Prashant Yadav

CS 4348.002 – Program 1 Report

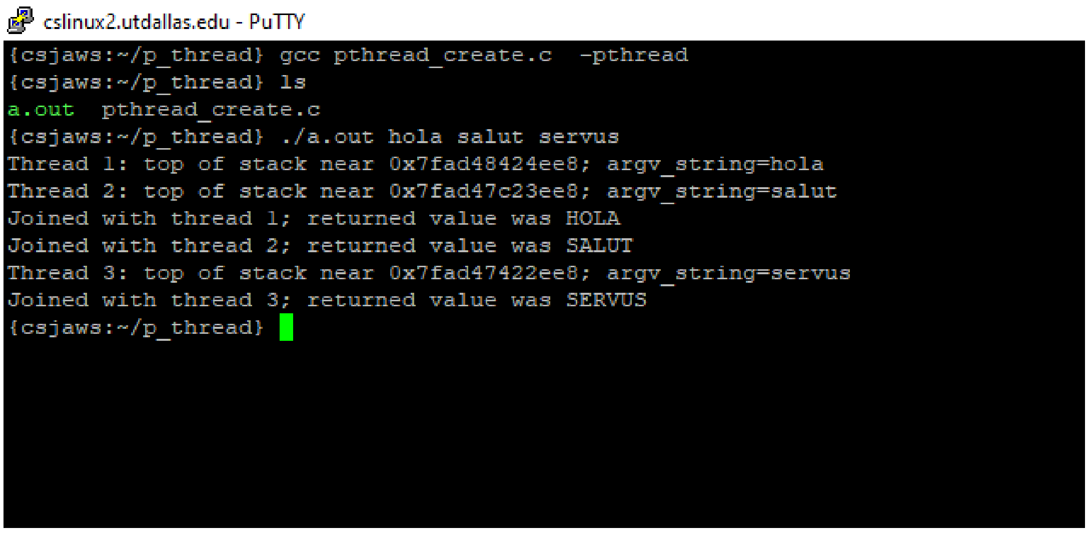
Problem statement –
Learn the creation of pthread by understanding source code given here.

Approach to solution –
I went through the C code at the given source. I ran the c code in university provided linux machine to understand it. I took help from online resources to understand few built in functions like strdup, getopt and strtoul. There are many structures that are used for pthread creation for example pthread_attr_t, thread_info, and pthread_t etc.

Solution description –
Basically, while creating pthread we have to consider following things/

- Initialize pthread_attr_t structure to hold attributes of pthread using pthread_attr_init routine. If nothing specified it will take default values to for all the properties.
  Example - *s = pthread_attr_init(&attr);*
- Consider stack size for each thread based on amount of work each thread has to do.
  Example - *s = pthread_attr_setstacksize(&attr, stack_size);*
- Initialize required memory that will be consumed by each thread while execution. It depends on number of threads and structure thread info. Below is an example of memory allocation.
  Example - *tinfo = calloc(num_threads, sizeof(struct thread_info));*
- Use pthread_create routine to create child threads. This routine will take thread_info, thread_id and address of the method which needs to be executed by each thread.
  Example - *s = pthread_create(&tinfo[tnum].thread_id, &attr, &thread_start, &tinfo[tnum]);*
- After creation threads will execute thread_start routine and hold the result in res property. Now we will have to make join call to wait for all the child thread to complete their execution. Here is the code snippet for the same.
  s = pthread_join(tinfo[tnum].thread_id, &res);
- Once all threads complete their execution, we have to do clean up activities. Delete attributes and free up the memory used by the pthreads. So that resources will be available for other process.
  free(res);

Here is snapshot of code execution –
I created pthread_create.c file. Command used to compile and create output file is *gcc pthread_create.c -pthread*. This command created a.out compile bytecode file. After that I executed *./a.out hola salut servus.* In this command a.out is our file name and hola, salut and servus are command line arguments. Code creates 3 thread, one for each command line argument, and converts command argument to Uppercase and later joins each thread. From the snapshot it is clear that Thread1 and Thread2 finishes there execution even before Thread3 starts its execution.



cslinux2.utdallas.edu - PuTTY

```
{csjaws:~/p_thread} gcc pthread_create.c  -pthread
{csjaws:~/p_thread} ls
a.out  pthread_create.c
{csjaws:~/p_thread} ./a.out hola salut servus
Thread 1: top of stack near 0x7fad48424ee8; argv_string=hola
Thread 2: top of stack near 0x7fad47c23ee8; argv_string=salut
Joined with thread 1; returned value was HOLA
Joined with thread 2; returned value was SALUT
Thread 3: top of stack near 0x7fad47422ee8; argv_string=servus
Joined with thread 3; returned value was SERVUS
{csjaws:~/p_thread}
```