Lab 5:

Task 1:

Create, compile and run a Pthreads "Hello world" program

```
root@VM:/home/JCogswell/Documents/Lab 5/pthreads# gcc -pthread hello.c -o hello
root@VM:/home/JCogswell/Documents/Lab 5/pthreads# ./hello
In main: creating thread 0
In main: creating thread 1
In main: creating thread 2
In main: creating thread 3
In main: creating thread 4
Hello World! It's me, thread #4!
Hello World! It's me, thread #3!
Hello World! It's me, thread #2!
Hello World! It's me, thread #1!
Hello World! It's me, thread #0!
root@VM:/home/JCogswell/Documents/Lab 5/pthreads# subl hello.c
root@VM:/home/JCogswell/Documents/Lab 5/pthreads#
```

/home/JCogswell/Documents/Lab 5/pthreads/hello.c - Sublime Text (UNREG

hello.c

```
/*****************************************************************************
* FILE: hello.c
* DESCRIPTION:
*   A "hello world" Pthreads program.  Demonstrates thread creation and
*   termination.
* AUTHOR: Blaise Barney
* LAST REVISED: 08/09/11
*****************************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
   long tid;
   tid = (long)threadid;
   printf("Hello World! It's me, thread #%ld!\n", tid);
   pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc;
   long t;
   for(t=0;t<NUM_THREADS;t++){
     printf("In main: creating thread %ld\n", t);
     rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
     if (rc){
       printf("ERROR; return code from pthread_create() is %d\n", rc);
       exit(-1);
       }
     }

   /* Last thing that main() should do */
   pthread_exit(NULL);
}
```

1. Thread Scheduling:

```
root@VM:/home/JCogswell/Documents/Lab 5/pthreads# gcc -pthread -o hello32 hello32.c -lm
hello32.c: In function 'Hello':
hello32.c:26:24: warning: implicit declaration of function 'sin' [-Wimplicit-function-declaration]
      result = result + sin(i) * tan(i);
                        ^
hello32.c:26:24: warning: incompatible implicit declaration of built-in function 'sin'
hello32.c:26:24: note: include '<math.h>' or provide a declaration of 'sin'
hello32.c:26:33: warning: implicit declaration of function 'tan' [-Wimplicit-function-declaration]
      result = result + sin(i) * tan(i);
                                 ^
hello32.c:26:33: warning: incompatible implicit declaration of built-in function 'tan'
hello32.c:26:33: note: include '<math.h>' or provide a declaration of 'tan'
hello32.c:28:11: warning: format '%ld' expects argument of type 'long int', but argument 2 has type 'void *' [-W
format=]
    printf("%ld: Hello World!\n", threadid);
           ^
root@VM:/home/JCogswell/Documents/Lab 5/pthreads# ./hello32
main(): Created 32 threads.
26: Hello World!
19: Hello World!
12: Hello World!
25: Hello World!
18: Hello World!
27: Hello World!
8: Hello World!
1: Hello World!
20: Hello World!
21: Hello World!
31: Hello World!
5: Hello World!
11: Hello World!
2: Hello World!
15: Hello World!
28: Hello World!
0: Hello World!
30: Hello World!
6: Hello World!
7: Hello World!
10: Hello World!
3: Hello World!
29: Hello World!
17: Hello World!
9: Hello World!
4: Hello World!
24: Hello World!
22: Hello World!
13: Hello World!
14: Hello World!
16: Hello World!
23: Hello World!
root@VM:/home/JCogswell/Documents/Lab 5/pthreads#
```

