•Tất cả các em ghi họ tên & MSSV lên ô chat của Google Meet.

Lab 1:

•Cài Virtual Machine

•Cài Ubuntu, Fedora, Kali, k8, CentOS

- Setup VM: https://youtu.be/wX75Z-4MEoM
- Virtual Box: https://www.virtualbox.org/
- Ubuntu: https://ubuntu.com/download/desktop
- Linux command: Introduction to Linux and Basic Linux Commands for Beginners
- Compile C program in Ubuntu: https://youtu.be/oLjN6jAg-sY

Content:

- Practice with vim - text editor.
- Programming with C language.
- Compile a program with Makefile.
- How to retrieve the information of running process? How is PCB table controlled by OS?

- Video tutorial for vim: https://youtu.be/wlR5gYd6um0
- Create a program with multiple processes.

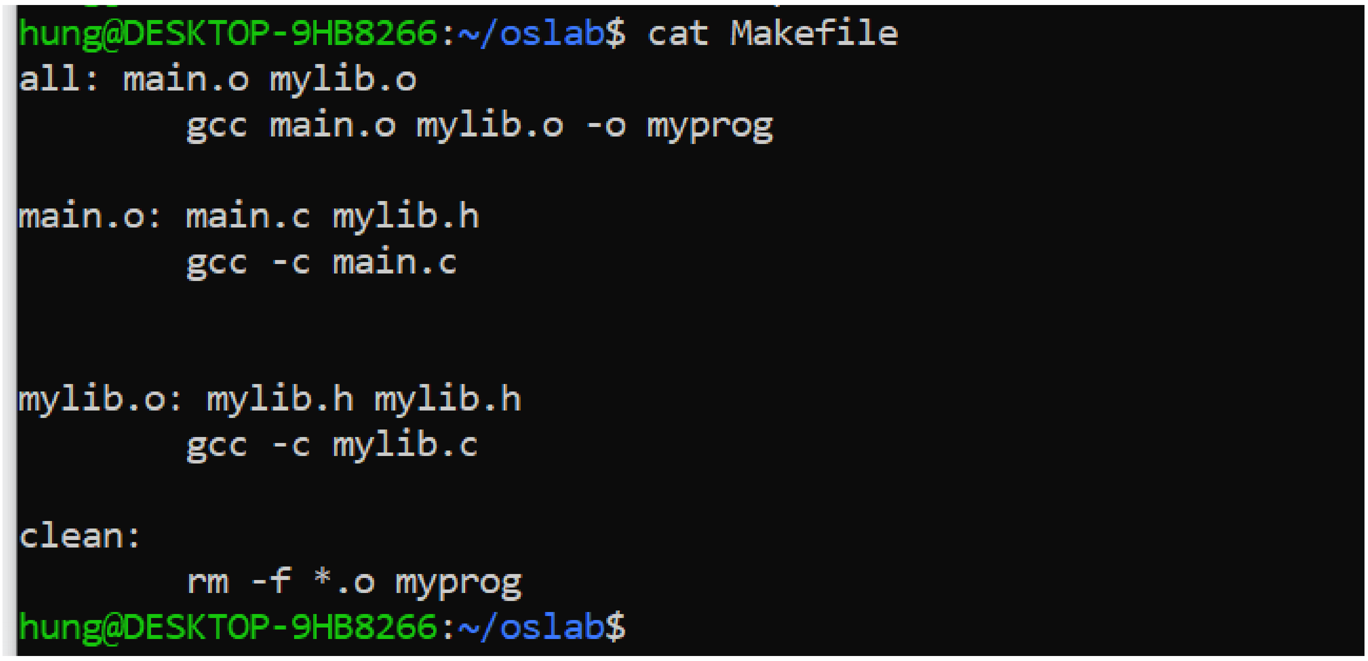
- C Programming: C Programming Tutorial 1 - Intro to C

- C programming Linux video: C Programming Tutorial 6 - Intro to UNIX/Linux - Part 1
  C Programming Tutorial 7 - Intro to UNIX/Linux - Part 2

- Linux command line interface: Linux Tutorial - Basic Command Line

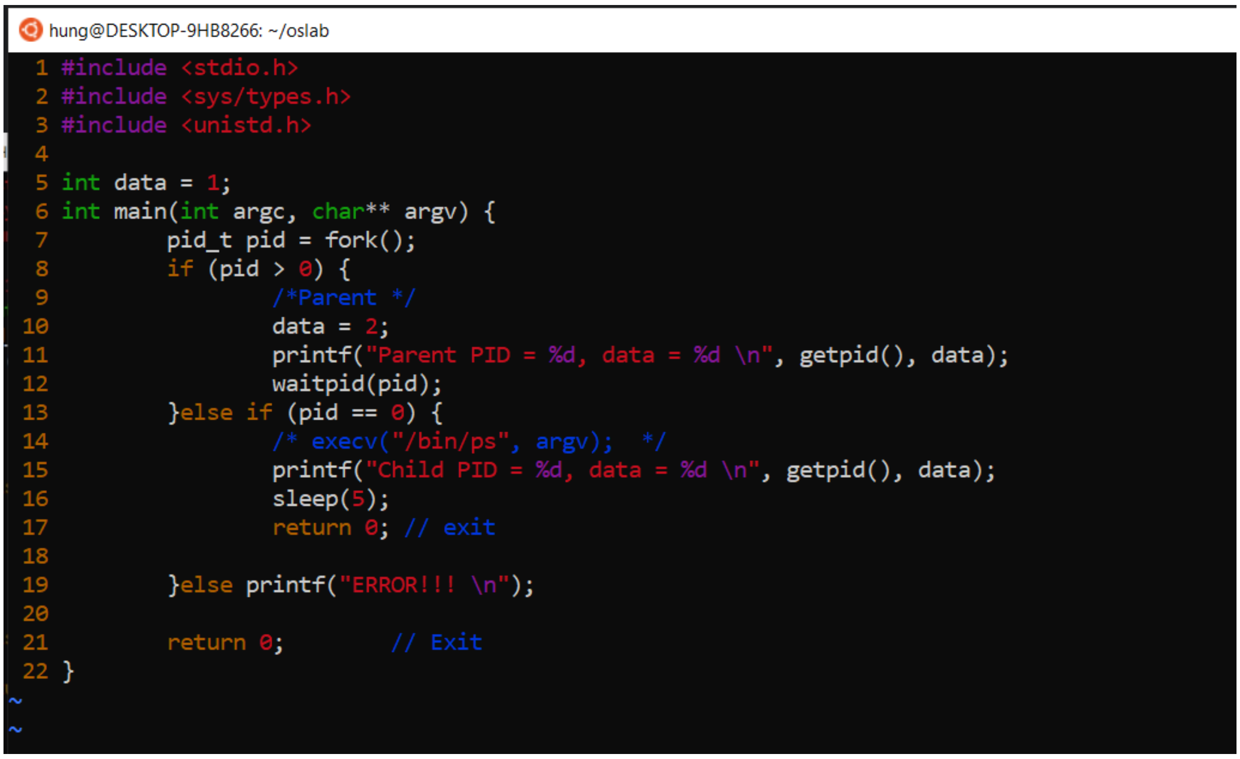
- C tutorial: https://www.tutorialspoint.com/cprogramming/index.htm
- GNU make: https://www.gnu.org/software/make/manual/make.html
- How to use make: C for Beginners: Creating makefiles for C Programs using CLANG/GCC
- C datatype: https://www.tutorialspoint.com/cprogramming/c_data_types.htm
- GNU GCC compilation: How to Compile and Run C program Using GCC on Ubuntu 18.04 LTS (Linux) / Ubuntu 20.04 LTS
- Làm Lab 1 - Test 1
- Nộp toàn bộ source code của Lab 1 - Programming code. Nén lại 1 file .zip/.tar.gz và nộp lên BKel.

