Concurrency

History, Processes, Threads

ICS432 - Fall 2008
Concurrent and High-Performance Programming

Henri Casanova (henric@hawaii.edu)

A brief history of concurrency (1)

- First machines were used in "single-user mode"
  - I declare: "I am going to use the machine for 2PM till 4PM"
  - I go in the special machine room and sit there for 2 hours
    - I try the punch cards that I have prepared in advance
    - I find bugs
    - I debug
    - etc.
- Extreme lack of productivity
  - During my "thinking time", this multi-million $ machine does nothing

A brief history of concurrency (2)

- Batch Processing!
  - Instead of reserving the machine for a lapse of time to do all my activities (including debugging), I "submit" requests to a "queue"
    - The queue serves requests in order (possibly with priorities)
    - When my program fails and stops, somebody else gets the machine immediately
  - Great but: CPU idle during I/O!

A brief history of concurrency (3)

- Multi-programming!
  - Multiple programs reside in memory at once
  - Required interrupts and memory protection
  - Interrupts are used to switch programs between devices and CPUs
- Lead to concurrency issues in the O/S
  - How do we synchronize all these programs and make sure it all works smoothly?
  - Beginning of theory of concurrent systems (1960)
- Made possible by
  - Increase in memory size
  - Development of virtual memory (ICS 431, ICS 412)

A brief history of concurrency (4)

- Time-sharing!
  - For fast, interactive response, one needs fast context switching
    - i.e., time-slicing of the CPU between jobs of different users
  - Makes it possible to have the illusion that one is alone on a (perhaps slower) machine
- Already common by 1970
- Eventally led to concurrency in user applications!
  - My application is "logically" two concurrent tasks
  - I can now implement it as two concurrent tasks!
  - This is what we talk about in this course
- There is a whole "hardware" side to concurrency, that will be the object of a future lecture

Processes

- A process is essentially a running program
- The O/S keeps track of running program in a "process data structure"
  - A pid
  - A username
  - A state (running, suspended, ...)
  - A program counter (points to the current instruction)
  - A stack (bookkeeping for function calls)
  - A set of file descriptors (open files, network connections,...
  - A page table (maps logical pages to physical pages in RAM or on disk)
    - See virtual memory in ICS412 or ICS431
  - A zone of memory for saving registers
  - The pid of a parent process
  - ...

Processes

- All modern O/S's support multiple active processes at the same time
- The O/S decides which process runs when
- Each process goes through three states
  - Ready: I can run if the O/S would let me
  - Running: I am running right now
  - Blocked: I am waiting for the disk, the network, etc.
- The O/S decides which ready process runs when and for how long
  - This decision impacts the performance and the responsiveness of the computer
  - A lot of thought goes into the design of the O/S's process scheduler (see ICS 412)

Processes

- What processes are running on my laptop?
- On a UNIX/Linux/MacOSX machine you can use the ps command
- That command takes many options
  - man ps
- Let's look at sample output

