Threads, Processes, and Dispatching

Mendel Rosenblum

Threads, Processes, and Dispatching

Optional readings:

Operating Systems: Principles and Practice: Chapter 4

CS Fundamental Concept

- What is the most fundamental concept in all of CS?

Problem decomposition

Given hard problem, chop it up into several simpler problems that can be thought about separately

Concurrency Motivation

- How to manage concurrency

  Need some way to **decompose** this activity into units that are manageable.

Thread Definition

- Thread: a piece of code executing sequentially on single core
  - Executes a series of instructions in order
    - Easy to reason about: Only one thing happens at a time
  - Decomposition: Concurrent activities can be implemented with a collection of threads
    - Each of which is easy to reason about

Hardware executes instructions concurrently

- Exports an illusion of sequential execution
  - Example of **virtualization**

Execution State

Can't compute in a vacuum: need **state** for the computation to operate on

- Execution state: everything that can affect, or be affected by, a thread:
  - Code, data, registers, call stack
  - Open files, network connections
  - Time of day, …

Thread example from CS106B: Word Count

```
#include <iostream>
#include <fstream>
#include <map>
using namespace std;

int main() {
    ifstream input("input.txt");
    map<string, int> counts;
    string word;

    while (input >> word) {
        counts[word]++;
    }

    for (auto entry : counts) {
        cout << entry.first << " : "
             << entry.second << endl;
    }
    return 0;
}
```

How do we keep track of thread's execution state?

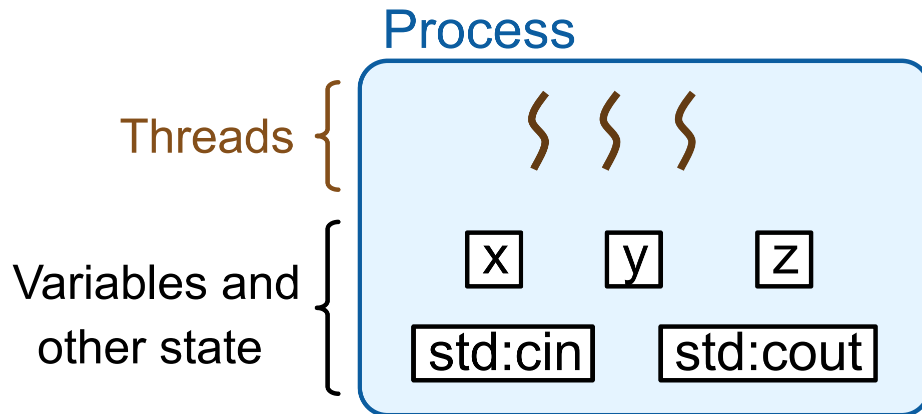
The **Process** abstraction

Process: one or more threads and their execution state

- Most state is shared among all threads
  - Some state private to a thread

Shared or not by threads?

- Code
- Variables
- Registers,
- Call stack
- Open files
- Network connections
- Time of day clock



Threads and Processes

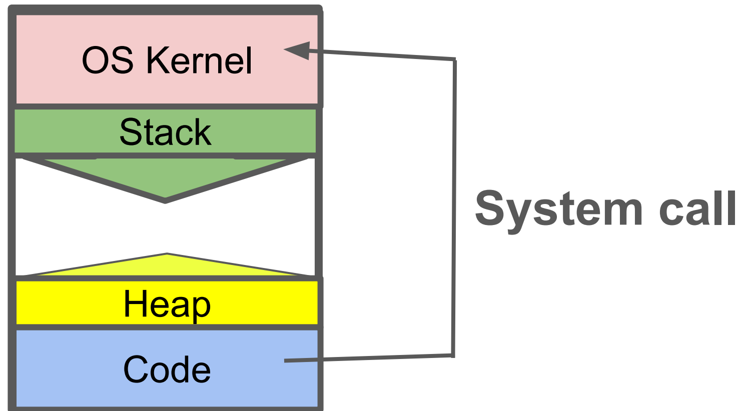
- Many processes have only one thread,
  - Early OSes only allowed one thread per process
  - Why allow multiple threads per process?
    - Take advantage of multicore
    - Simplify app structure
- A process with zero threads isn't interesting
  - Sometimes the term "process" refers to a process with one thread.

How do we create processes?