Makefile: UPEvent: GCC and Makefiles



```
hung@DESKTOP-9HB8266:~/oslab$ cat Makefile
all: main.o mylib.o
        gcc main.o mylib.o -o myprog

main.o: main.c mylib.h
        gcc -c main.c


mylib.o: mylib.h mylib.h
        gcc -c mylib.c

clean:
        rm -f *.o myprog
hung@DESKTOP-9HB8266:~/oslab$
```

Create new child process



hung@DESKTOP-9HB8266: ~/oslab

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int data = 1;
int main(int argc, char** argv) {
        pid_t pid = fork();
        if (pid > 0) {
                /*Parent */
                data = 2;
                printf("Parent PID = %d, data = %d \n", getpid(), data);
                waitpid(pid);
        }else if (pid == 0) {
                /* execv("/bin/ps", argv);  */
                printf("Child PID = %d, data = %d \n", getpid(), data);
                sleep(5);
                return 0; // exit

        }else printf("ERROR!!! \n");

        return 0;        // Exit
}
~
~
```

Shell Scripting Tutorial:

[Shell Scripting Tutorial | Shell Scripting Crash Course | Linux Certification Training | Edureka]

Git & GitHub Crash Course For Beginners

- Video Git: [Git & GitHub Crash Course For Beginners]

Shell Scripting Tutorial

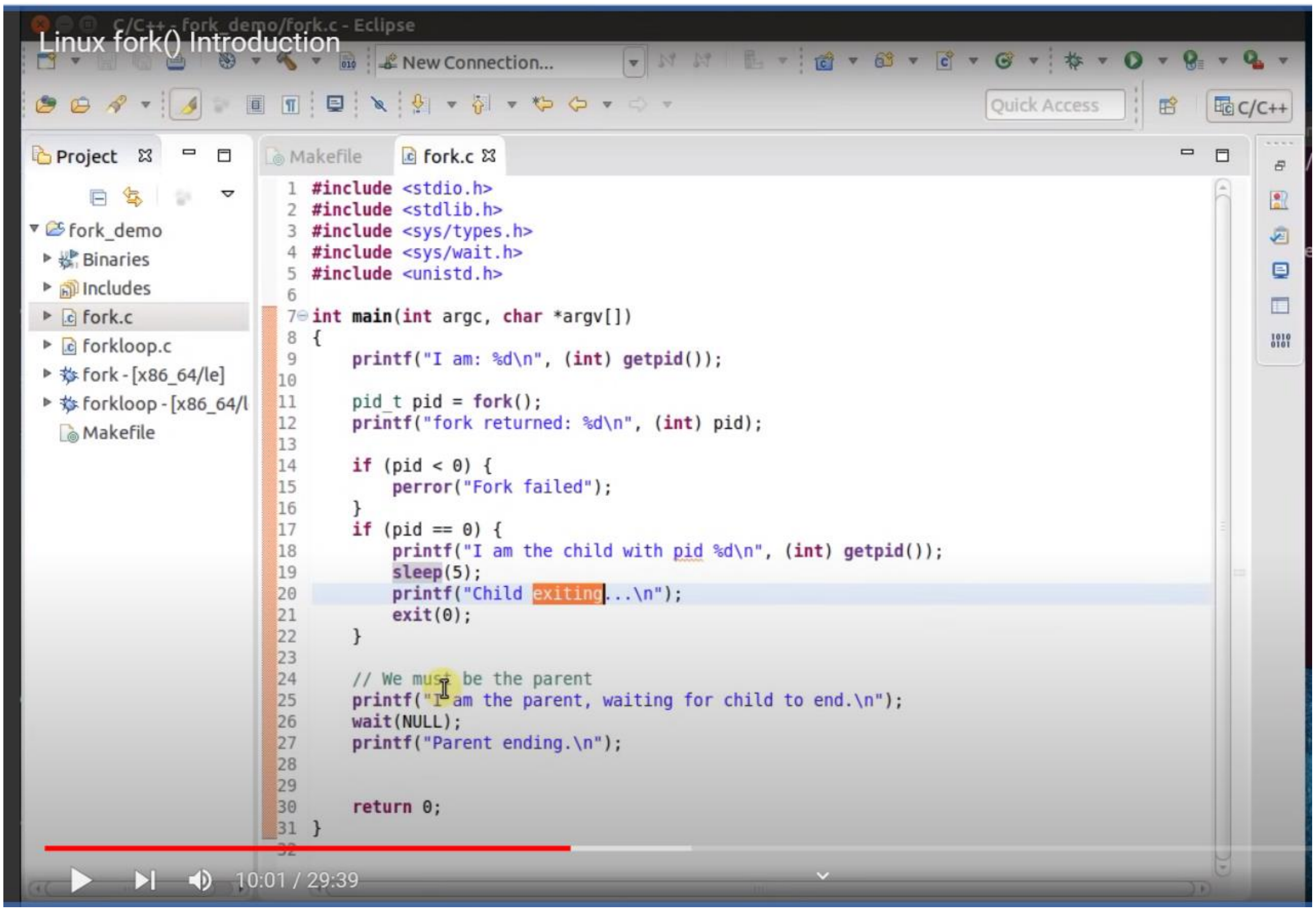
- https://www.tutorialspoint.com/unix/shell_scripting.htm
-

Kafka & Concluent: https://docs.confluent.io/platform/current/quickstart/ce-docker-quickstart.html

-

Create child process: fork()

- Linux fork() Introduction



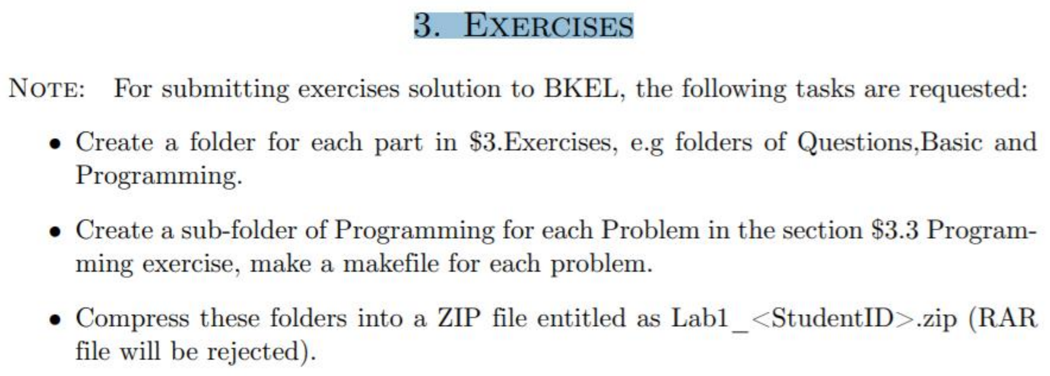
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    printf("I am: %d\n", (int) getpid());

    pid_t pid = fork();
    printf("fork returned: %d\n", (int) pid);

    if (pid < 0) {
        perror("Fork failed");
    }
    if (pid == 0) {
        printf("I am the child with pid %d\n", (int) getpid());
        sleep(5);
        printf("Child exiting...\n");
        exit(0);
    }

    // We must be the parent
    printf("I am the parent, waiting for child to end.\n");
    wait(NULL);
    printf("Parent ending.\n");


    return 0;
}
```