Terminal — vim — 117x48

```
% ps aux | grep casanova
USER       PID %CPU %MEM      VSZ    RSS  TT  STAT STARTED      TIME COMMAND
casanova 10406   1.3  9.1   446688  94900  ??  S     10:03AM   0:45.09 /Applications/Adobe Acrobat 7.0 Standard/Adob
casanova  8698   0.9  6.1   311432  63560  ??  S      2:46PM   5:10.75 /Applications/Microsoft Office X/Microsoft Po
casanova   264   0.6  2.6   246084  27704  ??  S     Mon09PM   1:49.33 /Applications/Utilities/Terminal.app/Contents
casanova   105   0.0  0.7    78376   7072  ??  Ss    Mon09PM   0:08.91 /System/Library/Frameworks/ApplicationService
casanova   106   0.0  1.5   215148  15524  ??  Ss    Mon09PM   0:08.40 /System/Library/CoreServices/loginwindow.app/
casanova   228   0.0  0.5    55756   5408  ??  Ss    Mon09PM   0:00.59 /System/Library/CoreServices/pbs
casanova   240   0.0  2.4   226624  25460  ??  S     Mon09PM   0:21.32 /System/Library/CoreServices/SystemUIServer.a
casanova   242   0.0  2.4   246360  25348  ??  S     Mon09PM   0:13.28 /System/Library/CoreServices/Finder.app/Conte
casanova   263   0.0  1.5   216792  16204  ??  S     Mon09PM   0:25.97 /Applications/Desktop Manager.app/Contents/Ma
casanova   266   0.0  1.1   209832  11552  ??  S     Mon09PM   1:22.72 /System/Library/PreferencePanes/UniversalAcce
casanova   267   0.0  0.9   209912   9632  ??  S     Mon09PM   0:00.65 /Applications/iCal.app/Contents/Resources/iCa
casanova   268   0.0  0.6   201368   6564  ??  S     Mon09PM   0:46.31 /Library/Frameworks/HPServicesInterface.frame
casanova   269   0.0  0.6   201688   6612  ??  S     Mon09PM   0:47.01 /Library/Frameworks/HPServicesInterface.frame
casanova   272   0.0  1.0   205856  10940  ??  S     Mon09PM   0:29.73 /Library/Printers/hp/HP Communications.app/Co
casanova   301   0.0  0.0    27268    424  ??  Ss    Mon09PM   0:00.00 /Library/Printers/hp/pipedaemon.app/Contents/
casanova   306   0.0  0.7   236744   7072  ??  S     Mon09PM   0:10.73 /System/Library/CoreServices/Dock.app/Content
casanova   413   0.0  0.5    38644   5228  ??  S     Tue08AM   0:03.28 /System/Library/Services/AppleSpell.service/C
casanova   443   0.0  2.5   240288  26248  ??  S     Tue08AM   0:11.18 /System/Library/CoreServices/Dock.app/Content
casanova   444   0.0  1.5   215196  16052  ??  S     Tue08AM   0:00.60 /System/Library/CoreServices/Dock.app/Content
casanova   445   0.0  2.0   224564  20916  ??  S     Tue08AM   0:00.76 /System/Library/CoreServices/Dock.app/Content
casanova   446   0.0  2.2   227872  23300  ??  S     Tue08AM   0:01.71 /System/Library/CoreServices/Dock.app/Content
casanova   447   0.0  2.6   221640  26836  ??  S     Tue08AM   0:11.95 /System/Library/CoreServices/Dock.app/Content
casanova   448   0.0  1.9   215124  19408  ??  S     Tue08AM   0:01.30 /System/Library/CoreServices/Dock.app/Content
casanova   449   0.0  1.7   217632  17664  ??  S     Tue08AM   0:00.60 /System/Library/CoreServices/Dock.app/Content
casanova   450   0.0  1.5   213276  15768  ??  S     Tue08AM   0:07.97 /System/Library/CoreServices/Dock.app/Content
casanova   451   0.0  1.5   212728  15584  ??  S     Tue08AM   0:17.70 /System/Library/CoreServices/Dock.app/Content
casanova   452   0.0  1.6   213308  16576  ??  S     Tue08AM   0:07.51 /System/Library/CoreServices/Dock.app/Content
casanova  8700   0.0  0.5   206224   4936  ??  S      2:46PM   0:07.51 /Applications/Microsoft Office X/Office/Micro
casanova  9961   0.0  2.8   250364  28836  ??  S      5:17PM   1:11.19 /Applications/Mail.app/Contents/MacOS/Mail -p
casanova 10046   0.0  0.1    31836   1004  p2  Ss+    5:34PM   0:00.06 -tcsh
casanova 10085   0.0  2.8   259148  28964  ??  S      9:00AM   0:32.87 /Applications/iChat.app/Contents/MacOS/iChat
casanova 10086   0.0  0.4   174704   3892  ??  Ss     9:00AM   0:00.92 /System/Library/Frameworks/InstantMessage.fra
casanova 10114   0.0  0.1    31836   1004  p3  Ss     9:05AM   0:00.07 -tcsh
casanova 10263   0.0  0.1    31836    944  p4  Ss     9:35AM   0:00.05 -tcsh
casanova 10265   0.0  0.2    28384   1836  p4  S+     9:35AM   0:00.09 vim TODO
casanova 10338   0.0  0.2    28384   1792  p3  S+     9:50AM   0:00.08 vim pc_email_list
casanova 10378   0.0  1.0   203780  10060  ??  S      9:59AM   0:00.45 /System/Library/CoreServices/System Events.ap
casanova 10388   0.0  0.5    61412   4840  ??  SNs   10:01AM   0:01.70 /System/Library/Frameworks/CoreServices.frame
casanova 10390   0.0  0.1    31836   1000  p1  Ss+   10:01AM   0:00.08 -tcsh
casanova 10407   0.0  0.2    39328   1584  ??  Ss    10:03AM   0:00.13 /Applications/Adobe Acrobat 7.0 Standard/Adob
casanova 10417   0.0  3.6   254840  37232  ??  S     10:04AM   0:08.02 /Applications/Safari.app/Contents/MacOS/Safar
casanova 10558   0.0  0.1    31836    972  p5  Ss    10:38AM   0:00.17 -tcsh
casanova 10609   0.0  2.3   226308  24308  ??  S     10:41AM   0:01.61 /Applications/Utilities/Grab.app/Contents/Mac
casanova 10638   0.0  0.0     8780      8  p5  R+    10:44AM   0:00.00 grep casanova
%
```

