Outline

- Chapter 4: Processes
- Chapter 6: CPU Scheduling
- Chapter 5: Threads
- Introduction to Threads: Birrell
- Continuations: Draves et al.

Processes

- Process is a program in execution
  - Program code
  - Data section
  - Execution context: Program counter, registers, stack
- Process has thread(s) of control
- Many processes "run" concurrently: Process scheduling
  - Fair allocation of CPU and I/O bound processes
    - Context switch

Process creation

- Creating new processes is expensive
  - Resource allocation issue
- Fork mechanism: UNIX, Windows NT
  - Duplicate the parent process
  - Shares file descriptors, memory is copied
  - Exec to create different process
  - Various optimizations to avoid copying the entire parent context (Copy on write (COW), etc..)
- Exec mechanism: VMS, Windows NT
  - New process is specifically loaded

Interprocess communication

- Processes need to communicate with each other
  - Naming
  - Message-passing
    - Direct (to process) or indirect (port, mailbox)
    - Symmetric or asymmetric (blocking, nonblocking)
    - Automatic or explicit buffering (capacity)
    - Send by copy or reference
    - Fixed size or variable size messages
  - Shared memory/mutexes
  - Remote Procedure Call (RPC/RMI)

CPU scheduling

- Interleave processes so as to maximize utilization of CPU and I/O resources
- Scheduler should be fast as time spent in scheduler is wasted time
  - Switching context (h/w assists – register windows [sparc])
  - Switching to user mode
  - Jumping to proper location
- Preemptive scheduling:
  - Process could be in the middle of an operation
  - Especially bad for kernel structures
- Non-preemptive (cooperative) scheduling:
  - Starvation

Classical algorithms

- First come, first served (FCFS)
  - Convoy effect (all processes get bunched up behind long process)
- Shortest job first (shortest next CPU burst)
  - "Optimal"
  - Need oracle to predict next duration: prediction based on history
- Priority scheduling
  - Priority inversion
  - Starvation
- Round robin scheduling
- Multilevel queue scheduling
- Multilevel feedback queue scheduling
- Multiprocessor scheduling
  - Gang scheduling, Non Uniform Memory Access, Processor affinity
- Real-Time scheduling
  - Hard and soft real time systems

Threads

- Applications require concurrency. Threads provide a neat abstraction to specify concurrency
- E.g. word processor application
  - Needs to accept user input, display it on screen, spell check and grammar check
  - Implicit: Write code that reads user input, displays/formats it on screen, calls spell checked etc. while making sure that interactive response does not suffer. May or may not leverage multiple processors
  - Threads: Use threads to perform each task and communicate using queues and shared data structures
  - Processes: expensive to create and do not share data structures and so explicitly passed

Threads - Benefits

- Responsiveness
  - If one "task" takes too long, other "tasks" can still proceed
- Resource sharing:
  - Grammar checker can check the buffer as it is being typed
- Economy:
  - Process creation is expensive (spell checker)
- Utilization of multiprocessor architectures:
  - If we had four processors (say), the word processor can fully leverage them
- Pitfalls:
  - Shared data should be protected or results are undefined
    - Race conditions, dead locks, starvation (more Thurs)

Thread types

- Continuum: Cost to create and ease of management
- User level threads (e.g. pthreads)
  - Implemented as a library
  - Fast to create
  - Cannot have blocking system calls
  - Scheduling conflicts between kernel and threads. User level threads cannot do anything is kernel preempts the process
- Kernel level threads
  - Slower to create and manage
  - Blocking system calls are no problem
  - Most OS's support these threads

Threading models

- One to One model
  - Map each user thread to one kernel thread
- Many to one model
  - Map many user threads to a single kernel thread
  - Cannot exploit multiprocessors
- Many to many
  - Map *m* user threads to *n* kernel threads

Threading Issues:

- Cancellation:
  - Asynchronous or deferred cancellation
- Signal handling:
  - Relevant thread
  - Every thread
  - Certain threads
  - Specific thread
- Pooled threads (web server)
- Thread specific data

Threads – Andrew Birrell

- Seminal paper on threads programming
  - Old but most techniques/experiences are still valid
- Birrell
  - Xerox PARC Dec SRC Microsoft Research
  - Invented Remote Procedure Calls (RPC)
  - Personal Juke box (hard disk based mp3 – Apple iPOD?)
  - Worked on Cedar, Distributed FS etc for ~25 years (1977-

Threads – Andrew Birrell

- Dec SRC and Xerox PARC were the premium systems research labs
- PARC researchers invented:
  - Personal computers - Alto
  - Mouse
  - Windows - Star
  - Bitmapped terminals
  - Icons
  - Ethernet
  - Smalltalk
  - Bravo – first WYSIWYG program
  - Laser printer
  - …

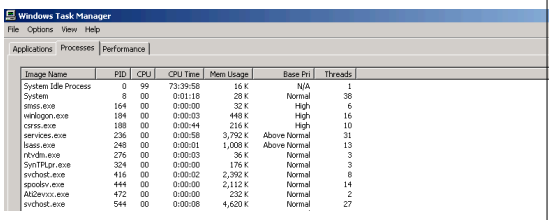
Continuations

- Richard Draves – Leads Systems and Networking group at Microsoft Research
  - Mach, Rialto
- Brian Bershad – Univ. of Washington CMU UW
  - Perennial SOSP contributor….
- Richard F. Rashid – CMU Head, MS Research
  - Mach, ….
- Randall Dean – CMU, VP, Mercury Systems?
  - Mach, ?

Key idea

- Optimizing Mach
  - Application level representation of state while blocked
  - Application code to restore stack
    - Tradeoff stack space for complexity (application code)
- Is this relevant?
  - Current processors have lots of memory, so why bother?
  - Per processor kernel stack
    - reduce cache and TLB misses
- Software engineering concerns?
- Interrupt driven, co-routine style services
  - Much current work in MS Research and other places

Windows 2000



wizard.cse.nd.edu status (using top)

```
load averages: 0.18, 0.09, 0.09 23:53:20
118 processes: 113 sleeping, 3 zombie, 2 on cpu
Memory: 512M real, 267M free, 129M swap in use, 1260M swap free

PID   USERNAME THR PRI  NICE  SIZE RES STATE TIME CPU COMMAND
17213 surendar  1 20 0 1600K  992K    cpu/0  0:08 16.94% yes
16623 root      1 58 0 4800K 3688K    sleep  0:01 0.19% sshd
17218 surendar  1 48 0 2112K 1136K    cpu/1  0:00 0.14% top
16840 www       3 48 0   14M 7096K    sleep  0:00 0.14% httpd
16626 surendar  1 48 0 3656K 2944K    sleep  0:01 0.08% tcsh
272   root     21 58 0 4448K 1648K    sleep  1:16 0.06% syslogd
          [...]
```

- **-L option for ps will list the information about lwps**

Discussion

- Constant tension between moving functionality to upper layers; involving the application programmer and performing automatically at the lower layers
- Automatically create/manage threads by compiler/ system? (open research question)