Processes and Threads

44-550: Operating Systems

Processes and Threads

- Exist at execution time
- Have fast state changes
  - in memory
  - waiting
- A process:
  - is a fundamental computational unit
  - can have one or more threads
  - is handled by the process management module
  - requires **system** resources

Processes

- Process (job): program in execution, ready to execute, or waiting for execution
- A program is static; processes are dynamic
- Different types of processes:
  - user processes
  - system processes
- Different wait queues exist for different types of processes

Process Management

- Major function of the OS
- OS manages which processes get what CPU/memory resources at what times
- This is called *multiprogramming*

Multiprogramming

- CPUs must be shared
- Scheduling minimizes idle time
- In Windows, the Task Manager will show how many processes are running
  - How many are there?
  - How many CPU cores are on your computer?
- On a Mac or Linux machine, run `top` or `htop` at the command line

Abstraction

- Big computer sciency term
- Abstraction is used with processes
- Every process gets:
  - Process IDentifier (PID)
  - address
  - memory space
  - program code
  - data
  - resources required

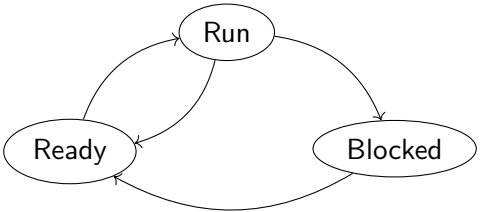
Process Example

Ending the Infinite Background Loop

- Check out `top`
- Run `ps -fu <username>`
- Find the PID of the process
- Kill it

```
kill <pid>
kill -9 <pid> # hard kill
```

Process States and Their Allowable Transitions



Process States

- Run $\rightarrow$ ready: interrupt
- Ready $\rightarrow$ run: preempt or timeslice end
- Run $\rightarrow$ wait/blocked: requesting unavailable resource
- Wait/blocked $\rightarrow$ ready: getting resources

PCB: Process Control Block

- Also known as a process descriptor
- Created with each new process
- Contains all data associated with a process
- OS Queues use a reference (pointer) to a PCB so the queue doesn't need to contain the entire PCB

PCB

name	PID
process owner/user	
state	
list of threads	
list of resources	
list of child process	
address space	
privileges or permissions	
CPU register image	

UNIX PCB

PID
state (ready, run, wait/blocked)
CPU Registers
List of resources
List of child resources
Parent pointer
Permissions
Stack and code pointers

Threads

- Are like “lightweight” processes
  - processes can contain many threads
  - threads change (dynamic within process)
- Current OSes support multithreading
  - multiple threads/tasks per process
- Multiple threads more efficient than multiple processes

Thread Descriptor

Thread ID
Context (program counter within process)
Execution Stack
Local memory
Reference to parent process (for shared resources)
Execution state
List of related threads
Thread Priority
Thread specific resources

Threads

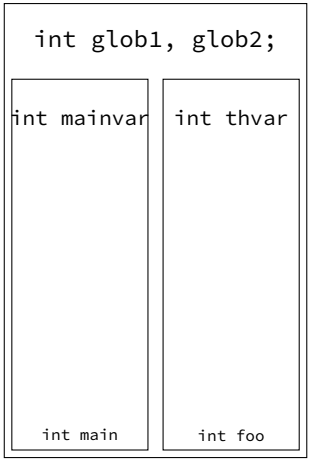
- Every process creates one thread on process creation
  - “main”
- Other threads spawned from the “main” thread
- Threads have same states as processes
- A process will terminate when all threads in the process terminate

Many Threads, One Process

```c
#include <stdio.h>

int globvar1;
int globvar2;

int main(int argc, char* argv[])
{
    int mainvar;
    // do some cool stuff
    return 0;
}
```

Many Threads, One Process

```c
#include <stdio.h>

int globvar1;
int globvar2;
// THIS PROTOTYPE IS INCORRECT
void foo(){
    int thvar;
    // do stuff
}
int main(int argc, char* argv[]){
    int mainvar;
    // do some cool stuff
    // launch thread calling foo
    return 0;
}
```



./a.out

Multithreading

- Multuple threads may exist in one process; regardless the process has only one process descriptor
- Threads have less overhead than processes
- Operations
  - Create thread
  - terminate thread
  - switch between threads (context switching)
  - communication between threads
- All operations have a cost

Multithreading, continued

- Threads share the code and resources of the parent process
  - No need to switch out code and resources when switching threads
- Thus, fewer cache misses!

More Threading!

- Threads exist at both user and kernel level
- User level (no kernel level intervention):
  - Windows
  - POSIX
  - Java...
- Kernel threads perform management tasks done by kernel processes
- A process can still be running even if a thread is blocked