Processes

- Let's look at the PowerPoint process

Terminal — tcsh — 67x10

```
~% ps -auxww | grep PowerPoint
casanova  8698   1.9  5.9   309848  61872  ??  S     2:46PM   4:57.
14 /Applications/Microsoft Office X/Microsoft PowerPoint /Applicati
ons/Microsoft Office X/Microsoft PowerPoint -psn_0_11403265
casanova 10608   0.0  0.0     8780      8  p5  R+   10:40AM   0:00.
00 grep PowerPoint
~%
```

Processes and Memory

- Each processes has its own address space: a set of memory locations that can be read from and written to
- Virtual memory (ICS431 and ICS412)
  - The illusion that there is a large memory (perhaps larger than the physical memory)
  - The illusion that it is the only one using the memory
  - The illusion is always maintained, but at the cost of degraded performance at times
- This is what makes is possible for developers to write programs and not care about the state of the computer when the program will be run
  - I write a program assuming a given, large address space
  - Only this program can modify this address space
- This is the kind of programming that you've done so far

(UNIX) Process Creation?

- Each time you invoke a command in a Shell (which is itself a process), you create a new process
- Or more appropriately, the Shell creates a new process on your behalf
- So somewhere in the code of the Shell program, there is a place where processes are created
- Processes are created using the fork system call, which can be called from C
- While this is more ICS 412 content, let's see a little bit how fork works
  - you can't graduate with a CS degree without knowing what fork is!

The Fork() System Call

- The fork system call creates a copy of a the process that calls it
  - In particular the memory is copied
- After the call, both processes are free to continue along following different execution paths in the program
- fork() returns an integer
  - It returns the PID of the new process to the "parent" process
  - It returns 0 to the "child" process
- Let's see this on an example

The Fork() System Call

- What does this program do?

```
int count = 0;
if (fork()) {
        while (1) {
                printf("%d\n",count++);
                sleep(1);
        }
} else {
        while(1) {
                printf("%d\n",count--);
                sleep(2);
        }
}
```

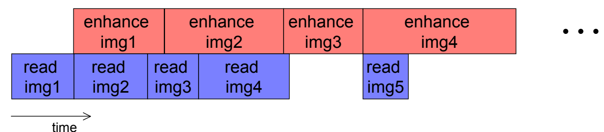
- Let's try it out ...

The Fork() System Call

- The two processes run without really knowing about each other
- The O/S is in charge of deciding when they run
- The O/S ends up interleaving they execution finely so as to provide the illusion that the processes never are interrupted and non-responsive
  - in our example, there is no such concern though because our processes are not meant to be interactive
- Furthermore the two processes have distinct address spaces
  - In our example, the "count" variable is not share between the processes but each process has its own copy of it

Processes for Concurrency?

- To implement a concurrent application, a simple idea is then to use multiple processes
  - Create them via the fork() system call
  - The operating system will be in charge of doing to interleaving, time-slicing
- Let's take the image processing example as in the last set of lecture notes



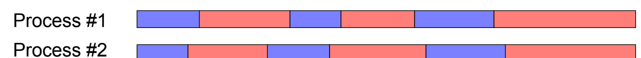
- Question: How do we implement this with processes?

Processes for Concurrency?

- Two processes:
  - P1: for image reading
  - P2: for image processing
- Problem:
  - P1 reads images into its address space
  - P2 cannot access P1's address space!
    - Just like in our example the "count" variable is private to each address space

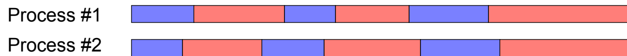
Concurrent Image Processing with Processes

- Without shared address spaces one could say:
  - I have N images to process
  - I am going to use 2 processes
  - Each process will process N/2 images
- Execution could look like this



- Why is this a bad idea?

