pthread Examples

Now that you know the pthread API…

- How do you create threads?
- How do you pass different values to them?
- How do you return values from threads?
- What are some common mistakes?

Passing Arguments to Threads

- **`pthread_create()`**
  - All arguments must be passed by reference and cast to **`(void *)`**
  - Only one argument to the thread start routine
  - For multiple arguments
    - Creating a structure that contains all of the arguments
    - Pass a pointer to that structure in **`pthread_create()`**

Passing Arguments to Threads

Where should these be declared?

- Passing an int:
  - ```
    int i = 42;
    pthread_create(..., my_func, (void *)&i);
    ```
- Passing a C-string:
  - ```
    char *str = "UIUC";
    pthread_create(..., my_func, (void *)str);
    ```
- Passing an array:
  - ```
    int arr[100];
    pthread_create(..., my_func, (void *)arr);
    ```



    ```

Passing Arguments to Threads

- Retrieving an int:
  - ```
    void *myfunc(void *vptr_value) {
        int value = *((int *)vptr_value);
    ```
- Retrieving a C-string:
  - ```
    void *myfunc(void *vptr_value) {
        char *str = (char *)vptr_value;
    ```
- Retrieving an array:
  - ```
    void *myfunc(void *vptr_value) {
        int *arr = (int *)vptr_value;
    ```



    ```

Passing Arguments to Threads

- Putting it all together:

```
void *myfunc(void *vptr_value) {
    int value = *((int *)vptr_value);
    printf("Thread value: %d", value);
    return NULL;
}

int main() {
    pthread_t tid;
    int i = 6;
    pthread_create(&tid, NULL, myfunc, &i);
    pthread_join(tid, NULL);
    return 0;
}
```

Will this be a problem?

Notifies the pthread library to use default attributes

Notifies the pthread library to ignore return value of myfunc

Thread Argument Passing

- How can you safely pass data to newly created threads, given their non-deterministic start-up and scheduling?
  - Make sure that all passed data is thread safe
    - i.e., it cannot be changed by other threads
- The following code fragment:
  - Demonstrates how to pass a simple integer to each thread
  - The calling thread uses a unique data structure for each thread
  - Each thread's argument remains intact throughout the program

Thread Argument Passing

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS    8

char *messages[NUM_THREADS];

void *PrintHello(void *threadid) {
   int *id_ptr, taskid;

   sleep(1);
   id_ptr = (int *) threadid;
   taskid = *id_ptr;
   printf("Thread %d: %s\n", taskid, messages[taskid]);
   pthread_exit(NULL);
}
```


Thread Argument Passing

```
int main(int argc, char *argv[]) {

    pthread_t threads[NUM_THREADS];
    int *taskids[NUM_THREADS];
    int rc, t;

    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";
```


Thread Argument Passing

```
    for(t=0;t<NUM_THREADS;t++) {
       taskids[t] = (int *) malloc(sizeof(int));
       *taskids[t] = t;
       printf("Creating thread %d\n", t);
       rc = pthread_create(&threads[t], NULL, PrintHello,
                           (void *) taskids[t]);
       if (rc) {
               printf("ERR; pthread_create() ret = %d\n", rc);
           exit(-1);
       }
    }
    pthread_exit(NULL);
}
```


Passing Complex Arguments

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS    8

char *messages[NUM_THREADS];

struct thread_data {
   int  thread_id;
   int  sum;
   char *message;
};

struct thread_data thread_data_array[NUM_THREADS];
```


Passing Complex Arguments

```
void *PrintHello(void *threadarg) {
    int taskid, sum;
    char *hello_msg;
    struct thread_data *my_data;

    sleep(1);
    my_data = (struct thread_data *) threadarg;
    taskid = my_data->thread_id;
    sum = my_data->sum;
    hello_msg = my_data->message;
    printf("Thread %d: %s  Sum=%d\n", taskid, hello_msg, sum);
    pthread_exit(NULL);
}
```


Passing Complex Arguments

```
int main(int argc, char *argv[]) {

    pthread_t threads[NUM_THREADS];
    int rc, t, sum;

    sum=0;
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";
```


Passing Complex Arguments

```
    for(t=0;t<NUM_THREADS;t++) {
       sum = sum + t;
       thread_data_array[t].thread_id = t;
       thread_data_array[t].sum = sum;
       thread_data_array[t].message = messages[t];
       printf("Creating thread %d\n", t);
       rc = pthread_create(&threads[t], NULL, PrintHello,
                           (void *) &thread_data_array[t]);
       if (rc) {
           printf("ERR; pthread_create() ret = %d\n", rc);
           exit(-1);
       }
    }
    pthread_exit(NULL);
}
```


Passing Complex Arguments

```
    for(t=0;t<NUM_THREADS;t++) {
        sum = sum + t;
        thread_data_array[t].thread_id = t;
        thread_data_array[t].sum = sum;
        rc = pthread_create(&threads[t], NULL, PrintHello,
                            (void *) taskids[t]);
        rc = pthread_create(&threads[t], NULL, PrintHello,
                            (void *) &thread_data_array[t]);
        if (rc) {
            printf("ERR; pthread_create() ret = %d\n", rc);
            exit(-1);
        }
    }
    pthread_exit(NULL);
}
```


