Concurrency

→ All about the structure

• divide a task into independent tasks

• open house example
  → greeter, checks, booth, security, food

• helps manage the complexity & opens up opportunities to execute tasks simultaneously (parallelism)
Threads (mechanism for concurrency) → a unit of execution / task
  • sequence of instructions

→ process = a container of resources (VAS, fds, other OS)
  + threads (executions)

main thread stack
    t2 stack
    t3 stack
    ...
    data
    code

VAS for multithreaded process

needs pc, sp, registers to have an execution

single threaded process

multithreaded process

✓ per thread stack
☆ everything else is shared.
Unit of scheduling.
- runs in the process VMs.
- 2 kinds of threads
  1) Kernel threads [managed and scheduled by the kernel]
     - Pthreads!
  2) Thread Control Block
     \{ tid, sp, registers, ptr to PCB \}
  3) Thread Life Cycle = Process Life Cycle
  4) Context switch = Switch from one thread to another

User threads (managed by user lib)
- lightweight alternative, multiplexed on top of kernel threads
Programmer's View | Possible Execution #1 | Possible Execution #2 | Possible Execution #3
---|---|---|---
\[ x = x + 1; \]
\[ y = y + x; \]
\[ z = x + 5y; \]
\[
\]
\[ x = x + 1; \]
\[ y = y + x; \]
\[ z = x + 5y; \]
\[
\]
\[ x = x + 1; \]
\[
\]
\[ x = x + 1; \]
\[ y = y + x; \]
\[
\]
\[ y = y + x; \]
\[ z = x + 5y; \]
\[
\]
\[ z = x + 5y; \]

### Threads API
- `pthread_create` = Spawn
- `pthread_join` = Wait
- `pthread_exit` = Exit
```c
#include <stdio.h>
#include <pthread.h>

int global_x = 0;

void* increment() {
    global_x += 1;
    return NULL;
}

int main(int argc, char* argv[]) {
    pthread_t tid1, tid2;
    pthread_create(&tid1, NULL, increment, NULL);
    pthread_create(&tid2, NULL, increment, NULL);

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

    printf("%d\n", global_x); // minimum? maximum?
    return 0;
}
```