CS 3305A

Intro to Threads

Lecture 7

Sept 30th 2019

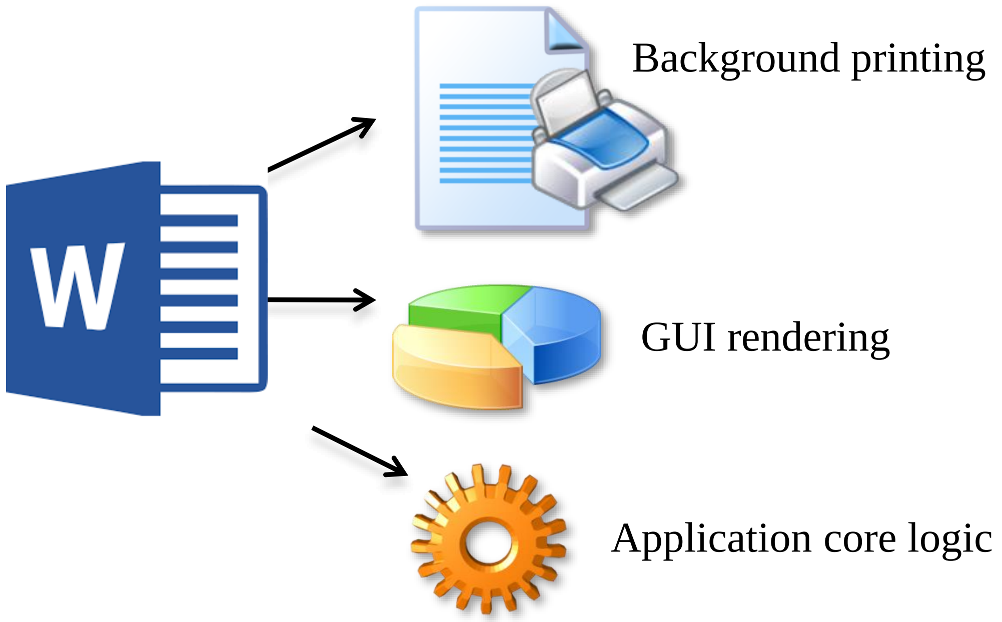
Introduction

- Multiple applications run concurrently!
- This means that there are multiple processes running on a computer

Introduction

- Applications often need to perform many tasks at once
- This requires multiple threads of execution

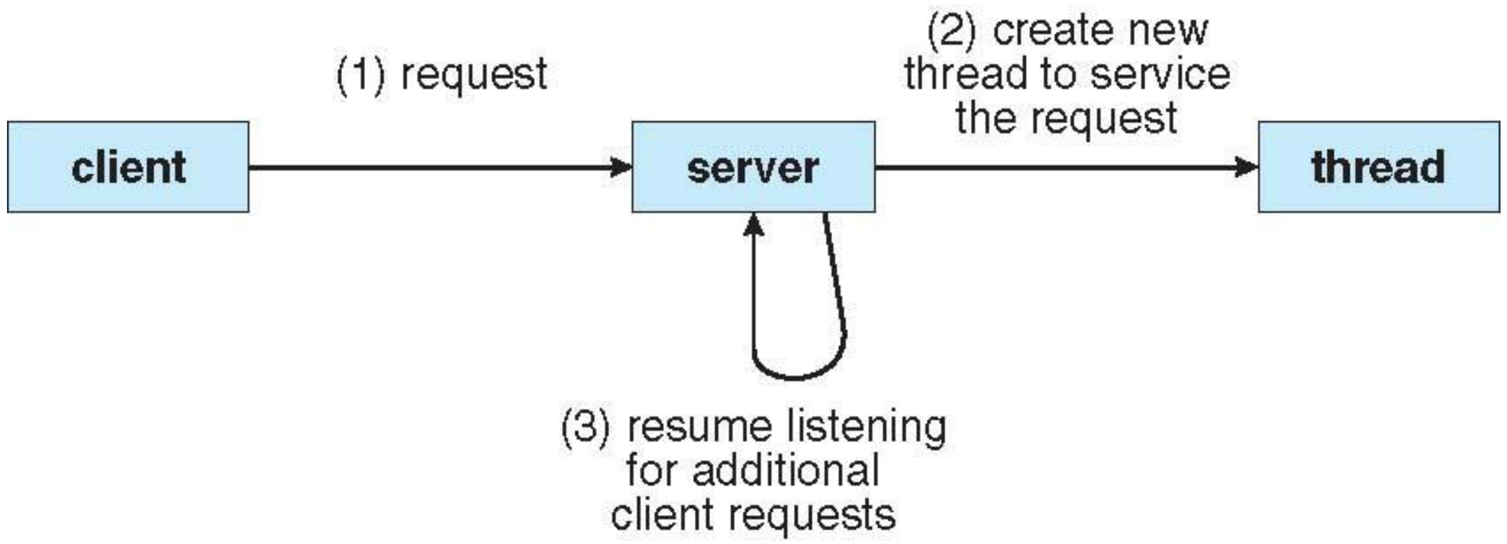
Example



- Example: Word processor
  - Tasks include:
    - Display graphics
    - Respond to keystrokes from the user
    - Perform spelling and grammar checking

Example

- Example: Web server
  - It is desirable to service requests concurrently


(1) request
client
server
(2) create new thread to service the request
thread
(3) resume listening for additional client requests

Introduction

- Earlier we discussed the use of forking to create a process
- For example we could
  - Word processor example: fork a process for each task
  - Web server example: fork a process for each request
- Not very efficient since a fork copies everything

Why Not Fork?

- You certainly can fork a new process
- In fact, the first implementation of Apache web servers (Apache 1.0) forked N processes when the web server was started
  - "N" was defined in a configuration file
  - Each child process handled one connection at a time
- Problem: Process creation is time consuming and resource intensive
- Creating threads is not as expensive. Why?

Thread State

- Threads share
  - Code
  - Data (global variables)
  - Open files, sockets
- Threads have their own CPU context
  - Program counter(PC), Stack pointer (SP), register state

Pthreads: POSIX Threads

- A thread library provides the programmer with an API for creating and managing threads

- Pthread Library (60+ functions)

- Programs must include the file `pthread.h`

Thread Creation

- Thread identifiers
  - Each thread has a unique identifier (ID), a thread can find out its ID by calling pthread_self().
  - Thread IDs are of type pthread_t which is usually an unsigned int.

pthread_create()

- Creates a new thread

```
int pthread_create (
            pthread_t *thread,
            pthread_attr_t *attr,
            void * (*start_routine),
            void *arg);
```

  - Returns 0 to indicate success, otherwise returns error code
  - **thread**: name of the new thread
  - **attr**: argument that specifies the attributes of the thread to be created (NULL = default attributes)
  - **start_routine**: function to use as the start of the new thread
  - **arg**: argument to pass to the new thread routine

pthread_create() example

Let us say that you want to create a thread that simply prints “hello world…I am a thread”

```
int main(int argc, char *argv) {

pthread_t worker_thread;

  if (pthread_create(&worker_thread, NULL,
                     do_work, NULL)) {
    printf(″Error while creating thread\n″);
    exit(1);
  }
  ...
}

void *do_work() {

Printf ("\n hello world..I am a thread");

 return NULL;
}
```

Problem

- Sharing global variables is dangerous - two threads may attempt to modify the same variable at the same time.
- Use support for mutual exclusion primitives that can be used to protect against this problem.
- The general idea is to lock something before accessing global variables and to unlock as soon as you are done.
- More on this topic later in the course