Recap
- General recap of all we have learnt!
- CPU, how computer starts? kernel/user mod, system calls, multitasking
Last week
- Heap vs stack (what size is a proc stack?)
- proc states and control block
- context switching (just overhead 😊)
- process and thread
- process in linux

Contents
- why do we need threads?
- What is a thread?
- What is multicore(multiprocess)?
- How does it fit into the story.
- Is more core ALWAYS better?
- How are threads implemented?
- End... move to networking

What is a thread?
- program=/=process /= thread
- consider client’s accessing a server. Design a model for interaction?
Modern OS are multi-threaded, multiple threads operate in the kernel, and each thread performs a specific task:
- managing devices
- managing memory
- interrupt handling.

What is a thread? (continue)
- program=/=process /= thread
- Have you written a multi-threaded program? main() + gc ...
- Garbage collection!
- If GUI many more thread is a basic unit of CPU utilization
Single threaded process vs. multi-threaded process.
- Why not multiple processes?
Data can be shared, but execution separated!

What is a thread? (continue)
Similar to process, a thread has
- thread ID,
- program counter
- register set
- stack
threads belonging to same process share
- code section
- data section
- OS resources, such as open files
single threaded vs multithreaded

<table>
<thead>
<tr>
<th>code</th>
<th>data</th>
<th>files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>registers</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

thread

Dinosaur book

How to see threads in Linux

- `ps -e -T` | grep firefox
- `-e` all processes
- `-T` all threads
- Exercise: find out threads for a number of well known processes
- *How many threads are running on your machine?*

Why threads?

- Responsiveness
  - a time consuming operation or lengthy process not blocking the whole process
  - Single threaded GUI may block the usage
- Resource sharing
  - Processes: shared memory and message passing (program writes code for them)
  - threads share the memory and the resources of the process to which they belong by default.

Why threads? (continue)

- Economy.
  - Allocating memory and resources for process and context switching is computationally costly
  - threads share the resources of the process; more economical to create and context-switch threads.
  - [in some cases creating a process is about thirty times costlier and switching context is five times slower]
- Scalability
  - multicore allows shared processing, so multi-threading is much faster than multi-processing

multi process, multicore and all that

- multiprocess: many CPU chip
  - Symmetric and asymmetric design
- multicore: single CPU chip has multiple computing core (register, cache...)
  - faster communication (as no inter process communication)
  - significantly less energy consumed
- Be aware: people use two phrases interchangeably!
- blade server (data centre and Cloud):
  - processor process, I/O boards, and networking cards are placed in the same chassis

multicore

- single core one thread executing at at time, two cores two threads...
- Single core illusion of parallelism (by fast switching), multiple core true parallelism
- In single core task run concurrently not parallel
Amdahl’s law

- If I add core will I always make execution faster?
- Speed up \( \leq \frac{1}{\frac{s}{N} + \frac{1-s}{N}} \)
- \( s \) percentage of portion that are serial
- \( N \) number of processing cores
- What happens if \( N \) becomes large (\( N \rightarrow \infty \))...

Throwing in more core is not going to solve the problem always!
You may need to change the program!

Threads an operating system

- User level threads
- Kernel level threads (managed by kernel support)

What is the relationship between the two groups:
- Many-to-one model
- One-to-one model
- Many-to-many model

many to one

- Multiple user-level threads to one kernel thread
- Thread management is done in user space so it is efficient

Not used widely any more:
- Only one thread can access kernel at a time
- All involving process block if a thread makes a blocking system call
- Multiple threads are unable to run in parallel on multicore systems.

one-to-one

- Linux and window use this
- Each user thread is mapped to a kernel thread
- When a thread makes blocking system call, another thread can run.
- Multiple threads can run in multiprocessors.
- But resource hungry and burden on performance
- Upper bound on the number of threads
- What is the maximum number of threads allowed on my machine?

```
cat /proc/sys/kernel/threads-max
```

many-to-many model

- Many user-level threads are handled by multiple kernel threads.
- Developer can create as many user thread
- Kernel can schedule one thread create maximum concurrent user threads is bounded by number of kernel threads
- When threads run in parallel on a multiprocessor there is advantage
- When one thread performs a blocking system call, the kernel can schedule another thread for execution.

How to program threads?

Thread library: API for creating and managing threads.

1. A library entirely in user space with no kernel support i.e. a local function call in user space and not a system call.
2. A kernel-level library supported directly by the operating system, i.e. code and data structures for in kernel space.

Invoking a function in the API for the library typically results in a system call to the kernel.
How to program threads? (continue)

main thread libraries:
1. Windows (uses kernel level library on Windows)
2. POSIX Pthreads (both user and kernel level)
   posix? [ (Portable Operating System Interface)
   family of standards by IEEE, ensure compatibility
   Unix like, but microsoft supports some parts, why?]
   cygwin posix compliant
3. Java threads: implemented using a thread library
   available on the host system[ windows or pthread]
   Java threads are object: implement runnable or extend thread...

communication and networking

End of preliminaries of OS
Be aware:
• lots left to learn
• similarities/differences between OS
• some topics important and we did not study: (memory access, registry/hive,...)
• Communicating processes (on a machine and across)
• You know one method for processes to communicate???

pipe |

• command1 | command2 (both in window and linux | dir
• pipes allow to process to communicate
• but how?
• a temporary file is generated on disk
• command1 writes into it and command2 reads?
• but how?
• standard input, standard output and standard error. (next lecture)
• ordinary pipe (anonymous pipe in Windows)
• named pipe (mkfifo) we dont study this.

running a process and java

• How do I load a process to memory?
• java is example of a process that is an execution environment for other code.
• JWM is a process that interprets the loaded Java code
• loaded java byte code?
• $javap –c programe

Summary

• Motivated and studied the reason for threads
• threads are units of computation
• multicore is better for threads
• sequential part of core dictates how many core can be useful... need to change code to benefit from manycore
• everything in linux is treated as files even pipe
• further mystery!