/home/JCogswell/Documents/Lab 5/pthreads/hello32.c - Sublime Text (UNREGISTERED)

hello.c | hello32.c

```
/*****************************************************************************
* FILE: hello32.c
* DESCRIPTION:
*   A "hello world" Pthreads program which creates a large number of
*   threads per process.  A sleep() call is used to insure that all
*   threads are in existence at the same time.  Each Hello thread does some
*   work to demonstrate how the OS scheduler behavior affects thread
*   completion order.
* AUTHOR: Blaise Barney
* LAST REVISED: 01/29/09
*****************************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#define NTHREADS    32

void *Hello(void *threadid)
{
   int i;
   double result=0.0;
   sleep(3);
   for (i=0; i<10000; i++) {
     result = result + sin(i) * tan(i);
     }
   printf("%ld: Hello World!\n", threadid);
   pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
pthread_t threads[NTHREADS];
int rc;
long t;
for(t=0;t<NTHREADS;t++){
  rc = pthread_create(&threads[t], NULL, Hello, (void *)t);
  if (rc){
    printf("ERROR: return code from pthread_create() is %d\n", rc);
    printf("Code %d= %s\n",rc,strerror(rc));
    exit(-1);
    }
  }
printf("main(): Created %ld threads.\n", t);
pthread_exit(NULL);
}
```

Review the example code hello32.c. Note that it will create 32 threads. A sleep(); statement has been introduced to help insure that all threads will be in existence at the same time. Also, each thread performs actual work to demonstrate how the OS scheduler behavior determines the order of thread completion. Compile and run the program. Notice the order in which thread output is displayed. Is it ever in the same order? How is this explained?

The threads are never in the same order because the OS scheduler controls when threads will be executed.

2. Argument Passing

Review the hello_arg1.c and hello_arg2.c example codes. Notice how the single argument is passed and how to pass multiple arguments through a structure. Compile and run both programs, and observe output. Now review, compile and run the bug3.c program. What's wrong? How would you fix it?

```
Terminal
[03/29/19]JCogswell@VM:~/.../pthreads$ gcc -pthread hello_arg1.c -o hello_arg1
hello_arg1.c: In function ‘PrintHello’:
hello_arg1.c:20:4: warning: implicit declaration of function ‘sleep’ [-Wimplicit-function-d
    sleep(1);
    ^
hello_arg1.c:22:11: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has
rmat=]
    printf("Thread %d: %s\n", taskid, messages[taskid]
          ^
[03/29/19]JCogswell@VM:~/.../pthreads$ ./hello_arg1
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Creating thread 4
Creating thread 5
Creating thread 6
Creating thread 7
Thread 4: German: Guten Tag, Welt!
Thread 3: Klingon: Nuq neH!
Thread 2: Spanish: Hola al mundo
Thread 6: Japan: Sekai e konnichiwa!
Thread 7: Latin: Orbis, te saluto!
Thread 1: French: Bonjour, le monde!
Thread 5: Russian: Zdravstvuyte, mir!
Thread 0: English: Hello World!
[03/29/19]JCogswell@VM:~/.../pthreads$
```

```c
/******************************************************************************
* FILE: hello_arg1.c
* DESCRIPTION:
*   A "hello world" Pthreads program which demonstrates one safe way
*   to pass arguments to threads during thread creation.
* AUTHOR: Blaise Barney
* LAST REVISED: 08/04/15
******************************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 8

char *messages[NUM_THREADS];

void *PrintHello(void *threadid)
{
   long taskid;

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

int main(int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
long 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: Zdravstvuyte, mir!";
messages[6] = "Japan: Sekai e konnichiwa!";
messages[7] = "Latin: Orbis, te saluto!";

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

pthread_exit(NULL);
}
```

```
[03/29/19]JCogswell@VM:~/.../pthreads$ gcc -pthread hello_arg2.c -o hello_arg2
hello_arg2.c: In function ‘PrintHello’:
hello_arg2.c:32:4: warning: implicit declaration of function ‘sleep’ [-Wimplicit-function-declaration]
    sleep(1);
    ^
[03/29/19]JCogswell@VM:~/.../pthreads$ ./hello_arg2
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Creating thread 4
Creating thread 5
Creating thread 6
Creating thread 7
Thread 5: Russian: Zdravstvytye, mir!  Sum=15
Thread 6: Japan: Sekai e konnichiwa!  Sum=21
Thread 4: German: Guten Tag, Welt!  Sum=10
Thread 3: Klingon: Nuq neH!  Sum=6
Thread 7: Latin: Orbis, te saluto!  Sum=28
Thread 2: Spanish: Hola al mundo  Sum=3
Thread 1: French: Bonjour, le monde!  Sum=1
Thread 0: English: Hello World!  Sum=0
```

```c
/*****************************************************************************
* FILE: hello_arg2.c
* DESCRIPTION:
*   A "hello world" Pthreads program which demonstrates another safe way
*   to pass arguments to threads during thread creation.  In this case,
*   a structure is used to pass multiple arguments.
* AUTHOR: Blaise Barney
* LAST REVISED: 01/29/09
*****************************************************************************/
#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];

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);
```

```
bug3.c: In function ‘PrintHello’:
bug3.c:17:4: warning: implicit declaration of function ‘sleep’ [-Wimplicit-function-declaration]
    sleep(1);
    ^
[03/29/19]JCogswell@VM:~/.../pthreads$ ./bug3
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Creating thread 4
Creating thread 5
Creating thread 6
Creating thread 7
Hello from thread 8
Hello from thread 8
Hello from thread 8
Hello from thread 8
Hello from thread 8
Hello from thread 8
Hello from thread 8
Hello from thread 8
[03/29/19]JCogswell@VM:~/.../pthreads$
```

```c
/*****************************************************************************
* FILE: bug3.c
* DESCRIPTION:
*   This "hello world" Pthreads program demonstrates an unsafe (incor
*   way to pass thread arguments at thread creation. Compare with hel
* AUTHOR: Blaise Barney
* LAST REVISED: 07/16/14
*****************************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     8

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

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

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

pthread_exit(NULL);
}
```

