CS 332/532 Systems Programming

Lecture 32

Thread / 2

Professor : Mahmut Unan – UAB CS

Agenda

Threads
Thread Syncronization
Mutex

exercise 1

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>


void *someFuncToCreateThread(void *someValue)
{
    sleep(2);
    printf("I am inside the thread \n");
    return NULL;
}

int main()
{
    pthread_t thread_id;
    printf("I am inside the main function\n");
    pthread_create(&thread_id, NULL, someFuncToCreateThread, NULL);
    pthread_join(thread_id, NULL);
    printf("Back to the main function\n");
    exit(0);
}

```

compile & run

To compile a multithreaded program, we will be using gcc and we need to link it with the pthreads library.

```
(base) mahmutunan@MacBook-Pro lecture31 % gcc exercise1.c -o exercise1 -lpthread
(base) mahmutunan@MacBook-Pro lecture31 % ./exercise1
I am inside the main function
I am inside the thread
Back to the main function
(base) mahmutunan@MacBook-Pro lecture31 %
```

exercise 2

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

void *function1(void *someValue)
{
    while(1==1) {
        sleep(1);
        printf("function 1 \n");
    }
}
void function2()
{
    while(1==1) {
        sleep(2);
        printf("function 2\n");
    }
}

int main()
{
    pthread_t thread_id;
    printf("I am inside the main function\n");
    pthread_create(&thread_id, NULL, function1, NULL);
    function2();
    exit(0);
}
```

compile & run

```
(base) mahmutunan@MacBook-Pro lecture31 % gcc exercise2.c -o exercise2 -lpthread
(base) mahmutunan@MacBook-Pro lecture31 % ./exercise2
I am inside the main function
function 1
function 2
function 1
function 1
function 2
function 1
function 1
function 2
function 1
function 1
function 2
function 1
function 1
```

exercise 3

```c
int globalVar = 50; //define a global variable

void *someFuncToCreateThread(void *someValue)
{
    int *threadId = (int *)someValue; // Store the value argument passed to this thread

    //define a static and a local variable
    static int staticVar = 75;
    int localVar = 10;

    // let's change the variables
    globalVar +=100;
    staticVar +=100;
    localVar +=100;
    printf("id =%d,global = %d,  local = %d, static =%d, \n", *threadId, globalVar,localVar ,staticVar);

    return NULL;
}

int main()
{
    int i;
    pthread_t thread_id;
    for (i = 0; i < 4; i++)
        pthread_create(&thread_id, NULL, someFuncToCreateThread, (void *)&thread_id);
    pthread_exit(NULL);

}
```

compile & run

```
(base) mahmutunan@MacBook-Pro lecture31 % gcc exercise3.c -o exercise3 -lpthread
(base) mahmutunan@MacBook-Pro lecture31 % ./exercise3
id =151261184,global = 150,  local = 110, static =175,
id =151261184,global = 150,  local = 110, static =175,
id =151261184,global = 250,  local = 110, static =275,
id =151261184,global = 250,  local = 110, static =275,
(base) mahmutunan@MacBook-Pro lecture31 % _
```

Remember, global and static variables are stored in data segment.

All threads share data segment, so they are shared by all threads.

pthread1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int nthreads;

void *compute(void *arg) {
  long tid = (long)arg;

  printf("Hello, I am thread %ld of %d\n", tid, nthreads);

  return (NULL);
}

int main(int argc, char **argv) {
  long i;
  pthread_t *tid;

  if (argc != 2) {
    printf("Usage: %s <# of threads>\n",argv[0]);
    exit(-1);
  }

  nthreads = atoi(argv[1]); // no. of threads
```

```
25
26      // allocate vector and initialize
27      tid = (pthread_t *)malloc(sizeof(pthread_t)*nthreads);
28
29      // create threads
30      for ( i = 0; i < nthreads; i++)
31        pthread_create(&tid[i], NULL, compute, (void *)i);
32
33      // wait for them to complete
34      for ( i = 0; i < nthreads; i++)
35        pthread_join(tid[i], NULL);
36
37      printf("Exiting main program\n");
38
39      return 0;
40    }
41
```

