

**Lab #2, Thread Creation**

**Date: 5th May 2022**

Course Code: **CSE325**

Course Title:

**Operating System**

Section No: **01**

Semester: **Spring2022**

**Submitted By**

**Md. Redwan Ahmed**

**2019-1-60-249**

**Submitted To**

**Dr. Md. Nawab Yousuf Ali**

**Professor**

**Department of Computer Science & Engineering**

**Abstract**

Threads are not independent of each other because processes allow threads to share their code sections, data sections, and OS resources such as open files and signals with other threads. But, like the process, a thread has its own program counter (PC), a register set and a stack space.

**Introduction**

Threads are a popular way to improve applications through parallelism. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads, one thread to format text, another thread to process input, and so on.

Threads work faster than processes for the following reasons:

1) Threads made much faster.

2) Changing the context between threads is much faster.

3) The thread can be closed easily

4) Quick communication between threads.

**Experimental Procedure**

**Problem 1:**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

pthread\_mutex\_t mutexFuel;

pthread\_cond\_t condFuel;

int fuel = 0;

void\* fuel\_filling(void\* arg) {

for (int i = 0; i < 5; i++) {

pthread\_mutex\_lock(&mutexFuel);

fuel += 15;

printf("Filled fuel... %d\n", fuel);

pthread\_mutex\_unlock(&mutexFuel);

pthread\_cond\_signal(&condFuel);

sleep(1);

}

}

void\* car(void\* arg) {

pthread\_mutex\_lock(&mutexFuel);

while (fuel < 40) {

printf("No fuel. Waiting...\n");

pthread\_cond\_wait(&condFuel, &mutexFuel);

// Equivalent to:

// pthread\_mutex\_unlock(&mutexFuel);

// wait for signal on condFuel

// pthread\_mutex\_lock(&mutexFuel);

}

fuel -= 40;

printf("Got fuel. Now left: %d\n", fuel);

pthread\_mutex\_unlock(&mutexFuel);

}

int main(int argc, char\* argv[]) {

pthread\_t th[2];

pthread\_mutex\_init(&mutexFuel, NULL);

pthread\_cond\_init(&condFuel, NULL);

for (int i = 0; i < 2; i++) {

if (i == 1) {

if (pthread\_create(&th[i], NULL, &fuel\_filling, NULL) != 0) {

perror("Failed to create thread");

}

} else {

if (pthread\_create(&th[i], NULL, &car, NULL) != 0) {

perror("Failed to create thread");

}

}

}

for (int i = 0; i < 2; i++) {

if (pthread\_join(th[i], NULL) != 0) {

perror("Failed to join thread");

}

}

pthread\_mutex\_destroy(&mutexFuel);

pthread\_cond\_destroy(&condFuel);

return 0;

}

Problem 2:

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

void\* thread\_F() {

printf("This is the starting thread\n");

sleep(3);

printf("Bye, this is the ending thread\n");

}

int main(int argc, char\* argv[]) {

pthread\_t p1, p2;

if (pthread\_create(&p1, NULL, &thread\_F, NULL) != 0) {

return 1;

}

if (pthread\_create(&p2, NULL, &thread\_F, NULL) != 0) {

return 2;

}

if (pthread\_join(p1, NULL) != 0) {

return 3;

}

if (pthread\_join(p2, NULL) != 0) {

return 4;

}

return 0;

}

**Problem 3:**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <time.h>

// chefs = threads

// stove = shared data (+mutex)

pthread\_mutex\_t stoveMutex[4];

int stoveFuel[4] = { 100, 100, 100, 100 };

void\* routine(void\* args) {

for (int i = 0; i < 4; i++) {

if (pthread\_mutex\_trylock(&stoveMutex[i]) == 0) {

int fuelNeeded = (rand() % 30);

if (stoveFuel[i] - fuelNeeded < 0) {

printf("No more fuel... going home\n");

} else {

stoveFuel[i] -= fuelNeeded;

usleep(500000);

printf("Fuel left %d\n", stoveFuel[i]);

}

pthread\_mutex\_unlock(&stoveMutex[i]);

break;

} else {

if (i == 3) {

printf("No stove available yet, waiting...\n");

usleep(300000);

i = 0;

}

}

}

}

int main(int argc, char\* argv[]) {

srand(time(NULL));

pthread\_t th[10];

for (int i = 0; i < 4; i++) {

pthread\_mutex\_init(&stoveMutex[i], NULL);