[HK221] THÔNG TIN ĐĂNG KÝ ĐỀ TÀI BTL/NGHIÊN CỨU MÔN HỌC HĐH HK212/2022-2023

https://docs.google.com/spreadsheets/d/1qNYGPAXf9KITupB6cP9PKB-X-yHdEUXwuln3d4Pfnpo/edit?usp=sharing

Tasks:

- Làm bài Lab 1 - Test 2.
- Codepost.io: https://codepost.io/signup/join?code=8KDCWAN2AN
- Nộp Lab 1: Problem 4 (lập trình) vào CodePost.io
- Problem 4 Given a file named "numbers.txt" containing multiple lines of text. Each line is a non-negative integer. Write a C program that reads integers listed in this file and stores them in an array (or linked list). The program then uses the fork() system call to create a child process. The parent process will count the numbers of integers in the array that are divisible by 2. The child process will count numbers divisible by 3. Both processes then send their results to the stdout. For examples, if the file "numbers.txt" contains the following lines
- Tất cả 3. EXERCISES còn lại submit lên BKel, Bài Lab 1.



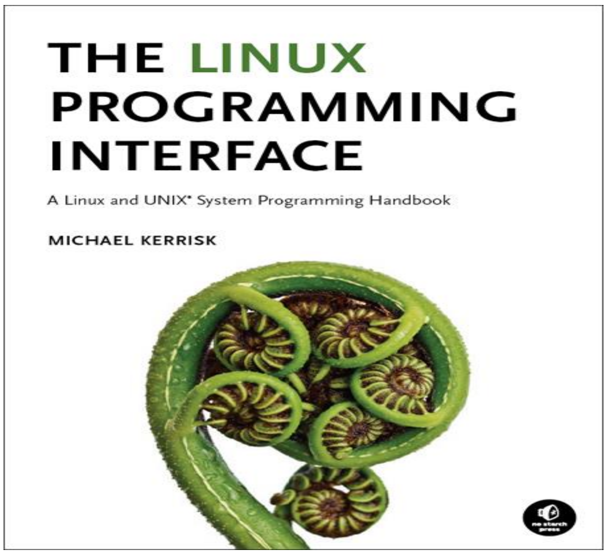
3. EXERCISES

NOTE: For submitting exercises solution to BKEL, the following tasks are requested:

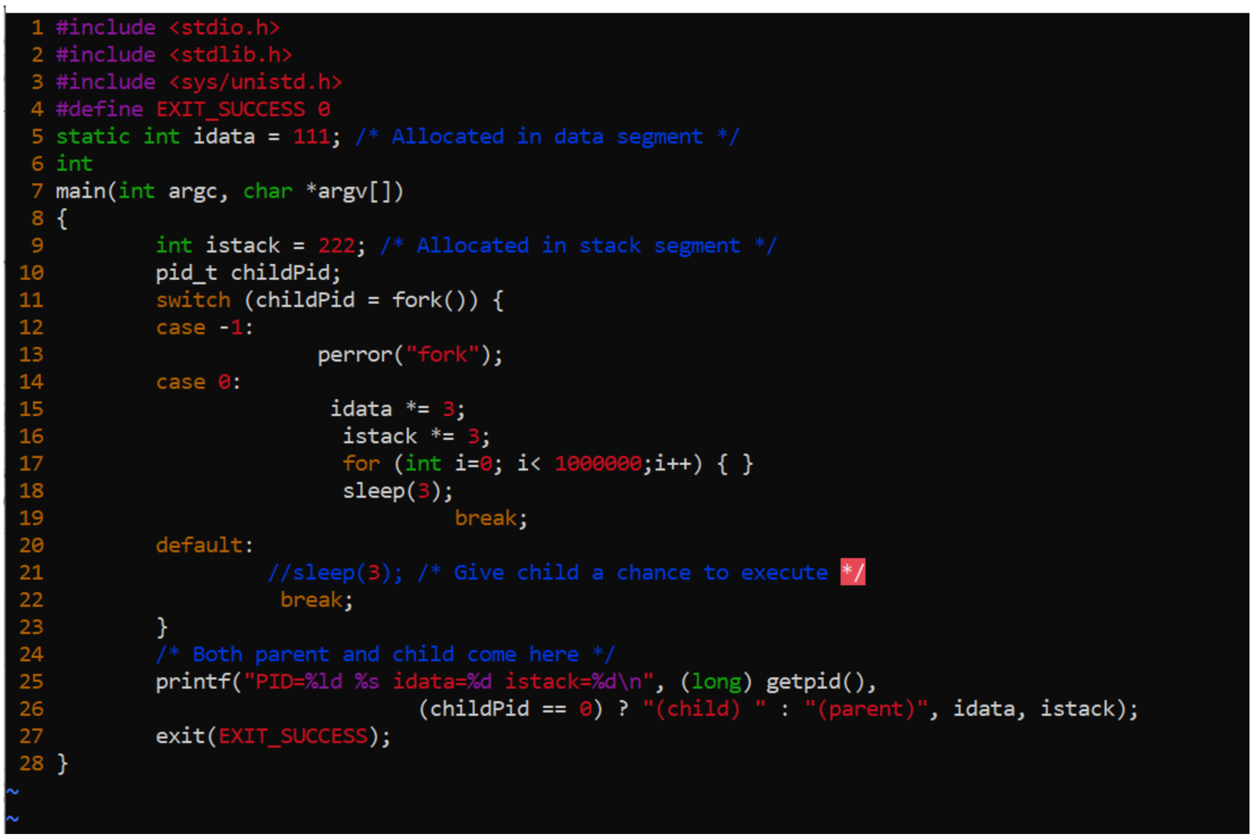
- Create a folder for each part in $3.Exercises, e.g folders of Questions,Basic and Programming.
- Create a sub-folder of Programming for each Problem in the section $3.3 Programming exercise, make a makefile for each problem.
- Compress these folders into a ZIP file entitled as Lab1_<StudentID>.zip (RAR file will be rejected).