Incorrect Argument Passing

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS    8

void *PrintHello(void *threadid)
{
    int *id_ptr, taskid;
    sleep(1);
    id_ptr = (int *) threadid;
    taskid = *id_ptr;
    printf("Hello from thread %d\n", taskid);
    pthread_exit(NULL);
}
```


Incorrect Argument Passing

```
int main(int argc, char *argv[]) {
    pthread_t threads[NUM_THREADS];
    int rc, t;

    for(t=0;t<NUM_THREADS;t++) {
        printf("Creating thread %d\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello,
                            (void *) &t);
        if (rc) {
            printf("ERR; pthread_create() ret = %d\n", rc);
            exit(-1);
        }
    }
    pthread_exit(NULL);
}
```

The loop that creates threads modifies the contents of the address passed as an argument, possibly before the created threads can access it.

What is the possible output?

More Pthreads Examples

```
#include <pthread.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid) {
        printf("\n%d: Hello World!\n", threadid);
        pthread_exit(NULL);
}

int main (int argc, char *argv[]) {
    pthread_t threads[NUM_THREADS];
    int rc, t;

    for(t=0;t < NUM_THREADS;t++) {
        printf("Creating thread %d\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        if (rc) {
           printf("ERROR; pthread_create() return code is %d\n", rc);
           exit(-1);
        }
    }
}
```

Will all threads get a chance to execute?

```
pthread_exit(NULL);
```


More Pthreads Examples

```
#include <pthread.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid) {
        printf("\n%d: Hello World!\n", threadid);
        pthread_exit(NULL);
}

int main (int argc, char *argv[]) {
    pthread_t threads[NUM_THREADS];
    int rc, t;

    for(t=0;t < NUM_THREADS;t++) {
        printf("Creating thread %d\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        if (rc) {
           printf("ERROR; pthread_create() return code is %d\n", rc);
           exit(-1);
        }
    }
    pthread_exit(NULL);
}
```

Will all threads get a chance to execute before the parent exits?

```
for(t=0;t < NUM_THREADS;t++) {
    pthread_join( thread[t], NULL);
    printf("Joining thread %d\n", t);
}
```


Returning data through `pthread_join()`

```
void *thread(void *vargp) {
    pthread_exit((void *)42);
}

int main() {
    int i;
    pthread_t tid;

    pthread_create(&tid, NULL, thread, NULL);
    pthread_join(tid, (void **)&i);
    printf("%d\n",i);
}
```

This is legal, but is it good programming practice?

Returning data through `pthread_join()`

```
void *thread(void *vargp) {
    int *value = (int *)malloc(sizeof(int));
    *value = 84;
    pthread_exit(value);
}

int main() {
    int i; pthread_t tid; void *vptr_return;

    pthread_create(&tid, NULL, thread, NULL);
    pthread_join(tid, &vptr_return);
    i = *((int *)vptr_return);
    free(vptr_return);
    printf("%d\n",i);
}
```


Incorrectly returning data through `pthread_join()`

What will happen here?

```
void *thread(void *vargp) {
    exit(42);
}

int main() {
    int i;
    pthread_t tid;

    pthread_create(&tid, NULL, thread, NULL);
    pthread_join(tid, (void **)&i);
    printf("%d\n",i);
}
```


Incorrectly returning data through `pthread_join()`

What will happen here?

```
void *thread(void *vargp) {
    pthread_detach(pthread_self());
    pthread_exit((void*)42);
}

int main() {
    int i = 0;
    pthread_t tid;

    pthread_create(&tid, NULL, thread, NULL);
    pthread_join(tid, (void**)&i);
    printf("%d\n",i);
}
```


Incorrectly returning data through `pthread_join()`

```
typedef struct _mystruct {
    double d;
    int i;
} mystruct;


void *myfunc(void *vptr) {
    mystruct my;
    my.d = 3.14159265;
    my.i = 42;
    pthread_exit((void*)&my);
}
```

```
int main() {
    pthread_t pid;
    mystruct my;
    void *vptr;

    pthread_create(&pid, NULL,
                   myfunc, NULL);
    pthread_join(pid, &vptr);

    my = *((mystruct *)vptr);
    free(vptr);

    printf("(%f, %d)", my.d,
                       my.i);

    return 0;
}
```


Returning data through `pthread_join()`

```
typedef struct _mystruct {
    double d;
    int i;
} mystruct;


void *myfunc(void *vptr) {
    mystruct *my = (mystruct *)
        malloc(sizeof(mystruct));
    my->d = 3.14159265;
    my->i = 42;
    pthread_exit((void*)my);
}
```

```
int main() {
    pthread_t pid;
    mystruct my;
    void *vptr;

    pthread_create(&pid, NULL,
                   myfunc, NULL);
    pthread_join(pid, &vptr);

    my = *((mystruct *)vptr);
    free(vptr);

    printf("(%f, %d)", my.d,
                       my.i);

    return 0;
}
```