The bug3.c program was passing the address of t instead of the value of t. You would fix this by looking at the line that says “rc = pthread_create(&threads,” etc. and at the end where it says “(void *) &t);” you would change the “&t” to something similar to what is in hello_arg1.c which is “taskids[t]);”

3. Thread Exiting

Review, compile (for gcc include the -lm flag) and run the bug5.c program. What happens? Why? How would you fix it?

Running bug5.c before → making necessary → changes to fix it.

```
[03/31/19]JCogswell@VM:~/.../pthreads$ gcc -pthread bug5.c -o
bug5 -lm
bug5.c: In function ‘PrintHello’:
bug5.c:19:11: warning: format ‘%ld’ expects argument of type ‘
long int’, but argument 2 has type ‘void *’ [-Wformat=]
    printf("thread=%ld: starting...\n", threadid);
           ^
bug5.c:22:11: warning: format ‘%ld’ expects argument of type ‘
long int’, but argument 2 has type ‘void *’ [-Wformat=]
    printf("thread=%ld result=%e. Done.\n",threadid,myresult);
           ^
[03/31/19]JCogswell@VM:~/.../pthreads$ ./bug5
Main: creating thread 0
Main: creating thread 1
Main: creating thread 2
Main: creating thread 3
Main: creating thread 4
Main: Done.
```

Original bug5.c program → → →

```c
/******************************************************************
* FILE: bug5.c
* DESCRIPTION:
*   A simple pthreads program that dies before the threads can do t
*   work. Figure out why.
* AUTHOR: 07/06/05 Blaise Barney
* LAST REVISED: 07/11/12
******************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
   int i;
   double myresult=0.0;
   printf("thread=%ld: starting...\n", threadid);
   for (i=0; i<1000000; i++)
     myresult += sin(i) * tan(i);
   printf("thread=%ld result=%e. Done.\n",threadid,myresult);
   pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
int rc;
long t;
for(t=0;t<NUM_THREADS;t++){
  printf("Main: creating thread %ld\n", t);
  rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
  if (rc){
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
    }
  }
printf("Main: Done.\n");
}
```

What is happening here is that the bug5.c program is creating threads that are dying before they're able to be executed. The reason for this is that the *main* portion of the program will finish and disappear along with all of its threads that it has created. Adding *pthread_exit(NULL);* solves this issue.

