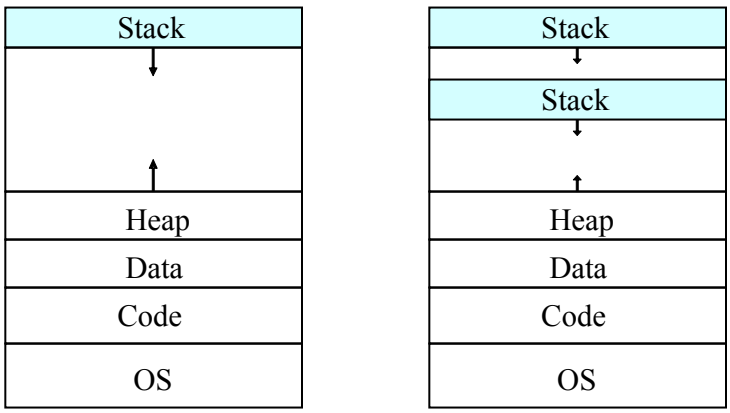
Introduction to Threads

Race Conditions

Process Address Space Revisited



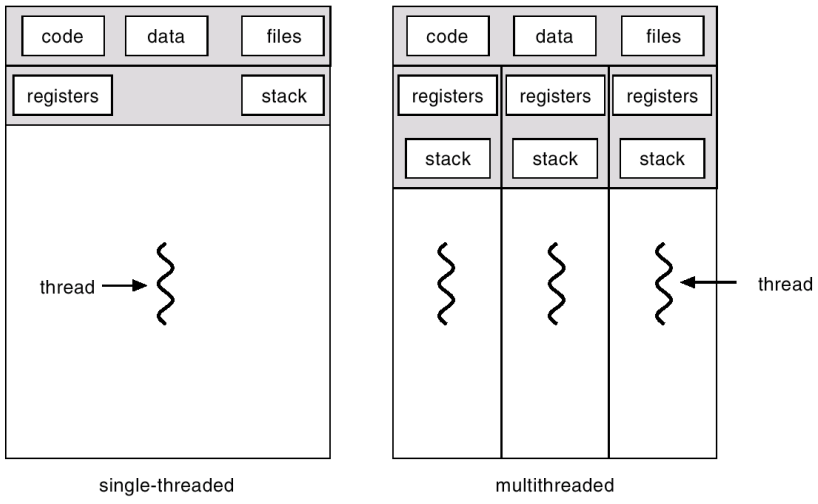
(a) Process with Single Thread

(b) Process with Two Threads

What are Threads?

- A thread is an independent stream of instructions
  - Basic unit of CPU utilization
- A thread has its own
  - thread ID
  - execution stack
- A thread shares with its sibling threads
  - The code, data and heap section
  - Other OS resources, such as open files and signals

Single and Multi-Threaded Processes



Multi-Threaded Processes

- Each thread has a private stack
- But threads share the process address space!
- There is no memory protection!
- Threads could potentially write into each other's stack

Why use Threads?

- A specific example, a Web server:

```
in an infinite loop
{       get web page request from client
        check if page exists and client has permissions
        transmit web page back to client
}
```

- If transmission takes very long time, the server is unable to answer other client's requests. Solution:

```
in an infinite loop
{       get web page request from client
        check if page exists and client has permissions
        create a thread to transmit web page back to client
}
```

pthreads (5.4)

- Refers to the POSIX standard (IEEE 1003.1c)
- API for thread creation and synchronization
- Common in UNIX operating systems

Java Threads (5.8)

- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface
- JVM manages Java threads
  - Creation
  - Execution
  - Etc.

Race Conditions for Threads

Concurrent Threads

- Concurrent threads come into conflict with each other when they come to use shared resources
- Atomic actions are indivisible. In hardware, loads and stores are indivisible.
- On a processor, a thread switch can occur between any two atomic actions; thus the atomic actions of concurrent threads may be interleaved in any possible order.
- Result of concurrent execution should not depend on the order in which atomic instructions are interleaved.

`badcnt.c`: An Incorrect Program

```
#define NITERS 100000000
```

```
unsigned int cnt = 0; /* shared */

int main() {
    pthread_t tid1, tid2;
    pthread_create(&tid1, NULL,
                   count, NULL);
    pthread_create(&tid2, NULL,
                   count, NULL);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);

    if (cnt != (unsigned)NITERS*2)
        printf("BOOM! cnt=%d\n",
               cnt);
    else
        printf("OK cnt=%d\n",
               cnt);
}
```

```
/* thread routine */
void * count(void *arg) {
    int i;
    for (i=0; i<NITERS; i++)
        cnt++;
    return NULL;
}
```

```
linux> ./badcnt
BOOM! cnt=198841183

linux> ./badcnt
BOOM! cnt=198261801

linux> ./badcnt
BOOM! cnt=198269672
```

`cnt` should be equal to 200,000,000. What went wrong?!

Critical Sections

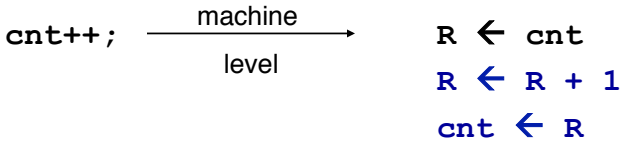
- Critical sections are blocks of code that access shared data.

```
/* thread routine */
void * count(void *arg) {
    int i;
    for (i=0; i<NITERS; i++)
        cnt++;
    return NULL;
}
```

- The objective is to make critical sections behave as atomic operations: if one process uses a shared piece of data, other processes can't access it.

Atomic Operations

- The statement



must be performed atomically (cnt is shared)

- An atomic operation is an operation that completes in its entirety, without interruption

Race Conditions

- One possible interleaving of statements is:

```
R1 = cnt
R1 = R1 + 1
<timer interrupt ! >
R2 = cnt
R2 = R2 + 1
in = R2
<timer interrupt !>
cnt = R1
```

Race condition:

The situation where several processes access shared data concurrently. The final value of the shared data may vary from one execution to the next.

- Then **cnt** ends up incremented once only! Threads overwrite each other's data.

Practice Exercise

- Consider two threads sharing a global variable count, initially 10:

Thread A	Thread B
`count++;`	`count--;`

- What are the possible values for count after both threads finish executing:

Practice Exercise

- Consider the following three concurrent threads that share a global variable g, initially 10:

Thread A	Thread B	Thread C
`g = g * 2;`	`g = g + 1;`	`g = g - 2;`

- What are the possible values for g, after all three threads finish executing?

Hands-on Session

- Complete the hands-on POSIX programming exercises posted on the class website.
  - Creating threads
  - Passing data to threads
  - **`pthread_create`**
  - **`pthread_exit`**
  - **`pthread_self`**
  - **`pthread_join`**
  - **`pthread_attach`**
  - **`pthread_detach`**