Code:

```
#include <stdio.h>,

#include <stdlib.h>

#include <pthread.h>

//  Example:  https://thispointer.com/posix-detached-vs-joinable-threads-pthread_join-pthread_detach-examples/


void* function( void* arg)

{

sleep((int) &arg[0]);

printf( "This is thread %d and waits time %d\n", pthread_self(), (int)&arg[0] );

        return( pthread_self());

}

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


pthread_t thread1;

pthread_t thread2;

void* id1;

void* id2;


// Set thread attributes

pthread_attr_t attr;

pthread_attr_init( &attr );

pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE );


// Create Threads

int retCrea1 = pthread_create(&thread1, &attr, &function, 6);

int retCrea2 = pthread_create(&thread2, &attr, &function, 3);


if ( retCrea1 != 0 ){
```

```
 printf ("pthread_create1: %s\n", strerror(retCrea1));
 exit(-1);
}
if ( retCrea2 != 0 ){
 printf ("pthread_create2: %s\n", strerror(retCrea2));
 exit(-1);
}
//Join threads to this one (main)
int retJoin1 = pthread_join(thread1, &id1);
int retJoin2 = pthread_join(thread2, &id2);

if ( retJoin1 != 0 ){
 printf ("pthread_join1: %s\n", strerror(retJoin1));
 exit(-1);
}
if ( retJoin2 != 0 ){
 printf ("pthread_join2: %s\n", strerror(retJoin2));
 exit(-1);
}
if (thread1 != id1) {
printf("THREAD ID: %d is not the same as: %d !\n", thread1, id1);
}
if (thread2 != id2) {
printf("THREAD ID: %d is not the same as: %d !\n", thread2, id2);
}
printf( "ThreadId 1: %d, ThreadId 2: %d\n", thread1, thread2);
return EXIT_SUCCESS;
}
```

Kommentare und Anmerkungen zum Code:

1. Beim Erzeugen der Threads mit pthread_create() ist es insbesondere wichtig, den Thread auf JOINABLE (statt DETEACHED) zu setzen. Bei DETEACHED arbeitet der Thread unabhängig von anderen. Bei JOINABLE ist es möglich, dass Threads sich an andere binden und auf diese zu wartet. Hierbei wird aber lediglich ein Flag in den Attributen der Threads gesetzt, sodass ein JOIN prinzipiell möglich wird, er wird aber noch nicht gejoint! Dies ist insbesondere wichtig, da dann Eigenschaften und Attribute des Thread nach dem er sich beendet hat, noch weiter im Speicher bleiben, falls andere Threads auf diesen warten. Wenn diese wartenden Threads bemerken, dass ersterer sich bereits beendet hat, können diese weiterarbeiten. Würde man diese Flag nicht setzen, würden die Threads ewig auf einen bereits beendeten Thread warten und das Programm liefe in einen Deadlock.
2. Des Weiteren muss den Threads nach deren Erstellung noch genau zugewiesen werden, mit welchen Threads sie sich joinen sollen. Dies passiert mittels der Befehls pthread_join(). Dabei wird der aktuelle Thread mit dem in der Klammer befindlichen verbunden. In unserem Beispiel verbindet sich der Main-Thread mit den beiden Unterthreads eins und zwei und beendet sich damit erst, wenn beide ebenfalls beendet sind.
3. Die Funktion der beiden neuen Threads besteht eigentlich lediglich daraus zu warten mittels dem Befehl sleep() und danach eine Ausgabe auf der Kommandozeile zu machen um nachverfolgen zu können, dass diese beiden Threads vor dem Main Thread beendet werden. Außerdem ist gewehrleistet, dass diese beiden Threads ihre jeweilige Task-ID als Success-Rückgabe wert liefern um dem Mainprogramm einen Erfolg des Ablauft zurückzugeben. Dieser Erfolg wird im Mainprogramm verifiziert.
4. Ein alternativ zum Betrachten des Returnwerts könnte man auch pthread_exit() verwenden. Es dient dazu dem aufrufenden Thread einen Rückgabewert in einem Pointer auf diesen mitzuteilen. In vorliegendem Code wurde diese über den normalen Rückgabewert der Funktion, die in den beiden Unterthreads aufgerufen wird umgesetzt.