Running bug5.c *after* adding → the *pthreads_exit(NULL);* to → the end of the program. →

```
[03/31/19]JCogswell@VM:~/.../pthreads$ gcc -pthread bug5.c -o
bug5 -lm
bug5.c: In function 'PrintHello':
bug5.c:19:11: warning: format '%ld' expects argument of type '
long int', but argument 2 has type 'void *' [-Wformat=]
    printf("thread=%ld: starting...\n", threadid);
           ^
bug5.c:22:11: warning: format '%ld' expects argument of type '
long int', but argument 2 has type 'void *' [-Wformat=]
    printf("thread=%ld result=%e. Done.\n",threadid,myresult);
           ^
[03/31/19]JCogswell@VM:~/.../pthreads$ ./bug5
Main: creating thread 0
Main: creating thread 1
Main: creating thread 2
Main: creating thread 3
Main: creating thread 4
Main: Done.
thread=4: starting...
thread=3: starting...
thread=2: starting...
thread=1: starting...
thread=0: starting...
thread=4 result=-3.153838e+06. Done.
thread=2 result=-3.153838e+06. Done.
thread=3 result=-3.153838e+06. Done.
thread=0 result=-3.153838e+06. Done.
thread=1 result=-3.153838e+06. Done.
[03/31/19]JCogswell@VM:~/.../pthreads$
```

New bug5.c program with the → added *pthreads_exit(NULL);*. →

```c
/******************************************************************
* FILE: bug5.c
* DESCRIPTION:
*   A simple pthreads program that dies before the threads can do t
*   work. Figure out why.
* AUTHOR: 07/06/05 Blaise Barney
* LAST REVISED: 07/11/12
******************************************************************
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
   int i;
   double myresult=0.0;
   printf("thread=%ld: starting...\n", threadid);
   for (i=0; i<1000000; i++)
     myresult += sin(i) * tan(i);
   printf("thread=%ld result=%e. Done.\n",threadid,myresult);
   pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
int rc;
long t;
for(t=0;t<NUM_THREADS;t++){
  printf("Main: creating thread %ld\n", t);
  rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
  if (rc){
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
    }
  }
printf("Main: Done.\n");
pthread_exit(NULL);
}
```

4. Thread Joining

Review, compile (for gcc include the -lm flag) and run the join.c program. Modify the program so that threads send back a different return code - you pick. Compile and run. Did it work? For comparison, review, compile (for gcc include the -lm flag) and run the detached.c example code. Observe the behavior and note there is

Executing the join.c → program.

```
[03/31/19]JCogswell@VM:~/.../pthreads$ ./join
Main: creating thread 0
Main: creating thread 1
Main: creating thread 2
Main: creating thread 3
Thread 3 starting...
Thread 2 starting...
Thread 1 starting...
Thread 0 starting...
Thread 2 done. Result = -3.153838e+06
Thread 1 done. Result = -3.153838e+06
Thread 3 done. Result = -3.153838e+06
Thread 0 done. Result = -3.153838e+06
Main: completed join with thread 0 having a status of 0
Main: completed join with thread 1 having a status of 1
Main: completed join with thread 2 having a status of 2
Main: completed join with thread 3 having a status of 3
Main: program completed. Exiting.
```