Book: The Linux Programming Interface

- Link: https://man7.org/tlpi/index.html


THE LINUX PROGRAMMING INTERFACE
A Linux and UNIX® System Programming Handbook
MICHAEL KERRISK

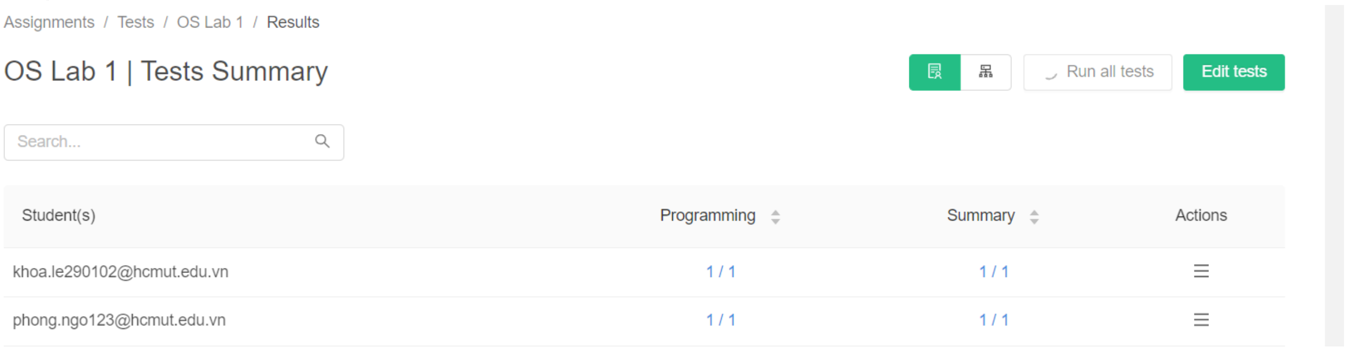
- Một số ví dụ khác về gọi fork() system call tạo process, khảo sát dữ liệu chia sẻ *idata*.



```
#include <stdio.h>
#include <stdlib.h>
#include <sys/unistd.h>
#define EXIT_SUCCESS 0
static int idata = 111; /* Allocated in data segment */
int
main(int argc, char *argv[])
{
        int istack = 222; /* Allocated in stack segment */
        pid_t childPid;
        switch (childPid = fork()) {
        case -1:
                     perror("fork");
        case 0:
                      idata *= 3;
                       istack *= 3;
                       for (int i=0; i< 1000000;i++) { }
                       sleep(3);
                                break;
        default:
                  //sleep(3); /* Give child a chance to execute */
                   break;
        }
        /* Both parent and child come here */
        printf("PID=%ld %s idata=%d istack=%d\n", (long) getpid(),
                             (childPid == 0) ? "(child) " : "(parent)", idata, istack);
        exit(EXIT_SUCCESS);
}
~
~
```

CodePost.io: Autograde

Kết quả:


Assignments / Tests / OS Lab 1 / Results

OS Lab 1 | Tests Summary

Student(s)	Programming	Summary	Actions
khoa.le290102@hcmut.edu.vn	1 / 1	1 / 1	☰
phong.ngo123@hcmut.edu.vn	1 / 1	1 / 1	☰

Process states

Zombie, orphan process: https://www.tutorialspoint.com/zombie-and-orphan-processes-in-linux

Lab 2:

MULTITHREADED PROCESS

ulimit

Cách tìm ulimit trên Linux

Cách tìm ulimit cho người dùng trên Linux

Lệnh ulimit Linux đặt hoặc hiển thị giới hạn tài nguyên tiến trình người dùng. Thông thường, các giới hạn được tìm thấy trong tệp /etc/security/limits.conf hoặc systemd.

Hai loại giới hạn

Tất cả các giới hạn Linux đều được chia làm hai loại:

- Soft limit
- Hard limit

Xem các ulimit cho tài khoản người dùng của Linux

Nhập lệnh sau để xem tất cả các soft và hard limit cho người dùng hiện tại:

```
ulimit -Sa ## Show soft limit ##
ulimit -Ha ## Show hard limit ##
```

$ ulimit -Sa

```
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 7823
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 7823
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

$ ulimit -Ha

```
core file size          (blocks, -c) unlimited
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 7823
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 65536
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 7823
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

Dynamic memory:

Dynamic memory allocation in C - malloc calloc realloc free

C Dynamic Memory Debugging with Valgrind

malloc() function

Description

The C library function void *malloc(size_t size) allocates the requested memory and returns a pointer to it.

Declaration

Following is the declaration for malloc() function.

```
void *malloc(size_t size)
```

Parameters

- size − This is the size of the memory block, in bytes.

Return Value

This function returns a pointer to the allocated memory, or NULL if the request fails.

Example

The following example shows the usage of malloc() function.

Live Demo

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

```c
int main () {
   char *str;

   /* Initial memory allocation */
   str = (char *) malloc(15);
   strcpy(str, "tutorialspoint");
   printf("String = %s,  Address = %u\n", str, str);

   /* Reallocating memory */
   str = (char *) realloc(str, 25);
   strcat(str, ".com");
   printf("String = %s,  Address = %u\n", str, str);

   free(str);

   return(0);
}
```

Let us compile and run the above program that will produce the following result −

```
String = tutorialspoint, Address = 355090448
String = tutorialspoint.com, Address = 355090448
```