```
(base) mahmutunan@MacBook-Pro lecture31 % gcc pthread1.c -o exercise4 -lpthread
(base) mahmutunan@MacBook-Pro lecture31 % ./exercise4 4
Hello, I am thread 0 of 4
Hello, I am thread 1 of 4
Hello, I am thread 2 of 4
Hello, I am thread 3 of 4
Exiting main program
```

pthread2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int nthreads;

void *compute(void *arg) {
  long tid = (long)arg;
  pthread_t pthread_id = pthread_self();

  printf("Hello, I am thread %ld of %d, pthread_self() = %lu (0x%lx)\n",
         tid, nthreads, (unsigned long)pthread_id, (unsigned long)pthread_id);

  return (NULL);
}

int main(int argc, char **argv) {
  long i;
  pthread_t *tid;
  pthread_t pthread_id = pthread_self();
```

```
21
22    if (argc != 2) {
23      printf("Usage: %s <# of threads>\n",argv[0]);
24      exit(-1);
25    }
26
27    nthreads = atoi(argv[1]); // no. of threads
28
29    // allocate vector and initialize
30    tid = (pthread_t *)malloc(sizeof(pthread_t)*nthreads);
31
32    // create threads
33    for ( i = 0; i < nthreads; i++)
34      pthread_create(&tid[i], NULL, compute, (void *)i);
35
36    for ( i = 0; i < nthreads; i++)
37      printf("tid[%ld] = %lu (0x%lx)\n", i, tid[i], tid[i]);
38
39    printf("Hello, I am main thread. pthread_self() = %lu (0x%lx)\n",
40           (unsigned long)pthread_id, (unsigned long)pthread_id);
41
42    // wait for them to complete
43    for ( i = 0; i < nthreads; i++)
44      pthread_join(tid[i], NULL);
45
46    printf("Exiting main program\n");
47
48    return 0;
49  }
```

```
(base) mahmutunan@MacBook-Pro lecture31 % ./exercise5 4
tid[0] = 123145541038080 (0x70000e3ab000)
tid[1] = 123145541574656 (0x70000e42e000)
tid[2] = 123145542111232 (0x70000e4b1000)
tid[3] = 123145542647808 (0x70000e534000)
Hello, I am main thread. pthread_self() = 4365594048 (0x10435adc0)
Hello, I am thread 1 of 4, pthread_self() = 123145541574656 (0x70000e42e000)
Hello, I am thread 2 of 4, pthread_self() = 123145542111232 (0x70000e4b1000)
Hello, I am thread 0 of 4, pthread_self() = 123145541038080 (0x70000e3ab000)
Hello, I am thread 3 of 4, pthread_self() = 123145542647808 (0x70000e534000)
Exiting main program
```

pthread3.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

typedef struct foo {
    pthread_t ptid;  /* thread id returned by pthread_create */
    int tid;         /* user managed thread id (0 through nthreads-1) */
    int nthreads;    /* total no. of threads created */
} FOO;

void *compute(void *args) {
  FOO *info = (FOO *)args;
  printf("Hello, I am thread %d of %d\n", info->tid, info->nthreads);

  return (NULL);
}

int main(int argc, char **argv) {
  int i, nthreads;
  FOO *info;

  if (argc != 2) {
    printf("Usage: %s <# of threads>\n",argv[0]);
    exit(-1);
  }
```

```
27      nthreads = atoi(argv[1]); // no. of threads
28
29      // allocate structure
30      info = (FOO *)malloc(sizeof(FOO)*nthreads);
31
32      // create threads
33      for ( i = 0; i < nthreads; i++) {
34        info[i].tid = i;
35        info[i].nthreads = nthreads;
36        pthread_create(&info[i].ptid, NULL, compute, (void *)&info[i]);
37      }
38
39      // wait for them to complete
40      for ( i = 0; i < nthreads; i++)
41        pthread_join(info[i].ptid, NULL);
42
43      free(info);
44      printf("Exiting main program\n");
45
46      return 0;
47  }
```

