Let's Thread

CSL301 - Operating Systems

Class Assignment - 7

Key Thread Functions and Types

- `#include <pthread.h>` – Required for thread functions.
- `pthread_t` – Type for thread handles.
- `pthread_create()` – Starts a new thread.
- `pthread_join()` – Waits for a thread to finish.

pthread_create()

Function Syntax:

```
int pthread_create(pthread_t *thread, const pthread_attr_t
    *attr, void *(*start_routine)(void *),  void *arg);
```

Argument Purpose:

- `thread`: Pointer to variable where thread ID will be stored.
- `attr`: Thread attributes. Usually NULL (default).
- `start_routine`: Function thread will execute (must take void* argument, return void*).
- `arg`: Pointer to data passed to thread function. (Can take any data type, pointer, struct variable etc.)

pthread_join()

Function Syntax:

```c
    int pthread_join(pthread_t thread, void **retva);
```

Usage:

- Use in main to wait until thread finishes.
- `retval` can collect the returned value from thread function, if needed.

How to Create a Thread (C/pthreads)

Example:

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

void* myfunc(void* arg) {
  printf("Hello from thread!\n");
  return NULL;
}

int main() {
  pthread_t tid;
  pthread_create(&tid, NULL, myfunc, NULL);
  pthread_join(tid, NULL);
  printf("Thread finished!\n");
  return 0;
}
```

Passing Data to Threads

- The thread function always takes a `void*` argument.
- You can pass any pointer (e.g., to an int, struct, array) when creating the thread.

Syntax Example:

```
void* print_id(void* arg) {
  int id = *(int*)arg;
  printf("Thread ID: %d\n", id);
  return NULL;
}

int main() {
  pthread_t tid;
  int myid = 42;
  pthread_create(&tid, NULL, print_id, &myid);
  pthread_join(tid, NULL);
}
```

Question 1

Task: Write a program to create 2 threads and define two int as id1 and id2 for them. Define one global variable and one local variable in main and one local variable in thread function. Print the addresses of the global variable and the local variables in the main function and in both threads. Observe and explain which addresses are the same and which are different.

Helper Code

```
void* thread_func(void* arg) {
    int thread_local;
    printf("Thread %d: local address = %p, global address
        = %p\n",
    *(int*)arg, (void*)&thread_local, (void*)&global_var);
    return NULL;
 }


 printf("Main: local address = %p, global address = %p\n"
     ,(void*)&main_local, (void*)&global_var);
```

Question 2

Task: Write a program that creates 10 threads. Each thread prints its thread number (for example, "Thread 1 running"). The main process should wait for all threads to finish using pthread_join, then print "All threads completed."

- Does the order of thread messages (e.g., “Thread 1 running”, “Thread 2 running”, etc.) always stay the same on each run?
- Why might the order vary between runs?

Hint: Use a loop and an integer array to give each thread its own number argument.

```
  for (int i = 0; i < N; i++) {
    pthread_join(threads[i], NULL); }
```

Question 3

Task: Write a program to create 10 threads. Each thread should increment a `counter` global variable 1000,000 times (no synchronization or locking).

Question:

- Why does the final value differ from the expected result?
- What is a *race condition*, and how does it affect your program?

Hint: Try running your program several times. Observe how the result changes. Think about what happens when two threads try to update the counter at the same time.

Question 4

Task: Write a C program that creates a thread, it receives a number as input. The thread computes the square of this number and returns the result to the main function.

```
    void* result;
    pthread_join(tid, &result);
```

Syntax:

```c
void* compute_square(void* arg) {
  int number = *(int*)arg;
  int* result = malloc(sizeof(int));
  *result = number * number;
  return result; // returned pointer must be freed
}
```

Use `pthread_join` to collect results into the main function.

Question 5

Task:

- Write a C program that creates N threads (e.g., N = 5).
- Each thread receives its thread number as input.
- Each thread computes the square of its number.
- Each thread returns the result to the main function.
- The main function waits for all threads, collects each result, prints results, and prints the sum of all returned values.

Example Output:

- Thread 1 returned 1
- Thread 2 returned 4
- Thread 3 returned 9
- Thread 4 returned 16
- Thread 5 returned 25
- Sum of all thread results: 55