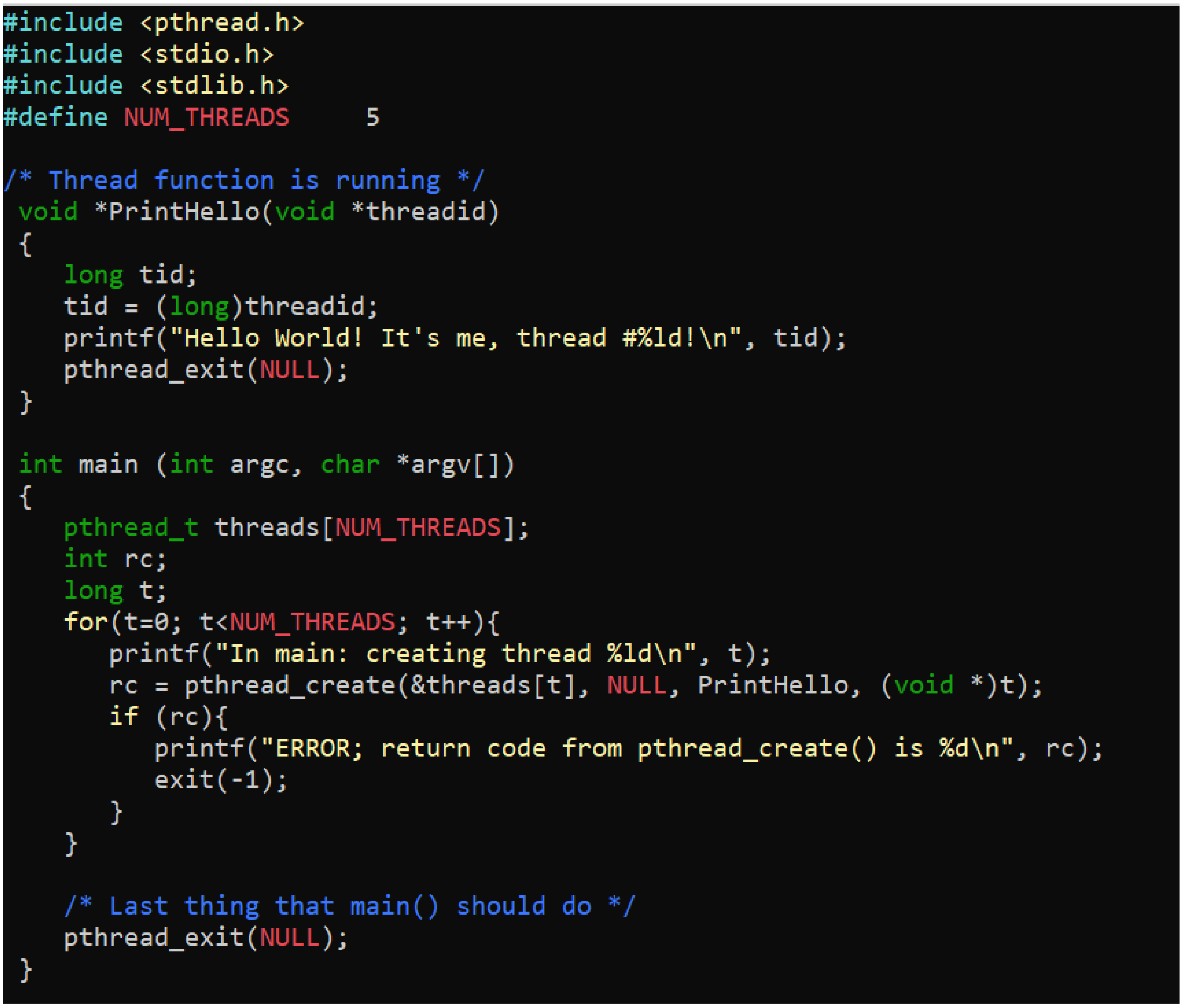
Pthread:

- Tạo file source code taothread.c
- Compile:
  $ gcc taothread.c -o taothread -lpthread

POSIX Threads Programming

- **POSIX Threads Programming,** https://hpc-tutorials.llnl.gov/posix/
-
- https://hpc-tutorials.llnl.gov/posix/mutex_variables/
- https://randu.org/tutorials/threads/

Tạo code: taothread2.c

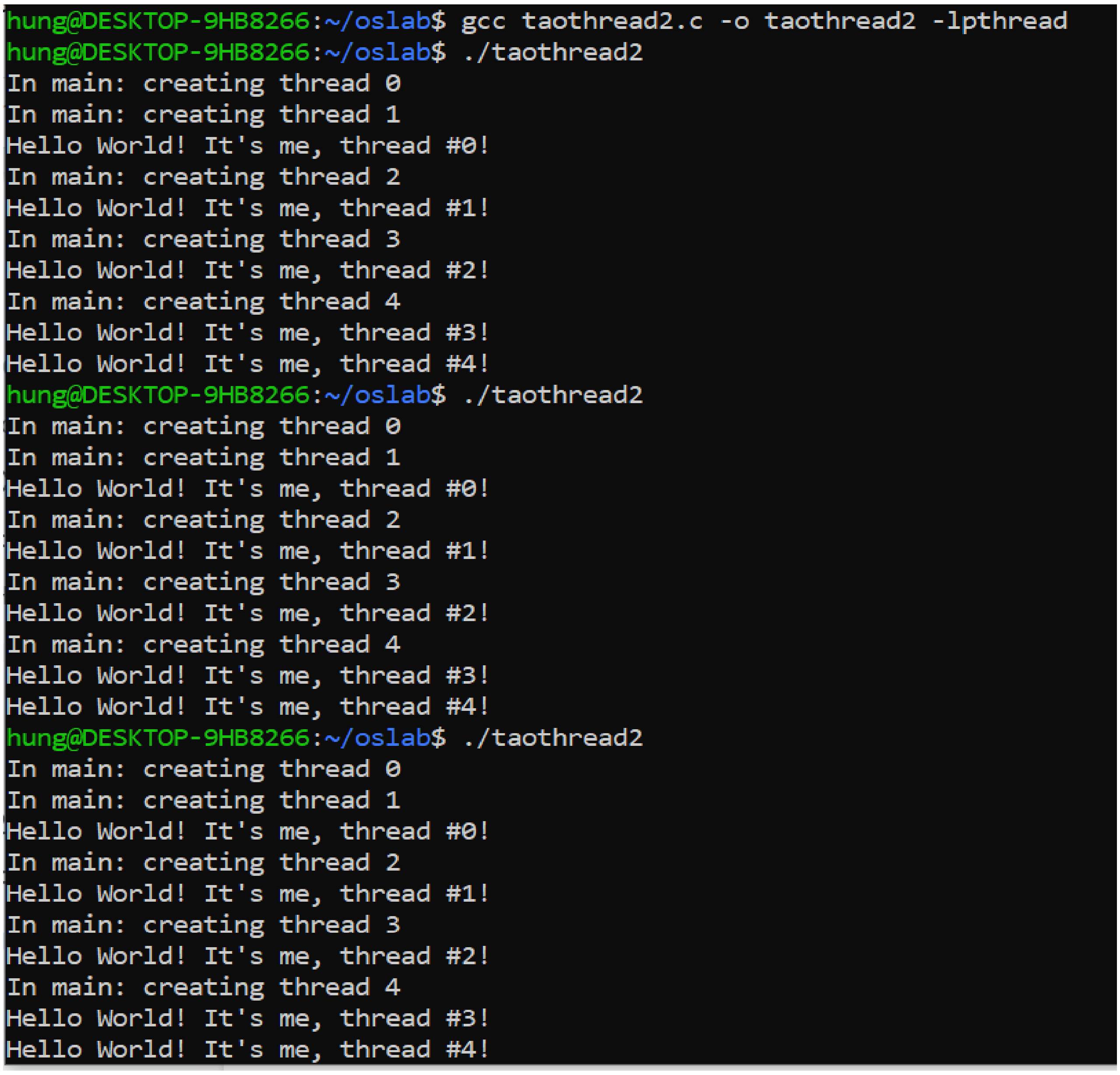


```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     5

/* Thread function is running */
 void *PrintHello(void *threadid)
 {
    long tid;
    tid = (long)threadid;
    printf("Hello World! It's me, thread #%ld!\n", tid);
    pthread_exit(NULL);
 }

 int main (int argc, char *argv[])
 {
    pthread_t threads[NUM_THREADS];
    int rc;
    long t;
    for(t=0; t<NUM_THREADS; t++){
       printf("In main: creating thread %ld\n", t);
       rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
       if (rc){
          printf("ERROR; return code from pthread_create() is %d\n", rc);
          exit(-1);
       }
    }

    /* Last thing that main() should do */
    pthread_exit(NULL);
 }
```

