Program A: Creates 2 threads that print the numbers from 0 to 9

#include <stdio.h>

#include <pthread.h>

void\* fa(void\* a) {

int i;

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

printf("A: %d\n", i);

}

return NULL;

}

void\* fb(void\* a) {

int i;

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

printf("B: %d\n", i);

}

return NULL;

}

int main(int arg, char\*\* argv) {

int i;

/\*delcaring the threads\*/

pthread\_t ta, tb;

/\*creating the threads\*/

pthread\_create(&ta, NULL, fa, NULL);

/\* &ta : the handle

\* NULL: a pointer to a structure of attributes which control the thread's execution - the default value

\* fa : a pointer to the function fa (just fa, we don't call the function)

\* NULL: the argument (this is actually the argument given to the function fa

\*/

pthread\_create(&tb, NULL, fb, NULL);

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

printf("M: %d\n", i);

}

/\*waiting for the thread\*/

pthread\_join(ta, NULL);

/\* ta : the handle

\* NULL : the value returned by the thread

\*/

pthread\_join(tb, NULL);

return 0;

}

Program B: Creates T(5 in this case) threads that print the numbers from 0 to N(10 in this case) and for each one it also shows the “id” of the thread before printing the number

#include <stdio.h>

#include <pthread.h>

#define T 5

#define N 10

void\* f(void\* a) {

int i;

for(i = 0; i < N; i++) {

printf("%lu: %d\n", pthread\_self(), i);

/\*pthread\_self() : similar to the PID of the procees, each thread has a unique one and it’s of type long\*/

}

return NULL;

}

int main(int arg, char\*\* argv) {

int i;

/\*delcaring the threads\*/

pthread\_t t[T];

/\*creating the threads\*/

for(i = 0; i < T; i++) {

pthread\_create(&t[i], NULL, f, NULL);

/\* &t[i] : the handle

\* NULL: a pointer to a structure of attributes which control the thread's execution - the default value

\* f : a pointer to the function fa (just fa, we don't call the function)

\* NULL: the argument (this is actually the argument given to the function fa

\*/

}

/\*waiting for the thread\*/

for(i = 0; i < T; i++) {

pthread\_join(t[i], NULL);

/\* t[i] : the handle

\* NULL : the value returned by the thread

\*/

}

return 0;

}

Program C: Creates T(5 in this case) threads that print the numbers from 0 to N(10 in this case) and for each one it also shows the number associated to it before printing the number – if we would’ve just send to the thread “I” and use it there, basically all threads would get the same number

Like this :

In main : pthread\_create(&t[i], NULL, f, (void\*)&i);

In f: printf("%d: %d\n", \*(int\*)a, i);

So we can create an array of integers which stores for each thread a copy of “I” so that everything runs as expected :

#include <stdio.h>

#include <pthread.h>

#define T 5

#define N 10

void\* f(void\* a) {

int i;

for(i = 0; i < N; i++) {

printf("%d: %d\n", \*(int\*)a, i);

/\*pthread\_self() : similar to the PID of the procees, each thread has a unique one\*/

}

return NULL;

}

int main(int arg, char\*\* argv) {

int i, a[T];

/\*delcaring the threads\*/

pthread\_t t[T];

/\*creating the threads\*/

for(i = 0; i < T; i++) {

a[i] = i;

pthread\_create(&t[i], NULL, f, (void\*)&a[i]);

}

/\*waiting for the thread\*/

for(i = 0; i < T; i++) {

pthread\_join(t[i], NULL);

}

return 0;

}

Or an int pointer that takes the value of "I" for every iteration and is passed as parameter to the function f:

int\* a;

for(i = 0; i < T; i++) {

a = (int\*)malloc(sizeof(int));

\*a = i;

pthread\_create(&t[i], NULL, f, (void\*)a);

}

Program D: using mutex to lock-unlock so that only one thread at a time gets to perform the operation

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#define T 5

#define N 10

int n = 0;

/\*creating the mutex\*/

pthread\_mutex\_t m;

void\* f(void\* a) {

int i;

for(i = 0; i < N; i++) {

pthread\_mutex\_lock(&m);

n++;

pthread\_mutex\_unlock(&m);

}

return NULL;

}

void\* g(void\* a) {

pthread\_mutex\_lock(&m);

n+= N;

pthread\_mutex\_unlock(&m);

return NULL;

}

int main(int arg, char\*\* argv) {

long long i;

/\*delcaring the threads\*/

pthread\_t t[T];

/\*initializing the mutex\*/

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

/\*creating the threads\*/

for(i = 0; i < T; i++) {

pthread\_create(&t[i], NULL, g, (void\*)i);

}

/\*waiting for the thread\*/

for(i = 0; i < T; i++) {

pthread\_join(t[i], NULL);

}

/\*destroying the mutex\*/

pthread\_mutex\_destroy(&m);

printf("%d\n", n);

return 0;

}