}

for (int i = 0; i < 10; i++) {

if (pthread\_create(&th[i], NULL, &routine, NULL) != 0) {

perror("Failed to create thread");

}

}

for (int i = 0; i < 10; i++) {

if (pthread\_join(th[i], NULL) != 0) {

perror("Failed to join thread");

}

}

for (int i = 0; i < 4; i++) {

pthread\_mutex\_destroy(&stoveMutex[i]);

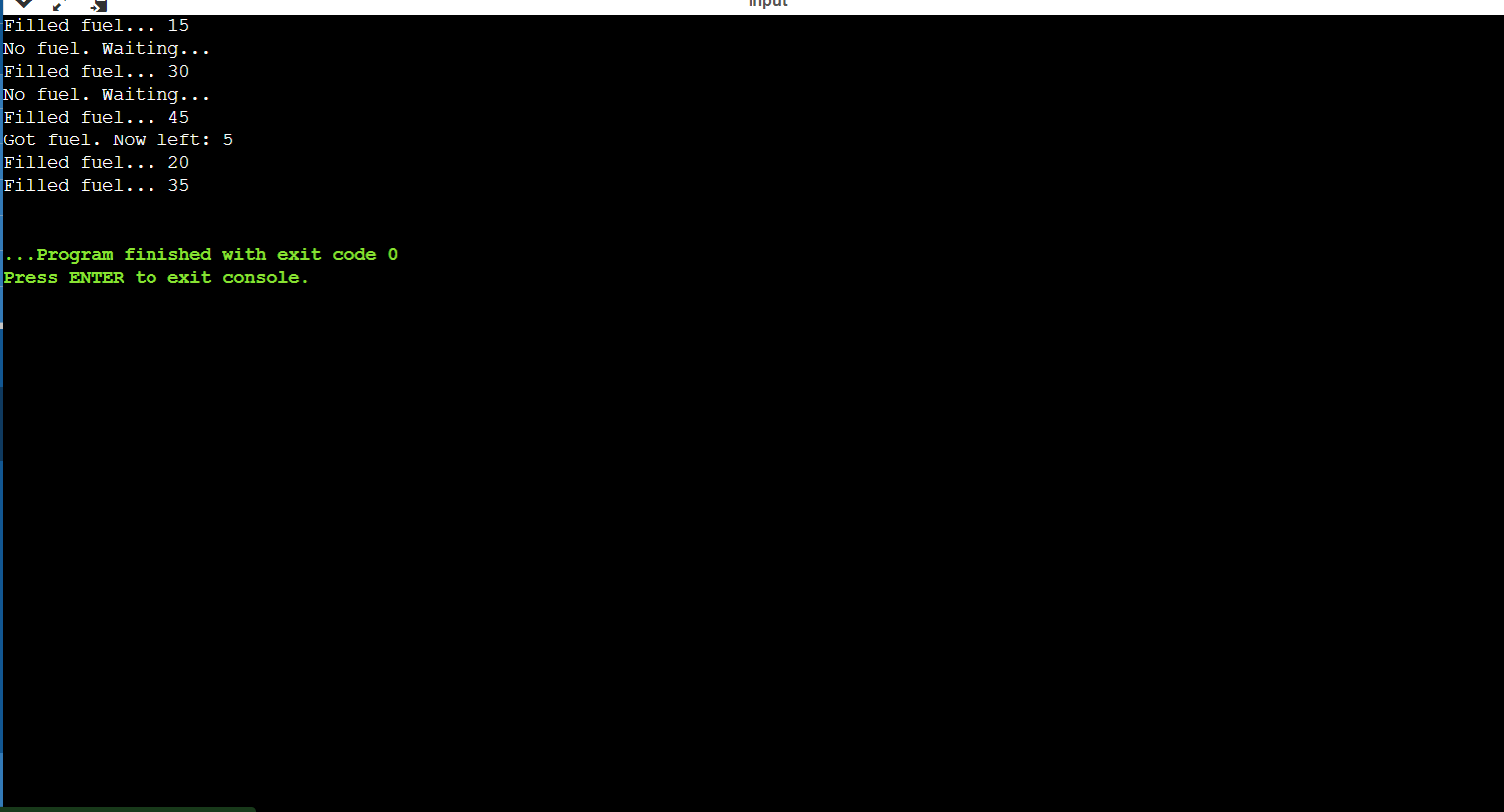
}

return 0;