Compile:

```
$ gcc taothread2.c -o taothread2 -lpthread
```

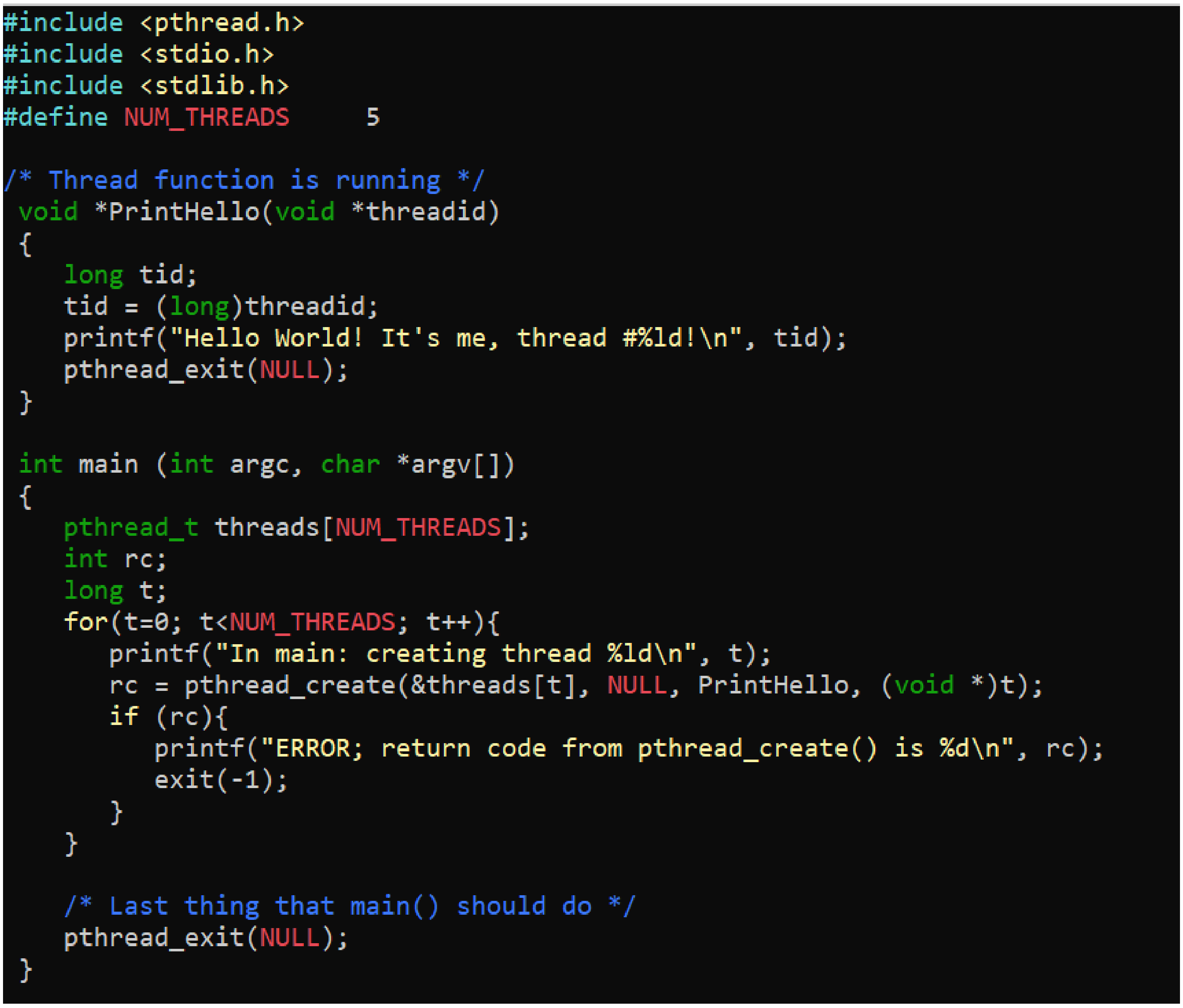
Kết quả:



```
hung@DESKTOP-9HB8266:~/oslab$ gcc taothread2.c -o taothread2 -lpthread
hung@DESKTOP-9HB8266:~/oslab$ ./taothread2
In main: creating thread 0
In main: creating thread 1
Hello World! It's me, thread #0!
In main: creating thread 2
Hello World! It's me, thread #1!
In main: creating thread 3
Hello World! It's me, thread #2!
In main: creating thread 4
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
hung@DESKTOP-9HB8266:~/oslab$ ./taothread2
In main: creating thread 0
In main: creating thread 1
Hello World! It's me, thread #0!
In main: creating thread 2
Hello World! It's me, thread #1!
In main: creating thread 3
Hello World! It's me, thread #2!
In main: creating thread 4
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
hung@DESKTOP-9HB8266:~/oslab$ ./taothread2
In main: creating thread 0
In main: creating thread 1
Hello World! It's me, thread #0!
In main: creating thread 2
Hello World! It's me, thread #1!
In main: creating thread 3
Hello World! It's me, thread #2!
In main: creating thread 4
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
```

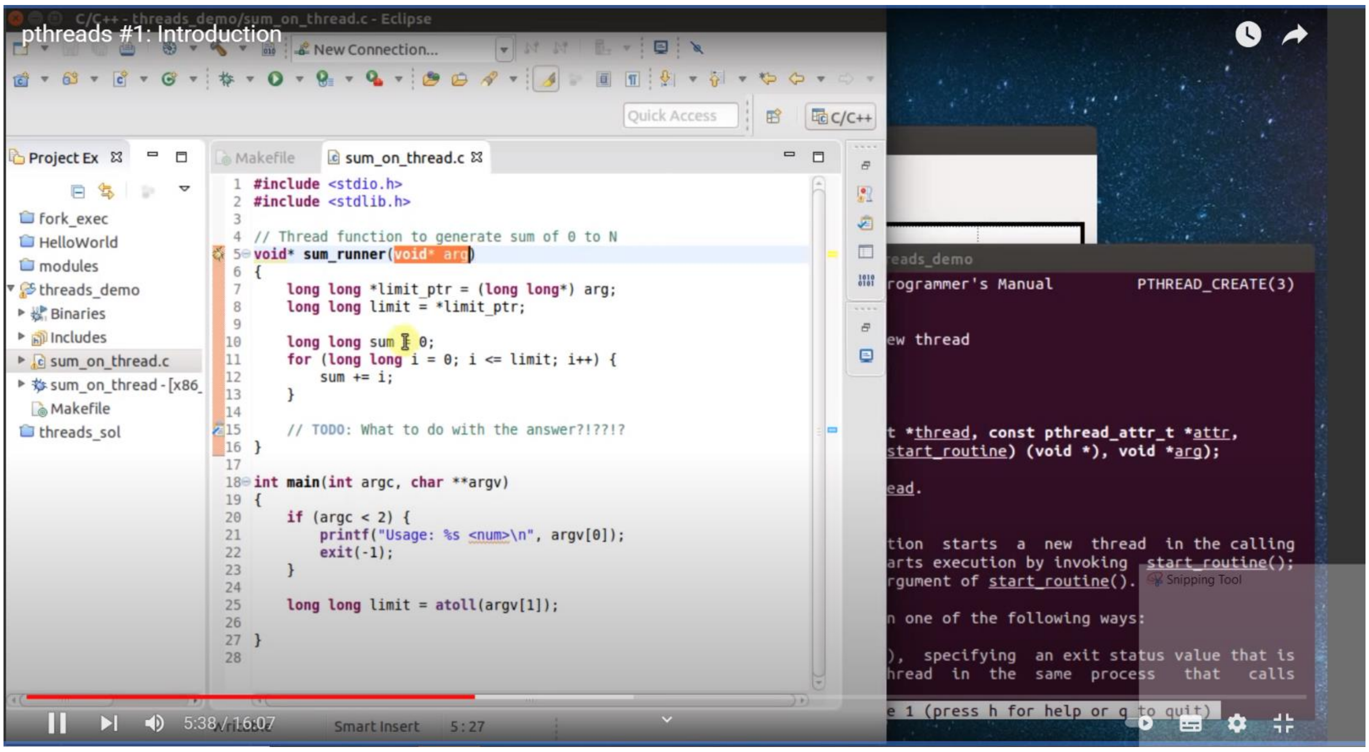
Tạo, kết thúc, và thiết lập các thuộc tính của Thread:

- https://hpc-tutorials.llnl.gov/posix/creating_and_terminating/



```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     5

/* Thread function is running */
 void *PrintHello(void *threadid)
 {
    long tid;
    tid = (long)threadid;
    printf("Hello World! It's me, thread #%ld!\n", tid);
    pthread_exit(NULL);
 }

 int main (int argc, char *argv[])
 {
    pthread_t threads[NUM_THREADS];
    int rc;
    long t;
    for(t=0; t<NUM_THREADS; t++){
       printf("In main: creating thread %ld\n", t);
       rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
       if (rc){
          printf("ERROR; return code from pthread_create() is %d\n", rc);
          exit(-1);
       }
    }

    /* Last thing that main() should do */
    pthread_exit(NULL);
 }
```

Introduction to Thread: [pthreads #1: Introduction]()



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

// Thread function to generate sum of 0 to N
void* sum_runner(void* arg)
{
    long long *limit_ptr = (long long*) arg;
    long long limit = *limit_ptr;

    long long sum = 0;
    for (long long i = 0; i <= limit; i++) {
        sum += i;
    }

    // TODO: What to do with the answer?!??!?
}

int main(int argc, char **argv)
{
    if (argc < 2) {
        printf("Usage: %s <num>\n", argv[0]);
        exit(-1);
    }

    long long limit = atoll(argv[1]);
}
```

```
rogrammer's Manual          PTHREAD_CREATE(3)

ew thread

t *thread, const pthread_attr_t *attr,
start_routine) (void *), void *arg);

ead.

tion  starts  a  new  thread  in the calling
arts execution by invoking  start_routine();
rgument of start_routine().

n one of the following ways:

),  specifying  an exit status value that is
hread  in  the  same  process  that  calls

e 1 (press h for help or q to quit)
```

Mutex Synchronization in Linux with Pthreads

[Mutex Synchronization in Linux with Pthreads]

Viết chương trình tính tổng của mảng dùng Multithread và Mutex:

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

#define NTHREADS      4
#define ARRAYSIZE   1000000
#define ITERATIONS   ARRAYSIZE / NTHREADS

double  sum=0.0, a[ARRAYSIZE];
pthread_mutex_t sum_mutex;


void *do_work(void *tid)
```

```
{
  int i, start, *mytid, end;
  double mysum=0.0;

  /* Initialize my part of the global array and keep local sum */
  mytid = (int *) tid;
  start = (*mytid * ITERATIONS);
  end = start + ITERATIONS;
  printf ("Thread %d doing iterations %d to %d\n",*mytid,start,end-1);
  for (i=start; i < end ; i++) {
    a[i] = i * 1.0;
    mysum = mysum + a[i];
    }

  /* Lock the mutex and update the global sum, then exit */
  pthread_mutex_lock (&sum_mutex);
  sum = sum + mysum;
  pthread_mutex_unlock (&sum_mutex);
  pthread_exit(NULL);
}


int main(int argc, char *argv[])
{
  int i, start, tids[NTHREADS];
  pthread_t threads[NTHREADS];
  pthread_attr_t attr;

  /* Pthreads setup: initialize mutex and explicitly create threads in a
     joinable state (for portability).  Pass each thread its loop offset */
  pthread_mutex_init(&sum_mutex, NULL);
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
  for (i=0; i<NTHREADS; i++) {
    tids[i] = i;
    pthread_create(&threads[i], &attr, do_work, (void *) &tids[i]);
    }
  /* Wait for all threads to complete then print global sum */
  for (i=0; i<NTHREADS; i++) {
    pthread_join(threads[i], NULL);
  }
  printf ("Done. Sum= %e \n", sum);

  sum=0.0;
  for (i=0;i<ARRAYSIZE;i++){
  a[i] = i*1.0;
  sum = sum + a[i]; }
  printf("Check Sum= %e\n",sum);

  /* Clean up and exit */
  pthread_attr_destroy(&attr);
  pthread_mutex_destroy(&sum_mutex);
  pthread_exit (NULL);
}
```

PI simulation

[Estimating Pi using Monte Carlo Simulation]()