```
(base) mahmutunan@MacBook-Pro lecture31 % gcc pthread3.c -o exercise6 -lpthread
(base) mahmutunan@MacBook-Pro lecture31 % ./exercise6 4
Hello, I am thread 1 of 4
Hello, I am thread 0 of 4
Hello, I am thread 2 of 4
Hello, I am thread 3 of 4
Exiting main program
```

Thread Synchronization using Mutexes

- We can use the mutexes provided by the Pthreads library to control access to critical sections of the program and provide synchronization across the threads.
- We will use the example of computing the sum of the elements in a vector to illustrate the use of mutexes.
- We have a vector of N elements, we would like to assign each thread to compute the partial sum of N/P elements (where P is the number of threads), then we will update the shared global variable *sum* with the partial sums using mutex locks.

- The API for the mutex lock and unlock functions are shown below:

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);

int pthread_mutex_unlock(pthread_mutex_t
*mutex);
```

pthread_sum.c

- The *mutex* variable is of type *pthread_mutex_t* and can be initially statically by assigning the value *PTHREAD_MUTEX_INITIALIZER*.
- Note that the mutex variable must be declared in global scope since it will be shared among multiple threads.
- A mutex can also be initialized dynamically using the function *pthread_mutex_init*.
- The *pthread_mutex_destroy* function can be used to destroy the mutex that was initialized using *pthread_mutex_init*.

pthread_sum.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;

double *a=NULL, sum=0.0;
int    N, size;

void *compute(void *arg) {
    int myStart, myEnd, myN, i;
    long tid = (long)arg;

    // determine start and end of computation for the current thread
    myN = N/size;
    myStart = tid*myN;
    myEnd = myStart + myN;
    if (tid == (size-1)) myEnd = N;

    // compute partial sum
    double mysum = 0.0;
    for (i=myStart; i<myEnd; i++)
      mysum += a[i];

    // grab the lock, update global sum, and release lock
    pthread_mutex_lock(&mutex);
    sum += mysum;
    pthread_mutex_unlock(&mutex);

    return (NULL);
}
```

pthread_sum.c

```
34  int main(int argc, char **argv) {
35      long i;
36      pthread_t *tid;
37
38      if (argc != 3) {
39         printf("Usage: %s <# of elements> <# of threads>\n",argv[0]);
40         exit(-1);
41      }
42
43      N = atoi(argv[1]); // no. of elements
44      size = atoi(argv[2]); // no. of threads
45
46      // allocate vector and initialize
47      tid = (pthread_t *)malloc(sizeof(pthread_t)*size);
48      a = (double *)malloc(sizeof(double)*N);
49      for (i=0; i<N; i++)
50        a[i] = (double)(i + 1);
51
52      // create threads
53      for ( i = 0; i < size; i++)
54        pthread_create(&tid[i], NULL, compute, (void *)i);
55
56      // wait for them to complete
57      for ( i = 0; i < size; i++)
58        pthread_join(tid[i], NULL);
59
60      printf("The total is %g, it should be equal to %g\n",
61             sum, ((double)N*(N+1))/2);
62
63      return 0;
64  }
```

pthread_sum.c

- compile & run

```
(base) mahmutunan@MacBook-Pro lecture32 % gcc pthread_sum.c -o exercise7 -lpthread
(base) mahmutunan@MacBook-Pro lecture32 % ./exercise7 200 4
The total is 20100, it should be equal to 20100
(base) mahmutunan@MacBook-Pro lecture32 % ./exercise7 1000 4
The total is 500500, it should be equal to 500500
(base) mahmutunan@MacBook-Pro lecture32 % _
```

pthread_sum2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

double *a=NULL, *partialsum;
int    N, nthreads;

void *compute(void *arg) {
    int myStart, myEnd, myN, i;
    long tid = (long)arg;

    // determine start and end of computation for the current thread
    myN = N/nthreads;
    myStart = tid*myN;
    myEnd = myStart + myN;
    if (tid == (nthreads-1)) myEnd = N;

    // compute partial sum
    double mysum = 0.0;
    for (i=myStart; i<myEnd; i++)
      mysum += a[i];

    partialsum[tid] = mysum;
    return (NULL);
}

