   }
}

int main()
{
   int i;
   pthread_t th[NTHREADS]; /* IDs of the
philospher threads */
   int   no[NTHREADS]; /* corresponding table
position numbers*/
   pthread_attr_t attr;
   for (i = 0; i < NTHREADS; i++) eat_count [i] = 0;

   pthread_mutex_init(&outlock, NULL);

   /* initialize the object chopsticks */
   chopsticks_init();
   fprintf(stdout,"\n\tDINING PHILOSOPHER's
PROBLEM - using MONITORS\n\n");
   fflush(stdout);
   fprintf(stdout,"\n   Enter number of steps to
run: ");
   fflush(stdout);
   fscanf(stdin,"%d",&maxsteps);

   pthread_attr_init (&attr);
   pthread_setconcurrency (4);

   pthread_attr_setscope (&attr,
PTHREAD_SCOPE_SYSTEM);
   /* set system-wide contention scope */

   /* start up the philosopher threads */
   for (i = 0; i < NTHREADS; i++)
   {
      no[i] = i;
      pthread_create(&th[i], NULL,
philosopher_body, (int *) &no[i]);
   }

   /* wait for all the threads to shut down */

   for (i = 0; i < NTHREADS; i++)
pthread_join(th[i], NULL);
```

```
    fprintf(stdout,"\n\n\t---NET COUNT OF ALL PHILOSOPHERS---\n\n");

    fflush(stdout);

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

    {

        fprintf (stderr, "\n   PHILOSHOPHER %d : ate %d times\n", i, eat_count [i]);

    }

    return 0;

}
```

SCREENSHOT

C:\Users\dhruv\Documents\CPP\Dining_philos\bin\Debug\Dining_philos.exe

```
        DINING PHILOSOPHER's PROBLEM - using MONITORS

   Enter number of steps to run: 10

   PHILOSOPHER 1 is eating

   PHILOSOPHER 2 is eating

   PHILOSOPHER 4 is eating

   PHILOSOPHER 1 is eating

   PHILOSOPHER 3 is eating

   PHILOSOPHER 5 is eating

   PHILOSOPHER 2 is eating

        ---NET COUNT OF ALL PHILOSOPHERS---

   PHILOSHOPHER 1 : ate 2 times

   PHILOSHOPHER 2 : ate 2 times

   PHILOSHOPHER 3 : ate 1 times

   PHILOSHOPHER 4 : ate 1 times

   PHILOSHOPHER 5 : ate 1 times

Process returned 0 (0x0)   execution time : 2.926 s
Press any key to continue.
```