- Operating system dependent
- Requires a request to the OS kernel: A **system call**
- Linux process manipulation:
    - `fork, execvp, waitpid`

System calls

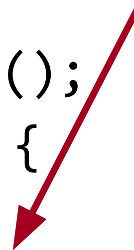
- An OS **system call**
  - Allows process to invokes code in the operating system
  - Exact calling sequence depends on the OS

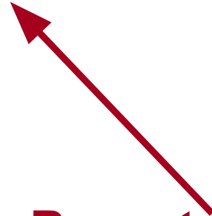


Linux Process Creation: fork system call

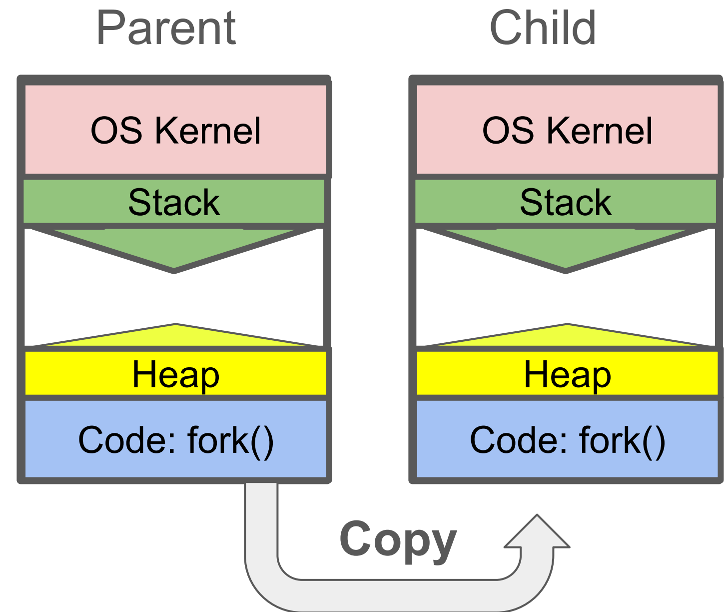
```
int child_pid_or_zero = fork();
if (child_pid_or_zero == 0) {
  // child process starts here
} else {
  // parent process continues here
};
```

Child process



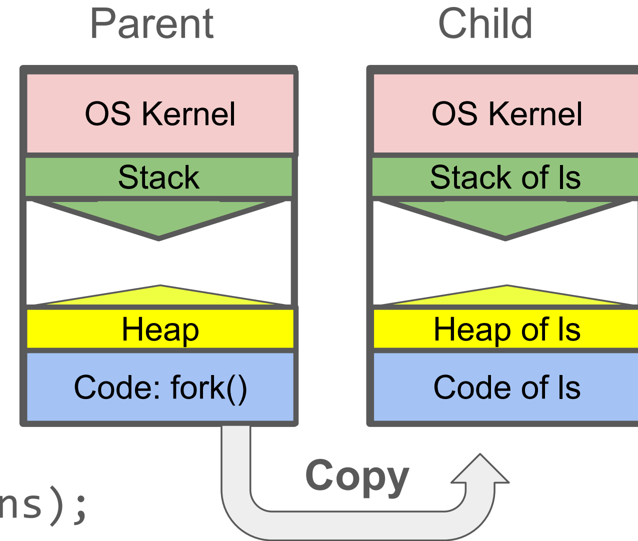


Parent process



Linux Fork/Exec/Wait Example

- execvp - Overwrite current process with new program code and data
- waitpid - Wait for a child process to finish




```
int child_pid_or_zero = fork();
if (child_pid_or_zero == 0) {
    execvp("ls", argv);
    // Not reached
} else {
    waitpid(child_pid_or_zero, &status, options);
};
```

Why copy then overwrite on process creation?

- Advantage: can modify process state before calling exec
  - Change environment
  - Open files
- Disadvantage: wasted work
  - Most of forked state gets thrown away

```
ls > ls.out
```

Advantage of fork/exec can be seeing in Windows…

Microsoft Windows® Process Creation

```
BOOL CreateProcess(
    LPCTSTR lpApplicationName,
    LPTSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation
);

WaitForSingleObject(lpProcessInformation->hProcess, INFINITE);
```

Creation Scorecard

Linux: 2 parameters
- fork (0)
- exec (2)

Windows: 10 params
- CreateProcess (10)

Fork today

- MacOS shell - no fork:
  - **posix_spawn** (pid_t * pid, char * path, posix_spawn_file_actions_t *file_actions, posix_spawnattr_t * attrp, char * argv[], char * envp[]);
- Linux shell - made fork fast enough

Thread Creation

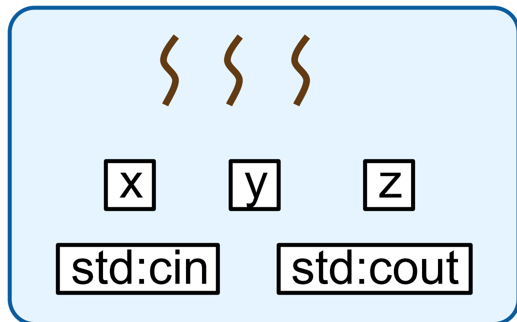
- Processes typically start with a single thread
- A system call can be used to start a new an additional thread
  - Linux: `clone`; MacOS: `thread_create_running`; Windows: `NtCreateThreadEx`
  - A process with more than one thread is called **multithreaded**
- Information for system call:
  - Starting program counter for thread (e.g. routine to call)
  - Starting thread stack region in memory and initial stack pointer
  - Usually some way of passing parameters to initial routine
- Programming language libraries wrap the thread start system call

C++ Thread Example

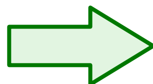
```
#include <thread>
...
std::thread t(func);
...
t.join();
```