```

pthread_sum2.c

```
27
28  int main(int argc, char **argv) {
29      long i;
30      pthread_t *tid;
31      double sum = 0.0;
32
33      if (argc != 3) {
34         printf("Usage: %s <# of elements> <# of threads>\n",argv[0]);
35         exit(-1);
36      }
37
38      N = atoi(argv[1]); // no. of elements
39      nthreads = atoi(argv[2]); // no. of threads
40
41      // allocate vector and initialize
42      tid = (pthread_t *)malloc(sizeof(pthread_t)*nthreads);
43      a = (double *)malloc(sizeof(double)*N);
44      partialsum = (double *)malloc(sizeof(double)*nthreads);
45      for (i=0; i<N; i++)
46        a[i] = (double)(i + 1);
47
48      // create threads
49      for ( i = 0; i < nthreads; i++)
50        pthread_create(&tid[i], NULL, compute, (void *)i);
51
```

pthread_sum2.c

```
51
52      // wait for them to complete
53      for ( i = 0; i < nthreads; i++)
54        pthread_join(tid[i], NULL);
55
56      for ( i = 0; i < nthreads; i++)
57        sum += partialsum[i];
58
59      printf("The total is %g, it should be equal to %g\n",
60             sum, ((double)N*(N+1))/2);
61
62      free(tid);
63      free(a);
64      free(partialsum);
65
66      return 0;
67  }
68
```

```
(base) mahmutunan@MacBook-Pro lecture32 % gcc pthread_sum2.c -o exercise8 -lpthread
(base) mahmutunan@MacBook-Pro lecture32 % ./exercise8 1000 4
The total is 500500, it should be equal to 500500
(base) mahmutunan@MacBook-Pro lecture32 % ./exercise8 200 4
The total is 20100, it should be equal to 20100
```

Extra exercises

```c
pthread_t tid[2];
int counter;

void* trythis(void* arg)
{
    unsigned long i = 0;
    counter += 1;
    printf("\n Job %d has started\n", counter);

    for (i = 0; i < (0xFFFFFFFF); i++)
        ;
    printf("\n Job %d has finished\n", counter);

    return NULL;
}

int main(void)
{
    int i = 0;
    int error;

    while (i < 2) {
        error = pthread_create(&(tid[i]), NULL, &trythis, NULL);
        if (error != 0)
            printf("\nThread can't be created : [%s]", strerror(error));
        i++;
    }

    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);

    return 0;
}
```

https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/?ref=rp

```
(base) mahmutunan@MacBook-Pro lecture32 % gcc exercise2.c -o exercise2 -lpthread
(base) mahmutunan@MacBook-Pro lecture32 % ./exercise2

 Job 1 has started

 Job 2 has started

 Job 2 has finished

 Job 2 has finished
```


https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/?ref=rp

```c
#include <pthread.h>
#include <stdio.h>
#include <string.h>


pthread_t tid[2];
int counter;
pthread_mutex_t lock;

void* trythis(void* arg)
{
    pthread_mutex_lock(&lock);

    unsigned long i = 0;
    counter += 1;
    printf("\n Job %d has started\n", counter);

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

    printf("\n Job %d has finished\n", counter);

    pthread_mutex_unlock(&lock);

    return NULL;
}
```


https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/?ref=rp

```c
int main(void)
{
    int i = 0;
    int error;

    if (pthread_mutex_init(&lock, NULL) != 0) {
        printf("\n mutex init has failed\n");
        return 1;
    }

    while (i < 2) {
        error = pthread_create(&(tid[i]),
                               NULL,
                               &trythis, NULL);
        if (error != 0)
            printf("\nThread can't be created :[%s]",
                   strerror(error));
        i++;
    }

    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    pthread_mutex_destroy(&lock);

    return 0;
}
```

https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/?ref=rp

```
(base) mahmutunan@MacBook-Pro lecture32 % gcc exercise3.c -o exercise3 -lpthread
(base) mahmutunan@MacBook-Pro lecture32 % ./exercise3

 Job 1 has started

 Job 1 has finished

 Job 2 has started

 Job 2 has finished
```


https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/?ref=rp