join.c ⬇

```c
/******************************************************************************
* FILE: join.c
* DESCRIPTION:
*   This example demonstrates how to "wait" for thread completions
*   the Pthread join routine.  Threads are explicitly created in a
*   state for portability reasons. Use of the pthread_exit status a
*   also shown. Compare to detached.c
* AUTHOR: 8/98 Blaise Barney
* LAST REVISED:  01/30/09
******************************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 4

void *BusyWork(void *t)
{
   int i;
   long tid;
   double result=0.0;
   tid = (long)t;
   printf("Thread %ld starting...\n",tid);
   for (i=0; i<1000000; i++)
   {
      result = result + sin(i) * tan(i);
   }
   printf("Thread %ld done. Result = %e\n",tid, result);
   pthread_exit((void*) t);
}

int main (int argc, char *argv[])
{
   pthread_t thread[NUM_THREADS];
   pthread_attr_t attr;
   int rc;
   long t;
   void *status;

   /* Initialize and set thread detached attribute */
   pthread_attr_init(&attr);
   pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

   for(t=0; t<NUM_THREADS; t++) {
      printf("Main: creating thread %ld\n", t);
      rc = pthread_create(&thread[t], &attr, BusyWork, (void *)t);
      if (rc) {
         printf("ERROR; return code from pthread_create() is %d\n",
         exit(-1);
         }
      }

   /* Free attribute and wait for the other threads */
   pthread_attr_destroy(&attr);
   for(t=0; t<NUM_THREADS; t++) {
      rc = pthread_join(thread[t], &status);
      if (rc) {
         printf("ERROR; return code from pthread_join() is %d\n", r
         exit(-1);
         }
      printf("Main: completed join with thread %ld having a status
      }

printf("Main: program completed. Exiting.\n");
pthread_exit(NULL);
}
```

Compiling and executing the updated join.c file named newjoin. →

```
[03/31/19]JCogswell@VM:~/.../pthreads$ gcc -pthread -o newjoin
 newjoin.c -lm
[03/31/19]JCogswell@VM:~/.../pthreads$ ./newjoin
Main: creating thread 0
Main: creating thread 1
Main: creating thread 2
Main: creating thread 3
Thread 3 starting...
Thread 2 starting...
Thread 1 starting...
Thread 0 starting...
Thread 0 done. Result = -1.704026e+06
Main: completed join with thread 0 having a status of 0
Thread 3 done. Result = -1.704026e+06
Thread 2 done. Result = -1.704026e+06
Thread 1 done. Result = -1.704026e+06
Main: completed join with thread 1 having a status of 1
Main: completed join with thread 2 having a status of 2
Main: completed join with thread 3 having a status of 3
Main: program completed. Exiting.
[03/31/19]JCogswell@VM:~/.../pthreads$ ./newjoin
Main: creating thread 0
Main: creating thread 1
Main: creating thread 2
Main: creating thread 3
Thread 3 starting...
Thread 2 starting...
Thread 1 starting...
Thread 0 starting...
Thread 0 done. Result = -1.704026e+06
Main: completed join with thread 0 having a status of 0
Thread 1 done. Result = -1.704026e+06
Main: completed join with thread 1 having a status of 1
Thread 3 done. Result = -1.704026e+06
Thread 2 done. Result = -1.704026e+06
Main: completed join with thread 2 having a status of 2
Main: completed join with thread 3 having a status of 3
Main: program completed. Exiting.
[03/31/19]JCogswell@VM:~/.../pthreads$
```

The updated code was only the addition of "+1" in the sin function. ⬇

```
#include <math.h>
#define NUM_THREADS  4

void *BusyWork(void *t)
{
   int i;
   long tid;
   double result=0.0;
   tid = (long)t;
   printf("Thread %ld starting...\n",tid);
   for (i=0; i<1000000; i++)
   {
      result = result + sin(i+1) * tan(i);
   }
   printf("Thread %ld done. Result = %e\n",tid, result);
   pthread_exit((void*) t);
}
```