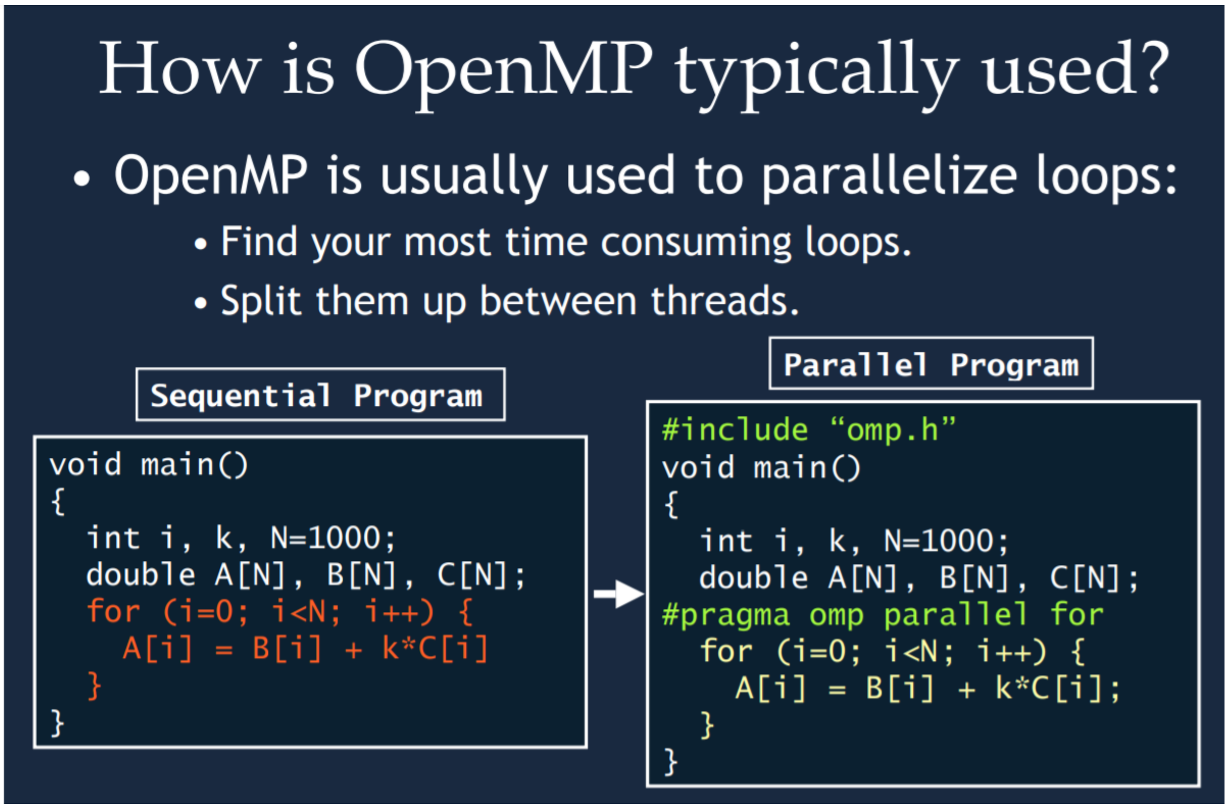
Parallel Programming Standards

- Thread Libraries
- Win32 API
- Posix threads
- Compiler Directives
- OpenMP (Shared memory programming)
- Message Passing Libraries - MPI (Distributed memory programming)

OpenMP Tutorial

- https://engineering.purdue.edu/~smidkiff/ece563/files/ECE563OpenMPTutorial.pdf

- Shared Memory Parallel Programming in the Multi-Core Era • Desktop and Laptop – 2, 4, 8 cores and … ? • A single node in distributed memory clusters – Steele cluster node: 2 8 (16) cores • Shared memory hardware Accelerators • Cell processors: 1 PPE and 8 SPEs • Nvidia Quadro GPUs: 128 processing units



How is OpenMP typically used?

- OpenMP is usually used to parallelize loops:
  - Find your most time consuming loops.
  - Split them up between threads.

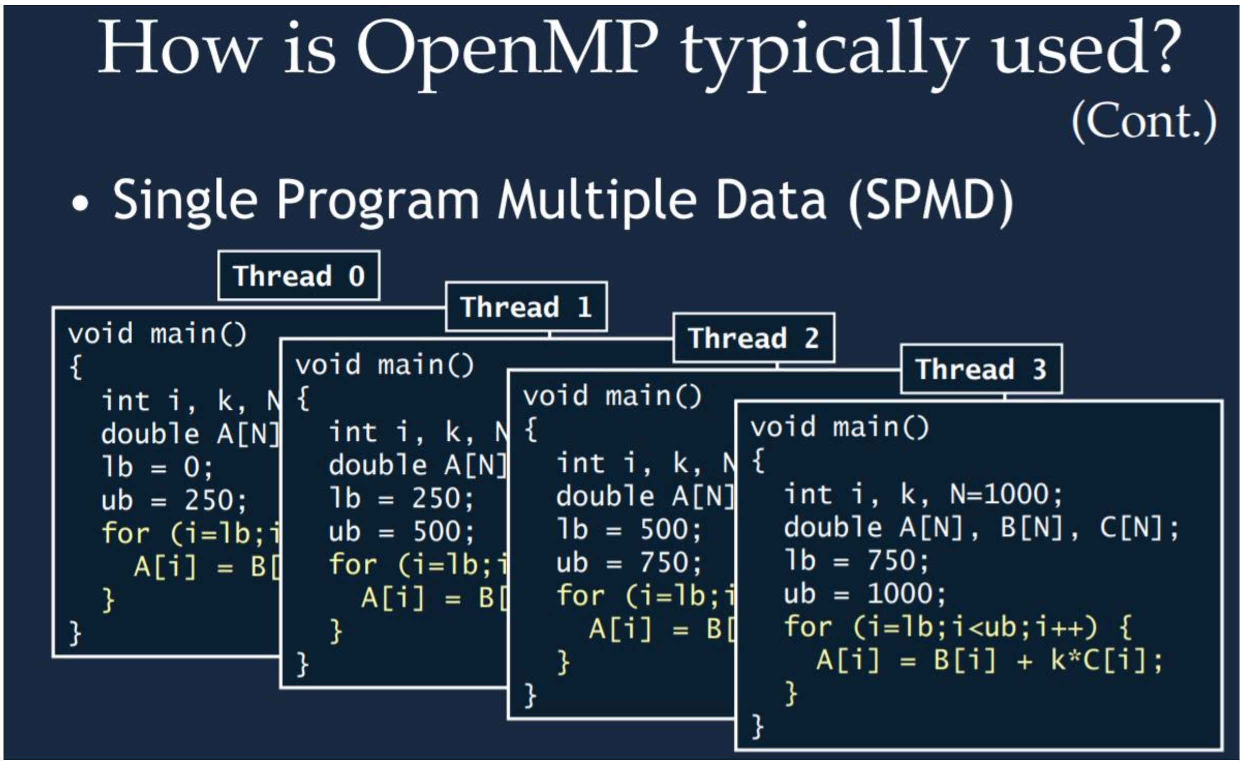
Sequential Program

```
void main()
{
  int i, k, N=1000;
  double A[N], B[N], C[N];
  for (i=0; i<N; i++) {
    A[i] = B[i] + k*C[i]
  }
}
```

$\rightarrow$

Parallel Program

```
#include "omp.h"
void main()
{
  int i, k, N=1000;
  double A[N], B[N], C[N];
#pragma omp parallel for
  for (i=0; i<N; i++) {
    A[i] = B[i] + k*C[i];
  }
}
```



How is OpenMP typically used?
(Cont.)

- Single Program Multiple Data (SPMD)

Thread 0

```
void main()
{
  int i, k, N
  double A[N]
  lb = 0;
  ub = 250;
  for (i=lb;i
    A[i] = B[
  }
}
```

Thread 1

```
void main()
{
  int i, k, N
  double A[N]
  lb = 250;
  ub = 500;
  for (i=lb;i
    A[i] = B[
  }
}
```

Thread 2

```
void main()
{
  int i, k, N
  double A[N]
  lb = 500;
  ub = 750;
  for (i=lb;i
    A[i] = B[
  }
}
```

Thread 3

```
void main()
{
  int i, k, N=1000;
  double A[N], B[N], C[N];
  lb = 750;
  ub = 1000;
  for (i=lb;i<ub;i++) {
    A[i] = B[i] + k*C[i];
  }
}
```