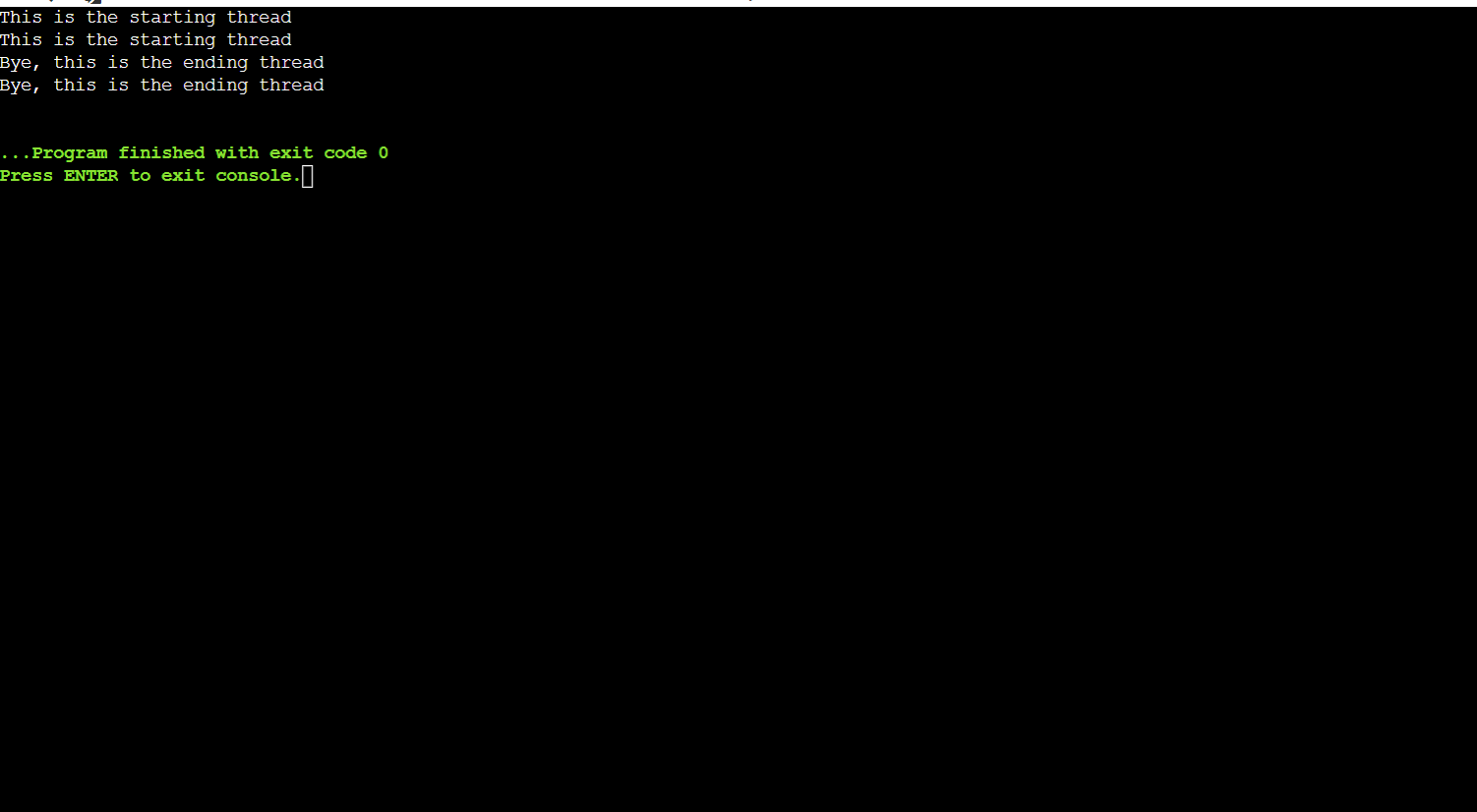
}

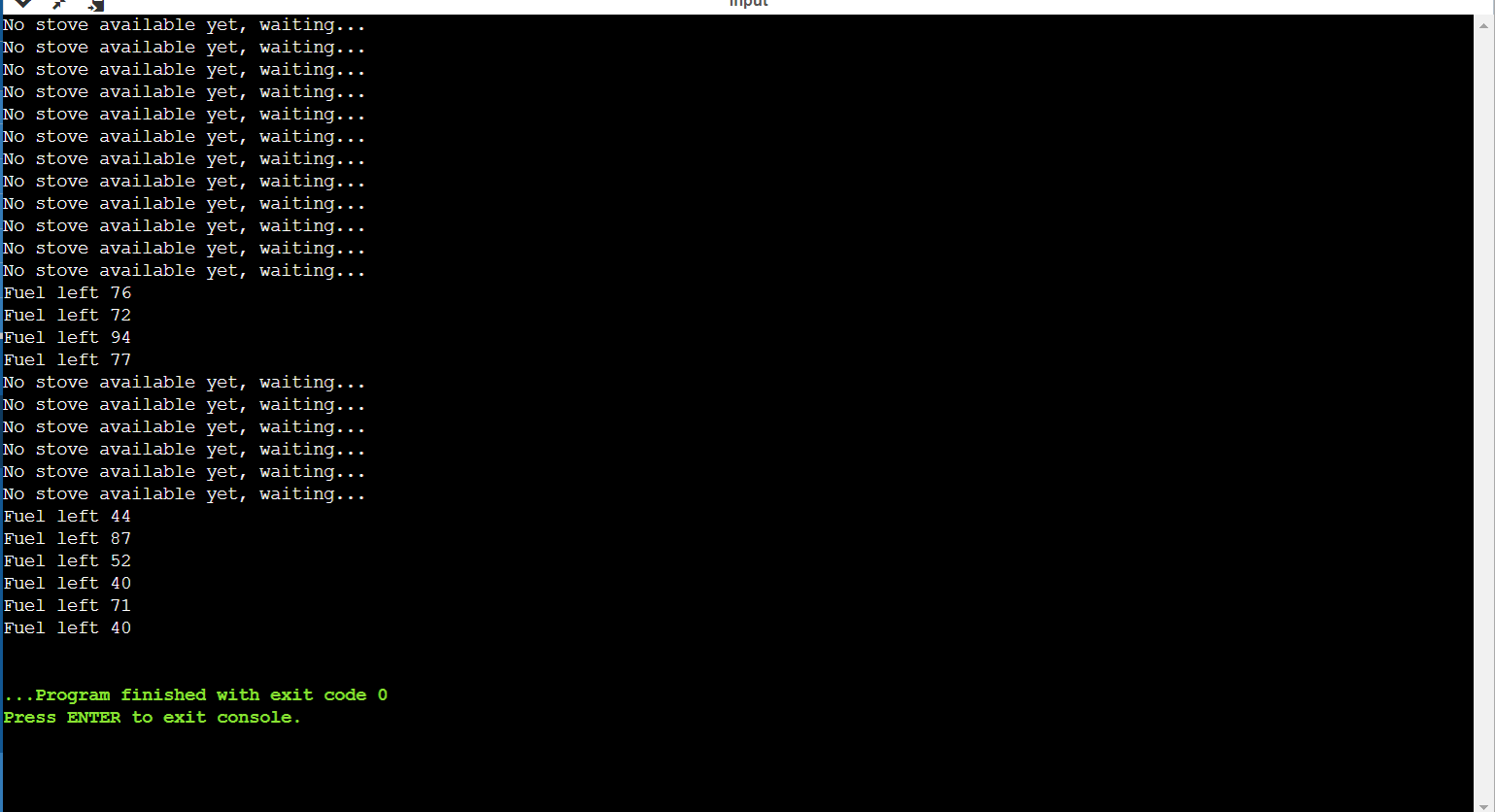
**Results**

**Output 1**



**Output 2:**



**Output 3:**

**Discussion**

Threading is a specialized form of multitasking and a multitasking is the feature that allows your computer to run two or more programs concurrently. In general, there are two types of multitasking: process-based and thread-based.

Process-based multitasking handles the concurrent execution of programs. Thread-based multitasking deals with the concurrent execution of pieces of the same program.

A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

C does not contain any built-in support for multithreaded applications. Instead, it relies entirely upon the operating system to provide this feature.

**Conclusion**

If you are working on Linux OS and we are going to write multi-threaded C program using POSIX. POSIX Threads, or Pthreads provides API which are available on many Unix-like POSIX systems such as FreeBSD, NetBSD, GNU/Linux, Mac OS X and Solaris.