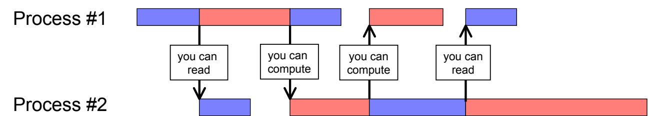
Concurrent Image Processing with Processes



- The problem: the same hardware resource (CPU or Disk) is used at the same time by the two processes
- As a result, they slow each other down
- In the worst case, each box in the figure above is stretched by a factor 2! No performance benefit whatsoever!
- There could be a very lucky execution in which, due to image sizes and image order, the processes are in perfect opposition of phase: one reads while the other compute
  - But you can't count on being lucky

Concurrent Image Processing with Processes

- Therefore, the processes need to be synchronized
  - "I am done writing"
  - "I am starting computing"
- For instance



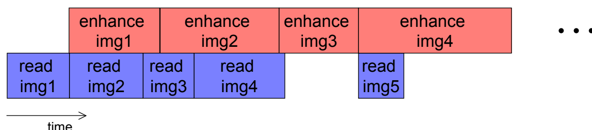
Concurrent Image Processing with Processes

- But what if one process ends up with more work than the other process?
- Then that process will be working alone at the end of the program, with no concurrency at all
- This could be avoided by measuring file sizes for instance and making sure that the two processes process similar number of bytes
- But this doesn't work for all applications
- A more sophisticated approach would be to have one process "give" work to the other
  - This requires some work to synchronize processes further

Concurrent Image Processing with Processes

- There are mechanisms to allow processes to communicate
  - Network communication via sockets
  - Communication via files (never a very good idea)
  - Communications via pipes
  - See ICS412 and ICS451
- But wouldn't sharing memory allow us to do easy synchronization?
  - For instance, share four boolean variables
    - P1_can_compute, P1_can_read: set by P2
    - P2_can_compute, P2_can_read: set by P1
  - When P2 is done computing it sets P2_can_compute to false and P1_can_compute to true
  - When P1 is done reading it sets P1_can_read to false and P2_can_read to true

Reader/Writer



- But, really, if we can share memory between processes, then we could just implement the application the way we had it initially
  - An image reader process
  - An image processor process
  - The data read in memory by the reader is used by the processor

Share Memory between Processes?

- This idea of sharing memory among processes goes completely against the notion of clean, separated address spaces
  - Virtual memory is all about separation, not sharing!
- But, clearly it would be useful and would make programming concurrent applications much simpler
- As a result, there are mechanisms to share memory between processes
- All UNIX systems provide the "shared memory segment" abstraction
  - One process creates a zone of sharable memory
  - It then tells another process: here is a zone we can share

Shared Memory Segments

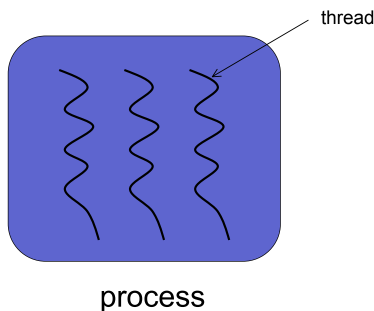
- The idea of shared memory segments is useful, but programming with them is very cumbersome
  - Many lines of code and bookkeeping
- We won't study them in this class, but an operating system's class could use them
  - The same principles about concurrency apply, so if you have to use shared memory segments for some reason, it shouldn't be very difficult after taking this class
  - Look at the man pages for shmget, shmat, shmdt, ..
  - Let's do a "sudo ipcs" on my machine and look...
- Nowadays, the traditional way in which processes share memory is not to use processes at all
- Instead, multiple threads within a single process
  - Especially if you're writing the application from scratch

