1. b. Heap memory
   c. Global variables

2. No. The kernel does not know the user-level threads that are created. And OS can only see one thread and run on one processor.

3. Yes. Parallelism occurs when two or more action are being performed at the same time. Concurrency means where two different actions or thread start working together in an overlapped time period, which doesn't mean they run at same instant.

4. 95% parallelizable with
   (a) 8 process: Maximum Speedup = 1 / [0.05 + (1 – 0.05) / 8] = 5.93
   (b) 16 process: Maximum Speedup = 1 / [0.05 + (1 – 0.05) / 16] = 9.14

   50% parallelizable with
   (a) 8 process: Maximum Speedup = 1 / [0.5 + (1-0.5) / 8] = 1.78
   (b) 16 process: Maximum Speedup = 1 / [0.5 + (1 – 0.5) / 16] = 1.88

5. 1) I will create two threads. One for input and one for output
   2) I will create four threads. There are four processors, and they can run parallelism. Less than four threads would lead some core idle, and more than four threads will create context switch time.

6. Pseudocode (change main function only):

```
int main(int argc, char *argv[]){
    pid_t pid;
    pthread_t tid[4];
    pthread_attr_t attr;
    pid = fork();
    if (pid == 0){
        pthread_attr_init (&attr);
        for i=0 to 4
            pthread_join (tid[i], null);
            id[i] = i;
        Print (“CHILD PROCESS FINISHED\n”);
    }
    else if (pid > 0){
        waitpid (pid. null, 0);
        Print (“PARENT PROCESS FINISHED\n”);
    }
    Return 0;
}
```