NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
ISLAMABAD

OPERATING SYSTEMS LAB SPRING 2023

Lab Manual 09
Thread Attributes

1 THREAD STATES

Threads are created in two ways

1. Joinable Threads
2. Detached Threads

2 JOINABLE OR NOT?

1. When a thread is created, one of its attributes defines whether it is joinable or detached.
2. Only threads that are created as joinable can be joined. If a thread is created as detached, it can never be joined.
3. When we create a thread with default attributes a joinable thread is created.

3 THREAD DETACHING

1. By default threads are created joinable.
2. Instead of waiting for a thread, the executing thread can specify that
   (a) It does not require a return value.
   (b) Or any explicit synchronization with that thread.
3. The pthread_detach() routine can be used to explicitly detach a thread even though it was created as joinable.

4. After this call, no thread can wait for detached thread and it executes independently until termination.

```
int pthread_detach(pthread_t tid);
```

5. There is no converse routine.

6. Threads can detach themselves by calling pthread_detach with an argument of pthread_self().

EXAMPLE 01

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void*thread_function(void*arg);
char message[] = "iam thread 1";
int main(){
        pthread_t tid;
        pthread_create(&tid,NULL,thread_function,(void*)message);
        // In 2nd run make following statement comment then see
        pthread_detach(tid);
        int joinret;
        joinret=pthread_join(tid,NULL);

        if(joinret==0)    //when join successfull
                printf("join was successfull:The main thread was
                                        waiting for thread 1\n");
        else
                printf("Join failed:The main thread is not
                                        waiting for thread 1\n");
        printf("The main thread finished, bye!\n");
        pthread_exit(NULL);
}

void *thread_function(void *arg){
        sleep(2);
        printf("thread_function is running. Argument was\" %s\"\n",
                                        (char*)arg);
        printf("thread 1 awaked from sleep, and exiting now\n");
        pthread_exit(NULL);
}
```

EXAMPLE 02

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *thread_function1(void *arg);
void *thread_function2(void *arg);
char message1[] = "iam thread 1";char message2[] = "iam thread 2";
pthread_t tid1,tid2;
int main(){
        pthread_create(&tid1,NULL,thread_function1,(void*)message1);
```

```
        pthread_create (&tid2, NULL, thread_function2, (void*)message2);
        pthread_detach(tid1);
        int joinret;
        joinret=pthread_join(tid1,NULL);

         //when join successfull
        if(joinret==0)
           printf("join was successfull:The main thread was waiting for thread 1\n");
        else
           printf("Join failed:The main thread is not waiting for thread 1\n");

        pthread_join(tid2,NULL);
        printf("The main received exit status from tid2, now exiting, bye!\n");
        pthread_exit(NULL);
}

void *thread_function1(void *arg){
        sleep(2);
        printf("thread_function 1 is running. Argument was %s\n", (char *)arg);
        sleep(5);
        printf("thread 1 awaked from sleep, and exiting now\n");
        pthread_exit(NULL);
}

void *thread_function2(void *arg){
        sleep(1);
        int joinret;
        joinret=pthread_join(tid1,NULL);

        if(joinret==0)    //when join successfull
           printf("join was successfull:The thread 2 was waiting for thread 1\n");
        else
           printf("join failed:The thread 2 is not waiting for thread 1\n");
        printf("thread_function 2 is running. Argument was %s\n", (char *)arg);
        printf("thread 2 is exiting now\n");
        pthread_exit(NULL);
}
```

4 THREAD ATTRIBUTES

The pthread interface allows us to fine-tune the behavior of threads using the pthread_attr_t structure by modifying the default attributes.

1. An initialization function exists to set the attributes to their default values.
2. Another function exists to destroy the attributes object. If the initialization func- tion allocated any resources associated with the attributes object, the destroy func-tion frees those resources.
3. Each attribute has a function to get the value of the attribute from the attribute object.

4. Each attribute has a function to set the value of the attribute. In this case, the value is passed as an argument, by value.

5 Thread Detaching Using Attributes

To explicitly create a thread as joinable or detached, the attr argument in the pthread_create() routine is used. The typical 4 step process is:

1. Declare a pthread attribute variable of the pthread_attr_t data type.
2. Initialize the attribute variable with pthread_attr_init().
3. Set the attribute detached status with pthread_attr_setdetachstate().
4. When done, free library resources used by the attribute with pthread_attr_destroy().

Important Points

- A detach thread can also be created using thread attributes.

```
pthread_attr_t thread_attr;
```

- Calling pthread_attr_init, the pthread_attr_t structure contains the default values for all the thread attributes supported by the implementation.

```
pthread_attr_init(&thread_attr);
```

- detachedstate: This attribute allows us to avoid the need for threads to rejoin.

```
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
```

- The two possible flag values for pthread_attr_setdetachstate are PTHREAD_CREATE_JOINABLE and PTHREAD_CREATE_DETACHED.

6 pthread_attr_destroy()

```
pthread_attr_destroy(pthread_attr_t *attr)
```

1. To deinitialize a pthread_attr_t structure, we call pthread_attr_destroy.
2. If an implementation of pthread_attr_init allocated any dynamic memory for the attribute object, pthread_attr_destroy will free that memory
3. pthread_attr_destroy will initialize the attribute object with invalid values, so if it is used by mistake, pthread_create will return an error code.

