Operating system concepts

Problems of Multi-threading
Slides Set #9

By Prof K R Chowdhary

JNV University

2023

Threads with shared data

Problem statement: We would like to declare a global variable "int counter =0;" and create two threads "A" and "B".

- Each thread runs in its own way (asynchronous threads) and tries to increment the "counter", through a loop variable i=0 to "< 100,000".
- There is no limit on the value of "counter" variable.
- Since two counters try to increment the counter by 100,000, the counter should become finally 200,000. But it does not!
- Why so?

Threads with shared data

```
/* thrd_sync.c */
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
static volatile int counter =0;
// mythread()
//add 1 to counter repeatedly, in a loop
// to add 100000 to counter, then it
                          shows the problem.
void *mythread(void *arg){
  printf("Thread %s: begins\n", (char *)arg);
  int i;
  for(i=0; i< 100000; i++) counter++;
  printf("Thread %s: ends\n", (char *)arg);
  return NULL;
}
```

Threads with shared data

```
//Just launch two threads
int main(){
   pthread_t p1, p2;
   printf("main: begin (counter = %d)\n", counter);
   pthread_create(&p1, NULL, mythread, "A");
   pthread_create(&p2, NULL, mythread, "B");

//join waits for the threads to finish
  pthread_join(p1, NULL);
  pthread_join(p2, NULL);
  printf("main: done with both counter = %d\n", counter);
  return 0;
}
```

Run 1: Threads with shared data

- What do we expect? Two threads, each increments counter by 100000, so $2X100000$ (?)

```
\$  gcc -o main main.c
\$ ./main
main: begin (counter = 0)
Thread A: begins
Thread B: begins
Thread B: ends
Thread A: ends
main: done with both counter = 168137
```

- Questions:
  - What are the global variables here?
  - The sum of two for loops, each 1-100000, is 2,00,000. But counter did not reach to 2,00,000 (???)

Run 2 (with same compiled file): Threads with "shared data"

- What do we expect? Two threads, each increments counter by 100000, so $2X100000$ total (?)

```
\$  gcc -o main main.c
\$ ./main
main: begin (counter = 0)
Thread A: begins
Thread B: begins
Thread A: ends
Thread B: ends
main: done with both counter = 134004
```

- The sum of two for loops, each 1-100000, is 2,00,000. But counter did not reach to 2,00,000 (???)
- It is **race** condition. Why this name?

Race conditions and synchronization

- What just happened is called a *race condition*
  - *Concurrent execution* can lead to different results
- *Critical section*: portion of code that can lead to race conditions
- What we need: *mutual exclusion*
  - Only one thread should be executing *critical section* at any time
- What we need: *atomicity* of the critical section
  - The critical section should execute like one uninterruptible (unbreakable) instruction
- That is: undivided "fetch + execute + store" is continuous for one instruction.
- How is it achieved? Locks (topic of next lecture)

Race conditions and synchronization...

- Questions:
  - Why the race condition occurs?
  - How the critical section can stop race condition?
  - What is mutual exclusion?
  - What is atomicity of an instruction?
  - Will the following assembly code provide atomicity, where 2000 is address of a global variable?

```
LXI H, 2000
MOV A, M
INR A
MOVE M, A
; This code is running in two threads
```