Multiprocessing: threads (continued)

COSC 208, Introduction to Computer Systems, 2021-11-17

Announcements

- Exam 3
  - Take-home: due at 11pm on ?
- Project 3 due Thursday, December 2

Outline

- Warm-up
- pthreads API
- Race conditions

Warm-up

Q1: *What are all possible outputs produced by this program?*

```
void *thread1_main(void *arg) {
    int *x = (int *)arg;
    *x += 1;
    printf("x is %d\n", *x);
    return NULL;
}
void *thread2_main(void *arg) {
    int *y = (int *)arg;
    *y -= 1;
    printf("y is %d\n", *y);
    return NULL;
}
int main() {
    int *z = malloc;
    *z = 0;
    // Start thread running thread1_main(z)
    // Start thread running thread2_main(z)
    // Wait for threads to finish
    printf("z is %d\n", *z);
}
```

```
x is 1
y is 0
z is 0
```

OR

```
y is -1
x is 0
z is 0
```

Pthreads API

- Can create and wait for threads to finish, just like processes, but API is different
- Use the pthreads library—`#include <pthread.h>`
- `int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void * (*start_routine)(void*), void * arg)`
    - `thread`—a struct that stores metadata for the thread
    - `attr`—configuration settings for the thread
    - `start_routine`—the function to start executing when the thread starts
        - Pass a pointer to a function
    - `arg`—an argument passed to the aforementioned function
    - *How do we create a new process?*—`fork`
- `int pthread_join(pthread_t thread, void **value_ptr)`
    - `thread`—the same struct passed at thread creation; used to identify the thread we want to wait for
    - `value_ptr`—the location where the function return value should be stored
        - Notice it's a pointer to a void pointer and the `start_routine` function specified in create returns a void pointer
    - *How do we wait for a process to finish?*—`wait` or `waitpid`
- Q2: *What are all possible outputs produced by this program?*

```c
#include <pthread.h>
void *printer(void *arg) {
    char *ch = (char*)arg;
    printf("I am %c\n", *ch);
    return NULL;
}
int main() {
    pthread_t thread1, thread2;
    char *ch1 = malloc(sizeof(char));
    *ch1 = 'X';
    char *ch2 = malloc(sizeof(char));
    *ch2 = 'Y';
    pthread_create(&thread1, NULL, &printer, ch1);
    pthread_create(&thread2, NULL, &printer, ch2);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
}
```

```
I am X
I am Y
```

OR

```
I am Y
I am X
```

- Q3: *What are all possible outputs produced by this program?*

```c
#include <pthread.h>
void *printer(void *arg) {
    char *ch = (char*)arg;
    printf("I am %c\n", *ch);
    return NULL;
}
int main() {
    pthread_t thread1, thread2;
    char *ch = malloc(sizeof(char));
    *ch = 'A';
    pthread_create(&thread1, NULL, &printer, ch);
    *ch = 'B';
    pthread_create(&thread2, NULL, &printer, ch);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
}
```

```
I am A
I am B
```

OR

```
I am B
I am B
```

- Q4: *What are all possible outputs produced by this program?*

```c
#include <pthread.h>
void *printer(void *arg) {
    char *ch = (char*)arg;
    printf("I am %c\n", *ch);
    return NULL;
}
int main() {
    pthread_t thread1, thread2;
    char *ch = malloc(sizeof(char));
    *ch = 'P';
    pthread_create(&thread1, NULL, &printer, ch);
    pthread_join(thread1, NULL);
    *ch = 'Q';
    pthread_create(&thread2, NULL, &printer, ch);
    pthread_join(thread2, NULL);
}
```

```
I am P
I am Q
```

Race conditions

- Example program

```
#include <pthread.h>
#include <stdio.h>
void *deposit(void *arg) {
    int *balance = (int *arg)
    int tmp = *balance
    tmp += 100;
    *balance = tmp;
    return NULL;
}
void *withdraw(void *arg) {
    int *balance = (int *arg)
    int tmp = *(int *)balance
    tmp -= 50;
    *balance = tmp;
    return NULL;
}
int main() {
    pthread_t thrA, thrB;
    int *balance = malloc(sizeof(int));
    *balance = 250;
    pthread_create(&thrA, NULL, &deposit, balance);
    pthread_create(&thrB, NULL, &withdraw, balance);
    pthread_join(thrB, NULL);
    pthread_join(thrA, NULL);
    printf("Balance: $%d\n", *balance);
}
```

- Possible interleaving of threads
  - Balance: $300

thrA	thrB
int tmp = *balance	
tmp += 100	
*balance = tmp	
	int tmp = *balance
	tmp -= 50
	*balance = tmp

  - Balance: $300

thrA	thrB
	int tmp = *balance
	tmp -= 50
	*balance = tmp
int tmp = *balance	
tmp += 100	
*balance = tmp	

  - Balance: $200

thrA	thrB
	`int tmp = *balance`
	`tmp -= 50`
`int tmp = *balance`	
`tmp += 100`	
`*balance = tmp`	
	`*balance = tmp`

- `Balance: $350`

thrA	thrB
`int tmp = *balance`	
`tmp += 100`	
	`int tmp = *balance`
	`tmp -= 50`
	`*balance = tmp`
`*balance = tmp`	

- Takeaway: be careful with shared memory!

Extra practice

Q5: *What are all possible outputs produced by this program?*

```c
#include <stdio.h>
#include <pthread.h>
void *printer2(void *arg) {
    char *ch = (char*)arg;
    printf("Start %c\n", *ch);
    printf("End %c\n", *ch);
    return NULL;
}
int main() {
    pthread_t thread1, thread2;
    char ch1='X', ch2='Y';
    pthread_create(&thread1, NULL, &printer2, &ch1);
    pthread_create(&thread2, NULL, &printer2, &ch2);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
}
```

```
Start X
End X
Start Y
End Y
```

OR

```
Start Y
End Y
Start X
End X
```

OR

```
Start X
Start Y
End Y
End X
```

OR

```
Start X
Start Y
End X
End Y
```

OR

```
Start Y
Start X
End X
End Y
```

OR

```
Start Y
Start X
End Y
End X
```