Threads

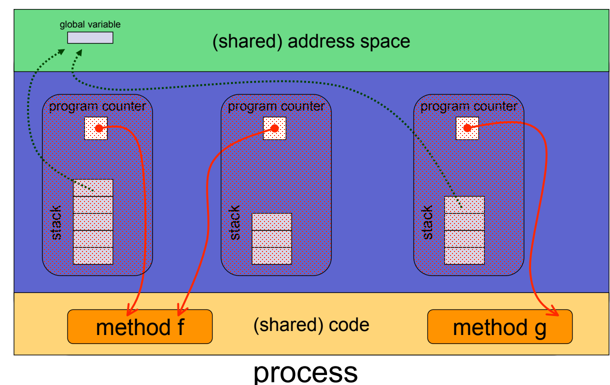
- Threads came about because of the need to write concurrent applications, that is the need for "tasks" that share memory
- Threads can be thought of as processes that share a single address space
- Threads are often called "lightweight processes"
  - N processes have N page tables, N address spaces, N PIDs, ...
  - N threads have 1 page table, 1 address space, 1 PID
- Thing that threads do not share: program counter and stack
  - N threads have N program counters
  - N threads have N stacks
- Therefore, multiple threads can be executing different executions of the program "at the same time", and have followed completely different calling sequences

Threads in a Process

- Typical (but probably useless) representation



Threads in a Process



User Threads / System Threads

- Often you'll hear talk of "user threads" and of "system threads"
- User threads: Implemented entirely by a user-level library
  - The library makes it possible to create, synchronize threads
  - But the O/S is not aware that there are threads in a process
    - As far as it's concerned, processes are not multi-threaded
  - Advantages:
    - Does not require O/S support
    - Can be tuned easily by the user
    - Very low-overhead thread operations because there are no system calls
  - Drawbacks:
    - Cannot leverage multi-core, multi-proc
    - The entire process blocks when one thread blocks

System Threads

- System Threads are provided by the O/S
  - Also called Kernel Threads
- The O/S schedules threads like it would processes
  - They are all "schedulable entities"
- Advantages:
  - Can leverage multi-core, multi-proc
  - The entire process does not block when one thread blocks
- Drawbacks:
  - Higher overhead for thread operations
  - O/S must scale to support large numbers of threads
- In this day and age, you'll probably only use System Threads

Threads vs. Processes

- Sharing memory with threads is straightforward
  - They were designed for this
- Threads are much “cheaper” than processes
  - Less time to start a thread
  - Less time to switch between threads
- Threads are more memory-constrained than processes
  - Simply because many threads can live in a process
- Threads do not benefit from memory protection
  - Can cause nasty bugs when a thread stomps over another thread’s memory (like in the days before virtual memory!)
- We will see that thread synchronization is in principle simple but also fraught with peril

Threads

- Concurrent applications today are almost always written with threads
- How about PowerPoint?
- Excerpt of the output of “ps uxM”

Threads

- Concurrent applications today are almost always written with threads
- How about PowerPoint?
- Excerpt of the output of “ps uxM”

Terminal — vim — 118x16

```
% ps uxM
USER       PID %CPU %MEM      VSZ    RSS  TT  STAT STARTED      TIME PRI     STIME     UTIME COMMAND
[...]
casanova  8698   6.1  9.2   427804  96904  ??  S    Tue02PM  27:09.92  45   2:14.82  24:52.02 Microsoft PowerPoint
          8698   0.0  9.2   427804  96904      S    Tue02PM  27:09.92  43   0:00.00   0:00.00
          8698   0.0  9.2   427804  96904      S    Tue02PM  27:09.92  43   0:00.27   0:00.09
          8698   0.0  9.2   427804  96904      S    Tue02PM  27:09.92  43   0:00.07   0:00.03
          8698   0.0  9.2   427804  96904      S    Tue02PM  27:09.92  54   0:00.14   0:00.25
          8698   0.0  9.2   427804  96904      S    Tue02PM  27:09.92  53   0:00.01   0:00.01
          8698   0.0  9.2   427804  96904      S    Tue02PM  27:09.92  46   0:00.00   0:00.00
          8698   0.0  9.2   427804  96904      S    Tue02PM  27:09.92  46   0:00.08   0:00.04
          8698   0.0  9.2   427804  96904      S    Tue02PM  27:09.92  46   0:00.00   0:00.00
          8698   0.0  9.2   427804  96904      S    Tue02PM  27:09.92  46   0:00.11   0:01.96
[...]
% 
```

- Let’s look at all processes on my laptop...

Programming with Threads

- Many of the programs you use everyday are multithreaded
- During the next lecture we’ll see how to write our first multi-threaded program in Java
- Fallacy: Multi-threading is new
  - Around for 20 years
  - Incorporated in programming languages
    - IBM’s PL/I F, Modula, Ada, etc.
- It’s just gotten very critical due to multi-core