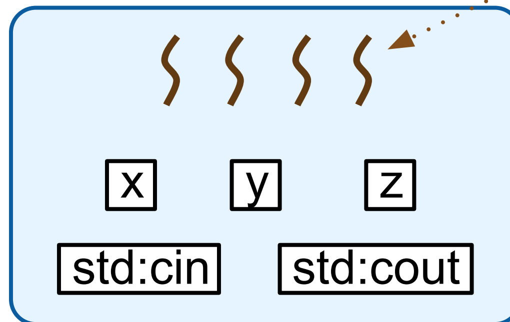
```
void func()
{
    /* This code will run concurrently with
       the code to the left. */
    ...
}
```

New thread shares state with existing threads

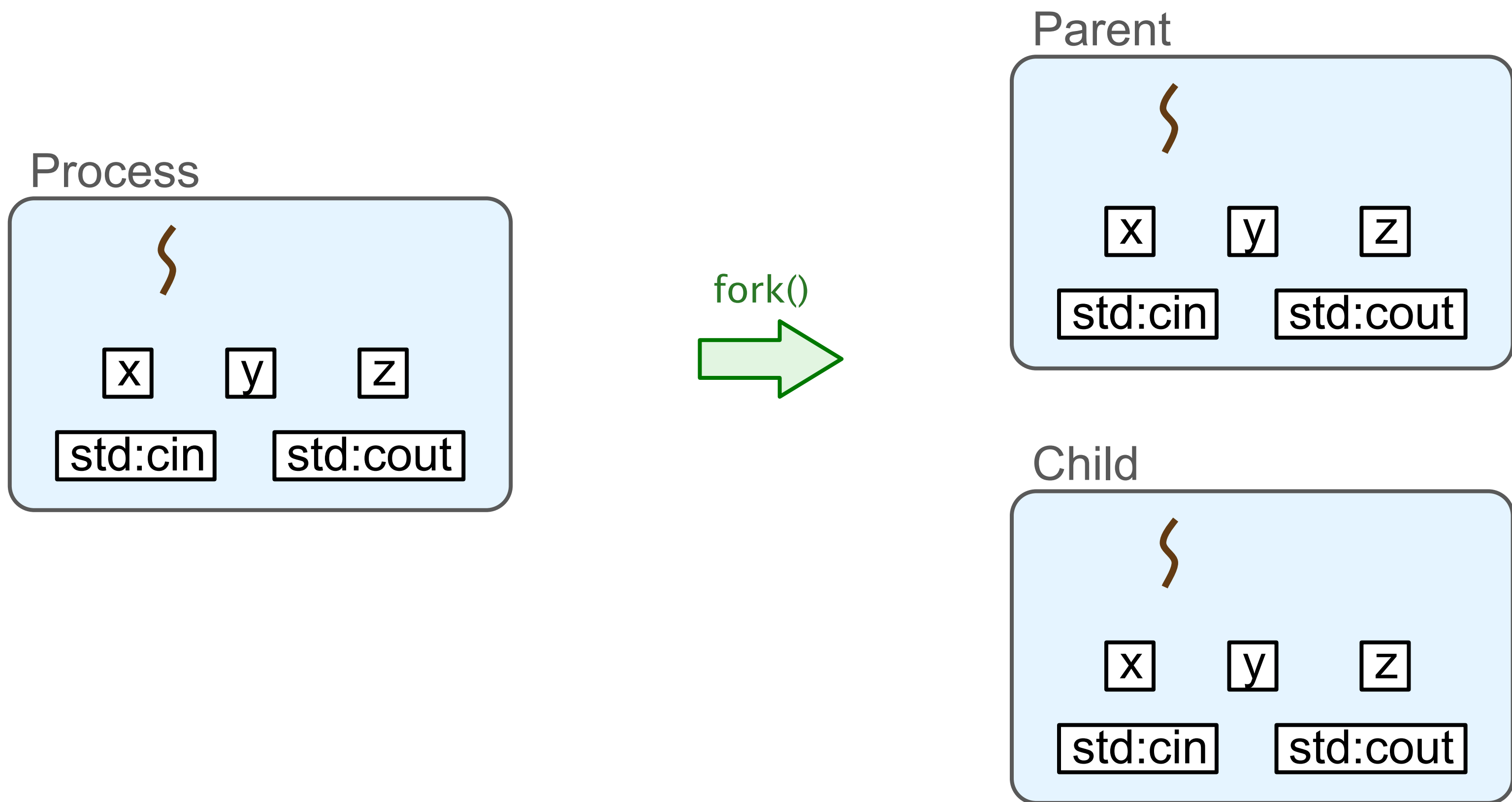


new std::thread(...)





Create New Process



Thread Creation: Other Programming Languages

- C language

```
pthread_create(&thread, NULL, func, &arg);
```

- go language

```
go func(arg);
```

- Python language

```
t = threading.Thread(target=func, args=(arg,))
t.start()
```