

Name: Mohammad Abubakr Khanooni

Cse a roll no 27

Practical 7 operating system

Aim:

Write C programs to implement threads and semaphores for process synchronisation.

Prac 7a-threads:

1.A simple C program to demonstrate use of pthread basic functions and to implement multiple threads with global and static variables

Code:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
// Let us create a global variable to change it in threads
int g = 0;
// The function to be executed by all threads
void *myThreadFun(void *vargp) {
    // Store the value argument passed to this thread
    int myid = (int)vargp;
    // Let us create a static variable to observe its changes
    static int s = 0;
    // Change static and global variables
    ++s;
    ++g;
    // Print the argument, static and global variables
    printf("Thread ID : %d,Static : %d, Global : %d\n", myid, ++s, ++g);
}
int main() {
    int i;
    pthread_t tid;
    // Let us create 3 Threads
    for (i = 0; i < 3; i++)
        pthread_create(&tid, NULL, myThreadFun, (void *)i);
    pthread_exit(NULL);
    return 0;
}
```

o/p:

```

~/prac7-os$ gcc main.c
main.c: In function 'myThreadFun':
main.c:9:14: warning: cast from pointer to integer of different
size [-Wpointer-to-int-cast]
   9 |     int myid = (int)vargp;
     |                  ^
main.c: In function 'main':
main.c:23:45: warning: cast to pointer from integer of differen
t size [-Wint-to-pointer-cast]
   23 |     pthread_create(&tid, NULL, myThreadFun, (void *)i);
     |                                           ^
~/prac7-os$ ./a.out
Thread ID : 0,Static : 2, Global : 2
Thread ID : 1,Static : 4, Global : 4
Thread ID : 2,Static : 6, Global : 6
~/prac7-os$ 

```

2.To demonstrate thread system calls

Code:

```

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
pthread_t tid[2];
void *doSomething(void *arg) {
    unsigned long i = 0;
    pthread_t id = pthread_self();
    if (pthread_equal(id, tid[0]))
        printf("\nFirst Thread Processing \n");
    else
        printf("\nSecond Thread Processing \n");
    for (i = 0; i < (0xFFFFFFFF); i++)
        ;
    return NULL;
}
int main(void) {
    int i = 0;
    int err;
    while (i < 2) {
        err = pthread_create(&tid[i], NULL, &doSomething, NULL);
        if (err != 0)
            printf("\nCan't create Thread : [%s]", strerror(err));
        else
            printf("\n Thread created successfully\n");
        i++;
    }
}

```

```
}  
sleep(5);  
return 0;  
}
```

o/p:

```
~/prac7-os$ gcc 2.c  
  
~/prac7-os$  
~/prac7-os$ ./a.out  
  
Thread created successfully  
  
First Thread Processing  
  
Second Thread Processing  
  
Thread created successfully  
█
```

3.Matrix Multiplication using Threads

Code:

```
#include <pthread.h>  
#include <stdio.h>  
#include <stdlib.h>  
#define size 3  
int A[size][size] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};  
int B[size][size] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};  
int C[size][size];  
typedef struct {  
    int row;  
    int col;  
} m;  
void *mult(void *args) {  
    m *index = (m *)args;  
    int r = index->row;  
    int c = index->col;  
    for (int i = 0; i < size; i++) {  
        C[r][c] += A[r][i] * B[i][c];  
    }  
}  
int main() {
```

```

pthread_t t[size][size];
printf("A matrix :\n");
for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        printf("%d ", A[i][j]);
    }
    printf("\n");
}
printf("B matrix :\n");
for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        printf("%d ", B[i][j]);
    }
    printf("\n");
}
for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        m *index = (m *)malloc(sizeof(m));
        index->row = i;
        index->col = j;
        pthread_create(&t[i][j], NULL, mult, (void *)index);
    }
}
for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        pthread_join(t[i][j], NULL);
    }
}
printf("Answer of Multiplication :\n");
for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        printf("%d ", C[i][j]);
    }
    printf("\n");
}
}

```

o/p:

```

~/prac7-os$ gcc 3.c
~/prac7-os$ ./a.out
A matrix :
1 2 3
1 2 3
1 2 3
B matrix :
1 2 3
4 5 6
7 8 9
Answer of Multiplication :
30 36 42
30 36 42
30 36 42
~/prac7-os$ █

```

4.Linear search using Multi-threading (use n number of threads)

Code:

```

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define max 20
#define thread_max 4
int a[max] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
              11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
int key = 15;
int flag = 0, i;
int current_thread = 0;
void *LinearSearch(void *args) {
    int num = current_thread++;
    for (num *(max / 4); i < ((num + 1) * (max / 4)); i++) {
        if (a[i] == key)
            flag = 1;
    }
}
int main() {
    pthread_t thread[thread_max];
    for (int i = 0; i < thread_max; i++)
        pthread_create(&thread[i], NULL, LinearSearch, (void *)NULL);
    for (int i = 0; i < thread_max; i++)
        pthread_join(thread[i], NULL);
    if (flag == 1)
        printf("Element found (%d)\n", key);
    else
        printf("Element not present(%d)\n", key);
}

```

o/p:

```
~/prac7-os$ gcc 4.c
~/prac7-os$ ./a.out
Element found (15)
~/prac7-os$
```

5.To find the maximum and minimum element in an array using Multithreading (for 100 to 200 numbers or more and create 10 or more threads)

Code:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define size 100
#define thread_max 10
int a[size];
int max = -9999;
int min = 9999;
void *find(void *args) {
    int tid = (int *)args;
    int subarray_size = size / thread_max;
    int s_index = tid * subarray_size;
    int e_index = s_index + subarray_size;
    int MAX = -9999;
    int MIN = 9999;
    for (int i = s_index; i < e_index; i++) {
        if (a[i] > MAX)
            MAX = a[i];
        else if (a[i] < MIN)
            MIN = a[i];
    }
    if (MAX > max)
        max = MAX;
    if (MIN < min)
        min = MIN;
    pthread_exit(NULL);
}
int main() {
    pthread_t t[thread_max];
    for (int i = 0; i < size; i++) {
        a[i] = rand() % 1000;
    }
    for (int i = 0; i < thread_max; i++) {
        pthread_create(&t[i], NULL, find, i);
    }
}
```

```

for (int i = 0; i < thread_max; i++) {
    pthread_join(t[i], NULL);
}
printf("Maximum Number of the array is %d\n", max);
printf("Minimum Number of the array is %d\n", min);
}

```

o/p:

```

~/prac7-os$ gcc 5.c
5.c: In function 'find':
5.c:10:13: warning: initialization of 'int' from 'int *' makes
integer from pointer without a cast [-Wint-conversion]
10 |     int tid = (int *)args;
    |               ^
5.c: In function 'main':
5.c:34:39: warning: passing argument 4 of 'pthread_create' make
s pointer from integer without a cast [-Wint-conversion]
34 |     pthread_create(&t[i], NULL, find, i);
    |                                     ^
    |                                     |
    |                                    int
In file included from 5.c:1:
/nix/store/1gf2flfqnpqbr1b4p4qz2f72y42bs56r-gcc-11.3.0/lib/gcc/
x86_64-unknown-linux-gnu/11.3.0/include-fixed/pthread.h:214:45:
note: expected 'void * restrict' but argument is of type 'int'
214 |     void *__restrict __arg) __THR
    |     ^~~~~~
OWNL __nonnull ((1, 3));
~~~~~^~~~~~

~/prac7-os$ ./a.out
Maximum Number of the array is 996
Minimum Number of the array is 11
~/prac7-os$

```

6.Example without synchronization FOR PRODUCER CONSUMER PROBLEM

Code:

```

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
void *producer(); // the thread
void *consumer(); // the thread
int main() {
    pthread_t ptid, ctid; // Thread Id for Producer &
    Consumer pthread_create(&ptid, NULL, producer, NULL); // Producer
    pthread_create(&ctid, NULL, consumer, NULL); // Consumer
    pthread_join(ptid, NULL);
    pthread_join(ctid, NULL);
}

```

```
// The thread will begin control in this function
void *producer(void *param) {
    do {
        printf("I am Producer\n");
    } while (1);
    pthread_exit(0);
}
// The thread will begin control in this function
void *consumer(void *param) {
    do {
        printf("I am Consumer\n");
    } while (1);
    pthread_exit(0);
}
```

o/p:


```

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define BufferSize 10
void *Producer();
void *Consumer();
int BufferIndex = -1;
char BUFFER[10];
pthread_cond_t Buffer_Empty = PTHREAD_COND_INITIALIZER;
pthread_cond_t Buffer_Full = PTHREAD_COND_INITIALIZER;
pthread_mutex_t mVar = PTHREAD_MUTEX_INITIALIZER;
int main() {
    pthread_t ptid, ctid;
    pthread_create(&ptid, NULL, Producer, NULL);
    pthread_create(&ctid, NULL, Consumer, NULL);
    pthread_join(ptid, NULL);
    pthread_join(ctid, NULL);
    return 0;
}
void *Producer() {
    int i;
    for (i = 0; i < 15; i++) {
        pthread_mutex_lock(&mVar);
        if (BufferIndex == BufferSize - 1)
            pthread_cond_wait(&Buffer_Empty, &mVar);
        BUFFER[++BufferIndex] = '#';
        printf("Produce : %d \n", BufferIndex);
        pthread_mutex_unlock(&mVar);
        pthread_cond_signal(&Buffer_Full);
    }
}
void *Consumer() {
    int i;
    for (i = 0; i < 15; i++) {
        pthread_mutex_lock(&mVar);
        if (BufferIndex == -1) {
            pthread_cond_wait(&Buffer_Full, &mVar);
        }
        printf("Consume : %d \n", BufferIndex--);
        pthread_mutex_unlock(&mVar);
        pthread_cond_signal(&Buffer_Empty);
    }
}

```

o/p:

```
~/prac71$ gcc 7.c
~/prac71$ ./a.out
Produce : 0
Produce : 1
Produce : 2
Produce : 3
Produce : 4
Produce : 5
Produce : 6
Produce : 7
Produce : 8
Produce : 9
Consume : 9
Consume : 8
Consume : 7
Consume : 6
Consume : 5
Consume : 4
Consume : 3
Consume : 2
Consume : 1
Consume : 0
Produce : 0
Produce : 1
Produce : 2
Produce : 3
Produce : 4
Consume : 4
Consume : 3
Consume : 2
Consume : 1
Consume : 0
~/prac71$
```

8. Readers Writers Problem solved with mutex and pthread.

Code:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define readers 5
#define writers 2
int sharedResource = 0;
pthread_mutex_t resourceLock = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t readerCountLock = PTHREAD_MUTEX_INITIALIZER;
```

```

int readerCount = 0;
void *reader(void *arg) {
    int readerId = *(int *)arg;
    pthread_mutex_lock(&readerCountLock);
    readerCount++;
    if (readerCount == 1) {
        pthread_mutex_lock(&resourceLock);
    }
    pthread_mutex_unlock(&readerCountLock);
    printf("Reader %d reads shared resource: %d\n", readerId, sharedResource);
    pthread_mutex_lock(&readerCountLock);
    readerCount--;
    if (readerCount == 0) {
        pthread_mutex_unlock(&resourceLock);
    }
    pthread_mutex_unlock(&readerCountLock);
    pthread_exit(NULL);
}

void *writer(void *arg) {
    int writerId = *(int *)arg;
    pthread_mutex_lock(&resourceLock);
    sharedResource = writerId;
    printf("Writer %d writes shared resource: %d\n", writerId, sharedResource);
    pthread_mutex_unlock(&resourceLock);
    pthread_exit(NULL);
}

int main() {
    pthread_t readerThreads[readers];
    pthread_t writerThreads[writers];
    int readerIds[readers];
    int writerIds[writers];
    for (int i = 0; i < writers; i++) {
        writerIds[i] = i + 1;
        pthread_create(&writerThreads[i], NULL, writer, &writerIds[i]);
    }
    for (int i = 0; i < readers; i++) {
        readerIds[i] = i + 1;
        pthread_create(&readerThreads[i], NULL, reader, &readerIds[i]);
    }
    for (int i = 0; i < readers; i++) {
        pthread_join(readerThreads[i], NULL);
    }
    for (int i = 0; i < writers; i++) {
        pthread_join(writerThreads[i], NULL);
    }
    return 0;
}

```

o/p:

```
~/prac71$ gcc 8.c
~/prac71$ ./a.out
Reader 3 reads shared resource: 0
Writer 1 writes shared resource: 1
Reader 4 reads shared resource: 1
Reader 5 reads shared resource: 1
Reader 2 reads shared resource: 1
Writer 2 writes shared resource: 2
Reader 1 reads shared resource: 2
~/prac71$ □
```

Result:

Programs for c in linux using threads has been executed .

Thank you!