```c
#include "_kernelCore.h"
#include "osDefs.h"

extern threadStruct threadCollection[MAX_THREADS];
extern int numThreads;
int threadCurr = 0;

//set priority of the PendSV interrupt
void kernelInit(void){
  SHPR3 |= 0xFF << 16;
}

//start running the kernel, i.e. the OS
bool osKernelStart(){
  if(numThreads > 0)
  {
    __set_CONTROL(1<<1); //enter threading
    __set_PSP((uint32_t) threadCollection[threadCurr].TSP); //set PSP to the first thread address

    osLoadFirst(); //begin running threads
  }

  return false; //once called, function should not end unless something went wrong in OS
}

//start running the first thread, which will lead into context switching between all the threads
void osLoadFirst(){
  //call context switching routine, while leaving PSP set to the first thread so that it ends up running (threadCurr stays equal to 0)
    ICSR |= 1<<28;
    __asm("isb");
}

//schedule the next thread to run and call the context switcher
void osSched(void){
  //move TSP of the running thread 16 memory locations lower, so that next time the thread loads the 16 context registers, we end at the same PSP
  threadCollection[threadCurr].TSP = (uint32_t*)(__get_PSP()-16*4);

  //cycle through the threads in the thread struct array
  if (numThreads > 1){
    threadCurr = (threadCurr+1)%numThreads;
  }

  //call context switching routine, which will use a PSP to a new thread when it starts loading in register contents (updated threadCurr)
  ICSR |= 1<<28;
  __asm("isb");
}

int task_switch(void){
  //set PSP to the thread we want to start running
  __set_PSP((uint32_t)threadCollection[threadCurr].TSP);

  return 0;
}
```