Multiprocessing: threads

COSC 208, Introduction to Computer Systems, 2022-04-28

Announcements

- Project 4 due Thursday, May 5

Outline

- Threads

Warm-up

- Q0: *What are all possible outputs produced by this program?*

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
int main() {
    printf("A\n");
    int x = fork();
    if (x == 0) {
        int y = fork();
        if (y == 0) {
            printf("B\n");
        }
        else {
            wait(NULL);
            printf("C\n");
        }
    }
    else {
        wait(NULL);
        printf("D\n");
    }
    printf("E\n");
}
```

```
A
B
E
C
E
D
E
```

Threads

- Threads are multiple execution contexts within the **same process**
  - Processes are multiple execution contexts within the **same machine**
- Because threads are within the same process, they share all of the process's resources—memory, CPU time, file descriptors (i.e., open files), etc.
- Consequently, two threads can update the same variable

```
void *thread1_main(void *arg) {
    int *x = (int *)arg;
    *x += 1;
    return NULL;
}
void *thread2_main(void *arg) {
    int *y = (int *)arg;
    *y += 2;
    return NULL;
}
int main() {
    int *z = malloc(sieof(int));
    *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);
}
```

```
z is 3
```

  - Two processes cannot update the same variable—memory is not shared; must use inter-process communication mechanism to share information

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

```c
void *thread_main(void *arg) {
    char *id = (char *)arg;
    printf("I am thread %c\n", *id);
    return NULL;
}
int main() {
    char a = 'A';
    char b = 'B';
    // Start thread running thread_main(&a)
    // Start thread running thread_main(&b)
    // Wait for threads to finish
}
```

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

OR

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

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 = '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
```

Extra Practice

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

```
void *proc1_main(void *arg) {
    int *x = (int *)arg;
    *x += 1;
    return NULL;
}
void *proc2_main(void *arg) {
    int *y = (int *)arg;
    *y += 2;
    return NULL;
}
int main() {
    int z = 0;
    int pid = fork();
    if (pid == 0) {
        proc1_main(&z);
    } else {
        proc2_main(&z);
        wait(NULL);
    }
    printf("z is %d\n", z);
}
```

```
z is 1
z is 2
```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main() {
    int pid = fork();
    if (pid == 0) {
        printf("Child\n");
        exit(22);
    } else {
        int status = 0;
        wait(&status);
        printf("Status %d\n", WEXITSTATUS(status));
        exit(44);
    }
}
```

```
Child
Status 22
```