Compiling and executing the detached.c file. →

```
[03/31/19]JCogswell@VM:~/.../pthreads$ gcc -pthread -o detache
d detached.c -lm
[03/31/19]JCogswell@VM:~/.../pthreads$ ./detached
Main: creating thread 0
Main: creating thread 1
Main: creating thread 2
Main: creating thread 3
Main: program completed. Exiting.
Thread 3 starting...
Thread 2 starting...
Thread 1 starting...
Thread 0 starting...
Thread 2 done. Result = -3.153838e+06
Thread 3 done. Result = -3.153838e+06
Thread 0 done. Result = -3.153838e+06
Thread 1 done. Result = -3.153838e+06
[03/31/19]JCogswell@VM:~/.../pthreads$
[03/31/19]JCogswell@VM:~/.../pthreads$
[03/31/19]JCogswell@VM:~/.../pthreads$ ./detached
Main: creating thread 0
Main: creating thread 1
Main: creating thread 2
Main: creating thread 3
Main: program completed. Exiting.
Thread 3 starting...
Thread 2 starting...
Thread 1 starting...
Thread 0 starting...
Thread 3 done. Result = -3.153838e+06
Thread 0 done. Result = -3.153838e+06
Thread 2 done. Result = -3.153838e+06
Thread 1 done. Result = -3.153838e+06
[03/31/19]JCogswell@VM:~/.../pthreads$ ./detached
Main: creating thread 0
Main: creating thread 1
Main: creating thread 2
Main: creating thread 3
Main: program completed. Exiting.
Thread 3 starting...
Thread 2 starting...
Thread 1 starting...
Thread 0 starting...
Thread 2 done. Result = -3.153838e+06
Thread 0 done. Result = -3.153838e+06
Thread 3 done. Result = -3.153838e+06
Thread 1 done. Result = -3.153838e+06
[03/31/19]JCogswell@VM:~/.../pthreads$
```

detached.c ⬇

```c
/******************************************************************************
* FILE: detached.c
* DESCRIPTION:
*   This example demonstrates how to explicitly create a thread in a
*   detached state. This might be done to conserve some system resources
*   if the thread never needs to join later. Compare with the join.c program
*   where the threads are created joinable.
* AUTHOR: 01/30/08 Blaise Barney
* LAST REVISED: 01/29/09
******************************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NUM_THREADS 4

void *BusyWork(void *t)
{
   long i, tid;
   double result=0.0;
   tid = (long)t;
   printf("Thread %ld starting...\n",tid);
   for (i=0; i<1000000; i++) {
     result = result + sin(i) * tan(i);
     }
   printf("Thread %ld done. Result = %e\n",tid, result);
}

int main(int argc, char *argv[])
{
pthread_t thread[NUM_THREADS];
pthread_attr_t attr;
int rc;
long t;

/* Initialize and set thread detached attribute */
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

for(t=0;t<NUM_THREADS;t++) {
   printf("Main: creating thread %ld\n", t);
   rc = pthread_create(&thread[t], &attr, BusyWork, (void *)t);
   if (rc) {
     printf("ERROR; return code from pthread_create() is %d\n", rc);
     exit(-1);
     }
  }

/* We're done with the attribute object, so we can destroy it */
pthread_attr_destroy(&attr);

/* The main thread is done, so we need to call pthread_exit explicitly to
*  permit the working threads to continue even after main completes.
*/
printf("Main: program completed. Exiting.\n");
pthread_exit(NULL);
}
```

When adding the +1 to the sin function and executing the program, I was able to change the number that the program returned, although it doesn't seem like that number does anything significant. If there was a different number I was supposed to change, I couldn't figure out which one after hours of searching. What I did find, though, is that the code in newjoin.c printed out in different orders than they were executed. The reason is because when threads are joined, they will wait until the child threads have completed before completing themselves.

5. Stack Management

Review, compile and run the bug2.c program. What happens? Why? How would you fix it?

