CSE430: Operating Systems
Project 2 Solution
Vageesh Bhasin

General Approach:

- A data structure to store thread data is used to store the *thread_id, for_loop_start* and *for_loop_end* indexes.
- Each *'pthread_create'* method call is supplied with a pointer to this data structure to extract relevant variables.
- Total number of threads is extracted from *'num_threads'* directive and places as a global constant in the output file.
- The statement block to be executed in parallel are extracted into a separate *parallel* function, which is then supplied to the *'pthread_create'* method call.
- Thread Private variables are declared and defined as local variables in the parallel function.
- Thread Shared variables are declared as global variables.
- After thread creation, a *for* loop is used to join all the threads into the *main* thread and continue execution.

Implementation of each directive:

- **Parallel:**
  A *for* loop is generated to create threads. Within the *for* loop body, *thread_id* is assigned to the data structure variable. The *pthread_create* method call is then passed a *parallel_function* name and the data structure pointer.

- **Parallel For:**
  Similar to parallel construct, a *for* loop is used to create threads. The for loop *start iteration, end iteration* and *total iterations* are extracted from the for loop definition. Then, work allocation to be done by the threads is calculated using (*total iterations / number of threads).* Extra iterations are also calculated using the modulo operator and are also divided between the threads.

- **Critical:**
  Mutex is being used to allow only one thread to gain access to the critical section. The critical section body in the parallel function is wrapped with a *mutex_lock* and a *mutex_unlock.*

- **Single:**
  A mutex is being used to allow only one thread to access the single section at a time. Within the single section, a flag is being used which is always true at first access. After the first access, the thread changes the flag to false. Therefore, after the first thread, all other threads will not execute the section block. Also, after the single section, a *barrier_wait* is used so that all threads wait for the last thread and then proceed execution.

- **Openmp_get_thread_num():**
  I'm assigning the *thread_id* within the *for* loop that creates the *pthreads.* Dereferncing the thread id is as easy as extracting the id value from the function argument, in this case the data structure member.

- **Sections:**
  A section counter is being using to count the number of sections within the program. An array of flags is used to track whether the access to a section is to be given to a thread. The whole section block is wrapped with a mutex_lock so that only one thread gains access to the block. The sections access is governed by an *if/if-else* condition, so that any thread executes exactly one section. Also, for brevity, any function call within the section block is also being passed a thread id.

- **Barrier:**
  A barrier object is defined in the scope and is initialized within the main function. The openmp directive for barrier is replaced by *barrier_wait* pthread directive. This way, all threads will wait for the last thread to reach the barrier before proceeding.

Total Input File Access:

General Parsing:
In the general parsing category, I'm parsing the file to:

1. Replace openmp_get_thread_num()
2. Extract total number of threads
3. Extract header information
4. Extract global variables (shared variables)
5. Extract local variables (private variables)
6. Create thread creation body

Special Construct Parsing:
In this parsing category, I'm parsing the file to loop for special construct and create a body for them:

1. For construct
2. Critical construct
3. Barrier construct
4. Single construct
5. Sections construct

Joining the pieces:
After parsing the file to extract specific information, I'm then parsing the file to decide the order of placement of these special blocks within my parallel function. After creating the body of my parallel function, I then parse the file to extract any other extra functions and the main function. Lastly, I write all these blocks to my output file. After writing, I re-organize the file so that any extra tab spaces or new lines are removed and the file looks much cleaner.