```
#include "_threadsCore.h"
#include "osDefs.h"

//global variables
uint32_t* endOfStack_ptr = NULL;
int numThreads = 0;
threadStruct threadCollection[MAX_THREADS];

//obtain the initial location of MSP by looking it up in the vector table
uint32_t* getMSPInitialLocation (void){
  uint32_t* MSP_ptr = (uint32_t*) 0x0; //define a pointer to a pointer that points to initial MSP
  printf("MSP: %08x\n", *MSP_ptr);
  if(endOfStack_ptr == NULL){ //only allow endOfStack_ptr to be set to initial MSP location once
    endOfStack_ptr = (uint32_t*) *MSP_ptr;
  }

  return (uint32_t*) *MSP_ptr; //dereference so that it returns just the pointer to initial MSP
}

//return address of new a PSP with offset of "offset" bytes from MSP
uint32_t* getNewThreadStack (uint32_t offset){
  //check if we are exceeding the max stack size
  if (MAX_STACK < offset*(numThreads+1)){
    printf("ERROR: Offset too large");
    return NULL;
      //make sure to look for a NULL return in future functions to check if getNewThreadStack failed or not
  }

  //calculate address of PSP from MSP
  uint32_t* MSP_ptr = getMSPInitialLocation();
  uint32_t PSP_adr = (uint32_t) MSP_ptr - offset;

  //check if PSP address is a number divisible by 8
  if(PSP_adr%8 != 0){
    PSP_adr = PSP_adr+sizeof(uint32_t); //add 4 to address to ensure valid address for the stack
  }

  //check if overwriting a previous stack
  if(PSP_adr > (uint32_t) endOfStack_ptr-(STACK_SIZE)){
    printf("ERROR: Overwriting old data");
    return NULL;
  }

  //assign PSP_ptr to point to PSP_adr
  uint32_t* PSP_ptr = (uint32_t*) PSP_adr;
  printf("PSP: %08x\n", (uint32_t) PSP_ptr);
  endOfStack_ptr = PSP_ptr;

  return PSP_ptr;
}


//LAB 1: set the value of PSP to threadStack and ensure that the microcontroller is using that value by changing the CONTROL register
/*void setThreadingWithPSP (uint32_t* threadStack){
  __set_PSP((uint32_t) threadStack);
  __set_CONTROL(1<<1);
}*/


//Initializes the thread stack and its initial context in memory
int osThreadNew(void (*fun_ptr)(void)){
  ++numThreads;
  int stackID = numThreads-1;

  //generate and store TSP
  threadCollection[stackID].TSP = getNewThreadStack(STACK_SIZE + numThreads*STACK_SIZE); //MSP stack + n*thread stacks

  //if getnewThreadStack encounters an error creating the thread pointer, TSP generated will be a NULL pointer
  if(threadCollection[stackID].TSP == NULL){
    --numThreads;
    return -1; //osThreadNew failed
  }

  //store the thread's function pointer
  threadCollection[stackID].fun_ptr = fun_ptr;

  //set the values for what the "running" thread will populate the registers with
  *(--threadCollection[stackID].TSP) = 1<<24; //xPSR
  *(--threadCollection[stackID].TSP) = (uint32_t) fun_ptr; //PC (program counter)

    //dummy values (need to be nonzero)
    *(--threadCollection[stackID].TSP) = 0xE; //LR
    *(--threadCollection[stackID].TSP) = 0xC; //R12
    *(--threadCollection[stackID].TSP) = 0x3; //R3
    *(--threadCollection[stackID].TSP) = 0x2; //R2
    *(--threadCollection[stackID].TSP) = 0x1; //R1
    *(--threadCollection[stackID].TSP) = 0x0; //R0

    //dummy values (for testing purposes)
    *(--threadCollection[stackID].TSP) = 0xB; //R11
    *(--threadCollection[stackID].TSP) = 0xA; //R10
    *(--threadCollection[stackID].TSP) = 0x9; //R9
    *(--threadCollection[stackID].TSP) = 0x8; //R8
    *(--threadCollection[stackID].TSP) = 0x7; //R7
    *(--threadCollection[stackID].TSP) = 0x6; //R6
    *(--threadCollection[stackID].TSP) = 0x5; //R5
    *(--threadCollection[stackID].TSP) = 0x4; //R4

  return 0;
}
```