Compiling and executing →

bug2.c →

```
[03/31/19]JCogswell@VM:~/.../pthreads$ gcc -pthread -o bug2 bu
g2.c -lm
bug2.c: In function 'Hello':
bug2.c:22:4: warning: implicit declaration of function 'sleep'
 [-Wimplicit-function-declaration]
    sleep(3);
    ^
bug2.c: In function 'main':
bug2.c:40:8: warning: format '%li' expects argument of type 'l
ong int', but argument 2 has type 'size_t {aka unsigned int}'
[-Wformat=]
 printf("Thread stack size = %li bytes (hint, hint)\n",stacksi
        ^
[03/31/19]JCogswell@VM:~/.../pthreads$ ./bug2
Thread stack size = 8388608 bytes (hint, hint)
Created 8 threads.
5: Hello World!   499999.000000
7: Hello World!   499999.000000
6: Hello World!   499999.000000
4: Hello World!   499999.000000
1: Hello World!   499999.000000
3: Hello World!   499999.000000
0: Hello World!   499999.000000
2: Hello World!   499999.000000
```

bug2.c →

```c
/* FILE: bug2.c
* DESCRIPTION:
*   A "hello world" Pthreads program that dumps core.  Figure out w
*   then fix it - or else see the solution bug2fix.c.
* AUTHOR: 9/98 Blaise Barney
* LAST REVISED: 01/29/09
******************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NTHREADS     8
#define ARRAY_SIZE      500000

void *Hello(void *threadid)
{
   double A[ARRAY_SIZE];
   int i;
   long tid;

   tid = (long)threadid;
   sleep(3);
   for (i=0; i<ARRAY_SIZE; i++)
   {
      A[i] = i * 1.0;
   }
   printf("%ld: Hello World!   %f\n", tid, A[ARRAY_SIZE-1]);
   pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
pthread_t threads[NTHREADS];
size_t stacksize;
pthread_attr_t attr;
int rc;
long t;
pthread_attr_init(&attr);
pthread_attr_getstacksize (&attr, &stacksize);
printf("Thread stack size = %li bytes (hint, hint)\n",stacksize);
for(t=0;t<NTHREADS;t++){
  rc = pthread_create(&threads[t], NULL, Hello, (void *)t);
  if (rc){
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
    }
  }
printf("Created %ld threads.\n", t);
pthread_exit(NULL);
}
```

What happens in bug2.c is that the ARRAY_SIZE is set to 500000 while the program is creating larger thread stack sizes than it can handle, resulting in a seg fault and a core dump. To fix this, you can set an adequate stack size to accommodate the overcome this issue. This is shown in bug2fix.c.

Compiling and executing → bug2fix.c

```
[03/31/19]JCogswell@VM:~/.../pthreads$ gcc -pthread -o bug2fix
 bug2fix.c -lm
bug2fix.c: In function ‘Hello’:
bug2fix.c:30:4: warning: implicit declaration of function ‘sle
ep’ [-Wimplicit-function-declaration]
    sleep(3);
    ^
bug2fix.c:37:11: warning: format ‘%li’ expects argument of typ
e ‘long int’, but argument 3 has type ‘size_t {aka unsigned in
t}’ [-Wformat=]
    printf("%ld: Thread stack size = %li bytes \n", tid, mysta
           ^
bug2fix.c: In function ‘main’:
bug2fix.c:51:8: warning: format ‘%li’ expects argument of type
 ‘long int’, but argument 2 has type ‘size_t {aka unsigned int
}’ [-Wformat=]
 printf("Thread stack size = %li bytes (hint, hint)\n",stacksi
        ^
[03/31/19]JCogswell@VM:~/.../pthreads$ ./bug2fix
Thread stack size = 5000000 bytes (hint, hint)
Created 8 threads.
6: Hello World!   499999.000000
6: Thread stack size = 5000000 bytes
1: Hello World!   499999.000000
1: Thread stack size = 5000000 bytes
2: Hello World!   499999.000000
2: Thread stack size = 5000000 bytes
0: Hello World!   499999.000000
0: Thread stack size = 5000000 bytes
7: Hello World!   499999.000000
7: Thread stack size = 5000000 bytes
5: Hello World!   499999.000000
5: Thread stack size = 5000000 bytes
4: Hello World!   499999.000000
4: Thread stack size = 5000000 bytes
3: Hello World!   499999.000000
3: Thread stack size = 5000000 bytes
[03/31/19]JCogswell@VM:~/.../pthreads$
```

