Multiprocessing: threads

COSC 208, Introduction to Computer Systems, 2021-11-15

Announcements

- Project 3 due Thursday, December 2

Outline

- Warm-up
- Threads

Warm-up

- Q1: *Consider the following processes:*

Process	Arrival time	Duration
A	Just before 0	60
B	Just before 5	15
C	Just before 10	15

- *Determine the schedule for the above processes using a Round Robin (RR) scheduler a time quantum of 10.*

```
┌───────┬───────┬───────┬───────┬────┬────┬────────────────────────────────┐
│ A     │ B     │ C     │ A     │ B  │ C  │ A                              │
└───────┴───────┴───────┴───────┴────┴────┴────────────────────────────────┘
0       10      20      30      40   45   50                              90
```

  - Average Turnaround = (90 + 40 + 40) / 3 = 56.6
  - Average Response = (0 + 5 + 10) / 3 = 5
  - Average Wait = (30 + 25 + 25) / 3 = 26.6

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

- Q2: *What are all possible outputs produced by this program?*

```
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
```

- Q3: *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
```

Extra practice

- Q4: *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
```