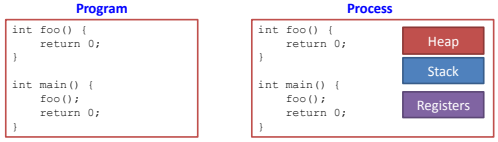
Threads

Programs and Processes

- A process is not a program
- Program: Static code and static data



- OS can host multiple processes of same program
  - E.g. many users can run 'ls' at the same time
- Once program can invoke multiple processes
  - E.g. make runs many processes to compile code
- No one-to-one mapping between programs and processes

Concurrency and Parallelism

- Many programs need to perform mostly independent tasks that do not need to be serialized
  - Web server: page requests
  - Text editor: auto-save, spell checking, text entry
  - Web client: tabbed browsing
- Concurrency
  - Multiple, generally different, tasks
  - For convenience
- Parallelism
  - Multiple copies of the same task
  - For performance
- This is one of the key problems we are facing today

How can we get this?

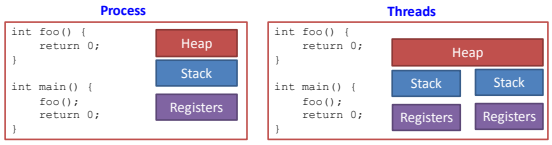
- Option 1: Fork a bunch of processes
  - Not very efficient
- A Process Requires
  - An address space
    - Code and data
  - Thread state
    - %EIP, %ESP, %EBP, other registers
  - OS resources
    - Open files, network connections, …

The thread model

- Traditional processes
  - 1 Process = 1 address space + **1** thread of execution
- Threads:
  - Separate address space from thread of execution
  - 1 process = 1 address space + **N** threads of execution

Threads and Processes

- A process is different than a thread
  - Conceptually (On Linux it is more complicated)
- Thread: Separate execution streams in same address space
  - "Lightweight process"



- Can have multiple threads within a process

Threads and processes

- Most modern OS's support both
  - Process – defines address space and general process attributes
  - Thread – defines a sequential execution stream within the process context
- Threads are bound to 1 process/address space
  - Isolated from other threads in other processes
- Communicating between processes is hard
  - But communicating between threads is easy
    - Threads share access to same global address space
    - Can read/write same global variables

Thread libraries

- Pthreads
  - POSIX standard API for thread creation and synchronization
- Win32 Threads
  - more complex
- Java Threads
  - Managed by the JVM
  - Exported via a class extension or interface implementation

POSIX Threads

- Common on UNIX variants (Linux, Mac OS X)

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

void pthread_exit(void * value_ptr);

int pthread_join(pthread_t thread, void ** value_ptr);

int pthread_yield(void);
```

pthread_create()

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

- A unique identifier for the thread
- Thread attributes or NULL for the default
- A C Function Pointer
- The argument to pass to the function

Compile

- Need the –pthread option to gcc
- Links in the library

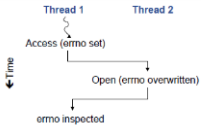
```
gcc –o threadtest threadtest.c -pthread
```

Complications with threads

- Different from fork()/exec()
  - Fork creates a duplicate copy
  - State is replicated, NOT shared
- Threads share all global state
  - Global variables, static variables
  - Open files, open network connections
  - Signals
- What to do about the stack?

Single threaded to multithreaded

- Global variables are a big problem
  - Not just your global variables



- File operations modify **errno** to indicate errors
  - What if two threads both operate on separate files?

Single-threaded to multi-threaded

- Many libraries are not re-entrant
- *Re-entrant:* ability to handle multiple calls to the same function at the same time
- Example a library that formats and prints messages to a log file:

```
int print_log(char * msg) {
      fprintf(logfile, "%s", get_timestamp());
      fprintf(logfile, msg);
      fprintf(logfile, "\n");
}
```