bug2fix.c

```
/*
* FILE: bug2fix.c
* DESCRIPTION:
*   This is just one way to fix the "hello world" Pthreads program
*   core.  Things to note:
*     - attr variable and its scoping
*     - use of the pthread_attr_setstacksize routine
*     - initialization of the attr variable with pthread_attr_init
*     - passing the attr variable to pthread_create
* AUTHOR: 9/98 Blaise Barney
* LAST REVISED: 01/29/09
******************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NTHREADS  8
#define ARRAY_SIZE      500000
#define MEGEXTRA        1000000

pthread_attr_t attr;

void *Hello(void *threadid)
{
   double A[ARRAY_SIZE];
   int i;
   long tid;
   size_t mystacksize;

   tid = (long)threadid;
   sleep(3);
   for (i=0; i<ARRAY_SIZE; i++)
   {
      A[i] = i * 1.0;
   }
   printf("%ld: Hello World!   %f\n", tid, A[ARRAY_SIZE-1]);
   pthread_attr_getstacksize (&attr, &mystacksize);
   printf("%ld: Thread stack size = %li bytes \n", tid, mystacksize
   pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
pthread_t threads[NTHREADS];
size_t stacksize;
int rc;
long t;
pthread_attr_init(&attr);
stacksize = ARRAY_SIZE*sizeof(double) + MEGEXTRA;
pthread_attr_setstacksize (&attr, stacksize);
pthread_attr_getstacksize (&attr, &stacksize);
printf("Thread stack size = %li bytes (hint, hint)\n",stacksize);
for(t=0;t<NTHREADS;t++){
  rc = pthread_create(&threads[t], &attr, Hello, (void *)t);
    if (rc){
      printf("ERROR; return code from pthread_create() is %d\n", rc
      exit(-1);
      }
  }
printf("Created %ld threads.\n", t);
pthread_exit(NULL);
}
```

Task 2: In this task, I used the banker.c program and input given values to show that the process is in a safe state. The program confirms that it is in a safe state. Then, I alter the values to put the program in an unsafe state.

```
[03/31/19]JCogswell@VM:~/.../pthreads$ gcc -o banker banker.c
[03/31/19]JCogswell@VM:~/.../pthreads$ ./banker

Enter the number of resources: 4

Enter the number of processes: 5

Enter Claim Vector: 8 5 9 7

Enter Allocated Resource Table: 2 0 1 1 0 1 2 1 4 0 0 3 0 2 1 0 1 0 3 0

Enter Maximum Claim table: 3 2 1 4 0 2 5 2 5 1 0 5 1 5 3 0 3 0 3 3

The Claim Vector is: 8 5 9 7
The Allocated Resource Table:
        2       0       1       1
        0       1       2       1
        4       0       0       3
        0       2       1       0
        1       0       3       0

The Maximum Claim Table:
        3       2       1       4
        0       2       5       2
        5       1       0       5
        1       5       3       0
        3       0       3       3

Allocated resources: 7 3 7 5
Available resources: 1 2 2 2

Process3 is executing.

The process is in safe state.
Available vector: 5 2 2 5
Process1 is executing.

The process is in safe state.
Available vector: 7 2 3 6
Process2 is executing.

The process is in safe state.
Available vector: 7 3 5 7
Process4 is executing.

The process is in safe state.
Available vector: 7 5 6 7
Process5 is executing.

The process is in safe state.
```

After altering the values, the processes are in an unsafe state as noted by the program. I changed the numbers in the allocated resource table (which is calculated by adding up all of the allocated resources from all 4 of the processes) so that there would be nothing available after completion, which would result in a potential deadlock. If I leave zero available resources, then I put the processes in an unsafe state which makes them a very likely candidate to be deadlocked, although not guaranteed to become deadlocked. If I leave less than zero available resources, then the processes are deadlocked. Another way to put them into an unsafe state would be to exhaust resources through other processes so that the resource would no longer be able to at least fulfill the maximum claim of one more process.