Example 03

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *thread_function (void *arg);
char message [] = "Iam thread 1";
int main (){
        pthread_t tid;
        pthread_attr_t thread_attr;
        pthread_attr_init(&thread_attr);
        //comment following line in 2nd run
        pthread_attr_setdetachstate(&thread_attr,PTHREAD_CREATE_DETACHED);
        //uncomment following line in 2nd run
        //pthread_attr_setdetachstate(&thread_attr, PTHREAD_CREATE_JOINABLE);
        pthread_create (&tid,&thread_attr,thread_function,(void*) message);
        pthread_attr_destroy(&thread_attr);
        int joinret;
        joinret=pthread_join (tid,NULL);

        if(joinret==0)    //when join successfull
           printf("join was successfull:The main thread was waiting for thread 1\n");
        else
           printf("Join failed:The main thread is not waiting for thread 1\n");
        printf("The main thread finished, bye!\n");
        pthread_exit (NULL);
}

void * thread_function (void *arg){
        printf("thread_function is running. Argument was\" %s\"\n", (char *)arg);
        sleep (4);
        printf("thread awaked from sleep, and exiting now\n");
        pthread_exit (NULL);
}
```

7 Some Other Thread Attributes

```c
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
int pthread_attr_setscope(pthread_attr_t *attr, int scope);
int pthread_attr_getscope(const pthread_attr_t *attr, int *scope);
int pthread_attr_setstacksize(pthread_attr_t *attr, int scope);
int pthread_attr_getstacksize(const pthread_attr_t *attr, int *scope);
```

8 Thread Cancelation

1. Sometimes, we want one thread to be able to ask another thread to terminate, rather like sending it a signal. There is a way to do this with threads using pthread_cancel() system call.

2. One thread can request that another in the same process be canceled by calling the pthread_cancel function.

```
pthread_cancel(pthread_t tid)
```

3. In the default circumstances, pthread_cancel will cause the thread specified by **tid** to behave as if it had called **pthread_exit** with an argument of PTHREAD_CANCELED.

9 Thread Cancelation Options

A thread can change its cancelability state by calling pthread_setcancelstate.

```
pthread_setcancelstate(int state, int *oldstate);
```

The first parameter is either:

1. **PTHREAD_CANCEL_ENABLE** which allows it to receive cancel requests.
2. **PTHREAD_CANCEL_DISABLE** which causes them to be ignored. The oldstate pointer allows the previous state to be retrieved. If you are not interested, you can simply pass NULL.

Important Points

- A thread starts with a default cancelability state of PTHREAD_CANCEL_ENABLE.
- When the state is set to PTHREAD_CANCEL_DISABLE, a call to pthread_cancel will not kill the thread.
- The cancellation request remains pending for the thread.When the state is en- abled again, the thread will act on any pending cancellation requests at the next cancellation point.

10 Thread Cancelation Types

If cancel requests are accepted, there is a second level of control the thread can take, the cancel type, which is set with pthread_setcanceltype.

```
int pthread_setcanceltype(int type, int *oldtype);
```

The type can take one of two values:

1. **PTHREAD_CANCEL_ASYNCHRONOUS** which causes cancellation requests to be acted upon immediately.
2. **PTHREAD_CANCEL_DEFERRED** which makes cancellation requests wait un- til the thread executes one of these functions: pthread_join, pthread_cond_wait, pthread_cond_timedwait, sem_wait etc.

By default, threads start with the *cancellation state* as **PTHREAD_CANCEL_ENABLE** and the *cancellation type* as **PTHREAD_CANCEL_DEFERRED**.

Example 04

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
void *thread_function1(void *arg);
pthread_t tid1;
int main(){
        pthread_create(&tid1, NULL, thread_function1, NULL);
        sleep(2);
        printf("Main thread: Canceling thread 1...\n");
        pthread_cancel(tid1);
        printf("Main thread: exiting\n");
        pthread_exit(NULL);
}

void *thread_function1(void *arg){
        int i, oldtype, oldstate;
        // comment following line in 2nd run
        pthread_setcancelstate(PTHREAD_CANCEL_DISABLE,&oldstate);
        // uncomment following line in 2nd run
        //pthread_setcancelstate(PTHREAD_CANCEL_ENABLE,&oldtype);
        printf("Thread 1: iam running and my old cancel state was %d\n",oldstate);
        for(i=0; i<10; i++) {
           printf("Thread 1: iam still running (%d)...\n",i);
           sleep(2);
        }
        pthread_exit(0);
}
```

EXAMPLE 05

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
void *thread_function1(void *arg);
void *thread_function2(void *arg);
pthread_t tid1,tid2;
int main(){
        pthread_create(&tid1, NULL, thread_function1, NULL);
        pthread_create(&tid2, NULL, thread_function2, NULL);s
        leep(1);
        printf("Main thread: Canceling thread 1...\n");
        pthread_cancel(tid1);
        printf("Main thread: exiting\n");
        pthread_exit(NULL);
}

void *thread_function1(void *arg){
        int i, oldstate, oldtype;;
        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE,&oldstate);
        pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED,&oldtype);
        //pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS,&oldtype);
        //sleep(1);
        printf("Thread 1:my old cancel state was %d and old cancel type
                                                was %d\n",oldstate,oldtype);
        for(i=0; i<5; i++) {
```

```
        printf("Thread 1: iam running (%d)\n", i);
        if (i=1){
         printf("Thread 1: iam waiting for thread 2 to finish\n");
         pthread_join(tid2, NULL);}
        }
        // Following statement will never run!!!!!
        printf("Thread 1: Thread 1 exited, now me aswell\n");
        pthread_exit(0);
}

void * thread_function2(void *arg){
        int i;
        sleep(2);
        for(i = 0; i < 5; i++) {
            printf("Thread 2: iam running (%d)\n", i);
            printf("Thread 2: thread 1 received cancel signal but is
                                        waiting for me to finish\n");
        }
        printf("Thread 2: iam exiting\n");
        pthread_exit(0);
}
```