```
/* CPSC 457 (Winter 2019)
 * Week 4 - Section 1
 * Tutorial
 * Sina Keshvadi
 *
 * Notes: No error handling!
 */
=================================================================================
pthread_create() function to create a thread.
pthread_create() takes 4 arguments, namely: thread, attr, start, arg.
The first argument is a pointer to thread_id which is set by this function.
the second is Null (If attr is NULL, the default attributes are used).
The third argument is name of function to be executed for the thread to be created.
The fourth argument is used to pass arguments to thread.
=================================================================================
// Sample thread in c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUMBER_OF_THREADS 2

void * thread_print(void * tid)
{
  printf("thread %ld running\n", tid);
  pthread_exit(0);
}

int main()
{
  pthread_t threads[NUMBER_OF_THREADS];
  long status, i;
  for (i = 0; i < NUMBER_OF_THREADS; i++)
  {
    printf("creating thread %d\n", i);
    status = pthread_create(&threads[i], NULL, thread_print, (void *) i);
    if (status != 0)
        {
      printf("Oops, pthread_create returned error code %d\n", status);
      exit(-1);
    }
  }
  for (i = 0; i < NUMBER_OF_THREADS; i++)
    pthread_join(threads[i], NULL);
  exit(0);
}
=================================================================================
// Sample two threads in c

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define numberOfThread  2

void * print(void * tid)
{
        if(tid%2==0)
                for(int j=0; j<=10; j+=2)
                {   printf("Thread %d : \n", tid);
                        printf("%d\n", j);
                }
        else
                for(int j=1; j<=10; j+=2)
                {   printf("Thread %d : \n", tid);
                        printf("%d\n", j);
                }
        pthread_exit(0);
}

int main()
{
        pthread_t myThreads[numberOfThread];
        int status, i;

        for(i=1; i<=numberOfThread; i++)
        {
                printf("Creating thread #%d\n", i);
                status=pthread_create(&myThreads[i],NULL, print, (void *) i);
        }

}
```

===

```
// Write the following code in c

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>   // for sleep

void *print_hello_world(void *unused)
{
  while(1)
  {
    printf("Hello Word\n");
    sleep(1);
  }
  return NULL;
}

int main(int argc, char *argv[])
{
  pthread_t mythread;
  printf("Main here ...\n");
  pthread_create(&mythread, NULL, &print_hello_world, NULL);

  while(1)
  {
    printf("Hello CPSC\n");
    sleep(1);
  }
  return 0;
}
```

a. compile using: `gcc –lpthread multithread1.c -o multithread1`
b. Run the code for about 1-2 minutes (press ctrl+c to terminate).
Is the output has a unique pattern or differs with time? Reason.

===

Comment the while loop inside main(), compile and run the program. Do you notice any changes in the output? Reason.

===

Create a file called multithread2.c and write the following piece of code in it:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUMBER_OF_THREADS 10
```

```
void *print_hello_world(void * tnum)
{
  printf("I am thread Number %d\n", *(int*)tnum);
}

int main(int argc, char *argv[])
{
  pthread_t threads[NUMBER_OF_THREADS];
  int status, i;

  for (i = 0; i < NUMBER_OF_THREADS; i++)
  {
    pthread_create(&threads[i], NULL, print_hello_world, (void *) &i);
  }

  for (i = 0; i < NUMBER_OF_THREADS; i++)
    pthread_join(threads[i], NULL);
}
===================================================================================
Run the code 5 times. Is the output has a unique pattern or differs with time? Reason.
===================================================================================
Below program is written to achieve the following features (multithread3.c)

1. Create 10 threads.
2. The thread number (NOT the thread id) is passed as an argument to the thread
function.
3. Each thread should print its corresponding thread number inside its function.
4. Main thread should wait for all the threads to complete. Program should terminate
after this.

Check whether all the 4 features are successfully implemented. If not, try to reason
why it is so and how you can fix the problem?

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUMBER_OF_THREADS 10

void *print_hello_world(void * tnum) {
  printf("Hello Word. Greetings from thread %d\n", *(int*)tnum);
}

int main(int argc, char *argv[]) {
  pthread_t threads[NUMBER_OF_THREADS];
  int status, i;

  for (i = 0; i < NUMBER_OF_THREADS; i++){
    status = pthread_create(&threads[i], NULL, print_hello_world, (void *) &i);
    if (status != 0) {
      printf("Oops, pthread_create returned error code %d\n", status);
      exit(-1);
    }
  }

  for (i = 0; i < NUMBER_OF_THREADS; i++)
    pthread_join(threads[i], NULL);
}

===================================================================================
// try this code in linux terminal to find the number of cpu cores
grep -c ^processor /proc/cpuinfo
or
nproc --all
```

```
==================================================================================
// sum up 100 numbers by using 4 threads

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

struct sum_struct
{
        int first;
        int last;
        int sum;
};


// Thread Sum Function to sum 0 to N
void* writeNumbers (void* arg)
{
        // casting arg from void* to desired type
        struct sum_struct *arg_struct = (struct sum_struct*) arg;
        arg_struct->sum = 0;
        for(int i=arg_struct->first; i<=arg_struct->last; i++)
        {
                printf(" %d, ", i);
                arg_struct->sum+=i;
        }
        printf("\n\n");
        pthread_exit(0);
}

int main(int argc, char **argv)
{
        int num_thread = 4;
        int main_number = 100;
        struct sum_struct args[num_thread];

        // Create Multi Threads
        pthread_t tids[num_thread];

        for(int i=0; i<num_thread; i++)
        {
                args[i].first = (i*(main_number/num_thread))+1;
                args[i].last = (i+1)*(main_number/num_thread);
                pthread_attr_t attr;
                pthread_attr_init(&attr);
                pthread_create(&tids[i], &attr, writeNumbers, &args[i]);
        }

        int overal_sum = 0;
        for(int i=0; i<num_thread; i++)
        {
                pthread_join(tids[i], NULL);
                overal_sum += args[i].sum;
                printf("Sum of thread %d is : %d \n", i, args[i].sum);
        }

        printf("Overal Sum is : %d \n", overal_sum);
        return(0);
}
==================================================================================
// Using wait() (written in c++)

#include <iostream>
#include <cstdio>
```

```
#include <unistd.h>
#include <sys/wait.h>

using namespace std;

int main()
{
        int status;
        pid_t pid;

        if (fork())
        {
                cout<<"I am Parent and want to wait"<<endl;
                pid = wait(&status);
                for(int i=1; i<=4; i++)
                        cout<<i<<endl;
                cout<<"Child ID is : "<<pid<<endl       ;
        }
        else
        {
                cout<<"I am child. My ID is: "<<getpid()<<endl;
                for(char i='a'; i<='f'; i++)
                        cout<<i<<endl;
        }
        cout<<"Good Bye"<<endl;

   return 0;
}
==================================================================================
// Global Varible in Fork example
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int x;

void do_something()
{
    x = 11;
    exit(0);
}

int main()
{
  x = 10;
  int pid = fork();
  if( pid == 0)
  {
    do_something();
  }
  else
  {
    wait( NULL);
  }
  printf("x=%d\n", x);
}

// Before running the code, what is your guess about the output?
// remove     exit(0); in do_something() and run the code? what is the output? Why?
```