Application Interface

- Unstructured
  - MS Dos
- Event driven
  - PalmOS
- File system based
  - UNIX, Plan 9
- Object oriented
  - Hydra, OPAL
- Distributed OS
  - Amoeba
- Real time
  - QNX
- Single Address Space OS (SASOS)
  - OPAL

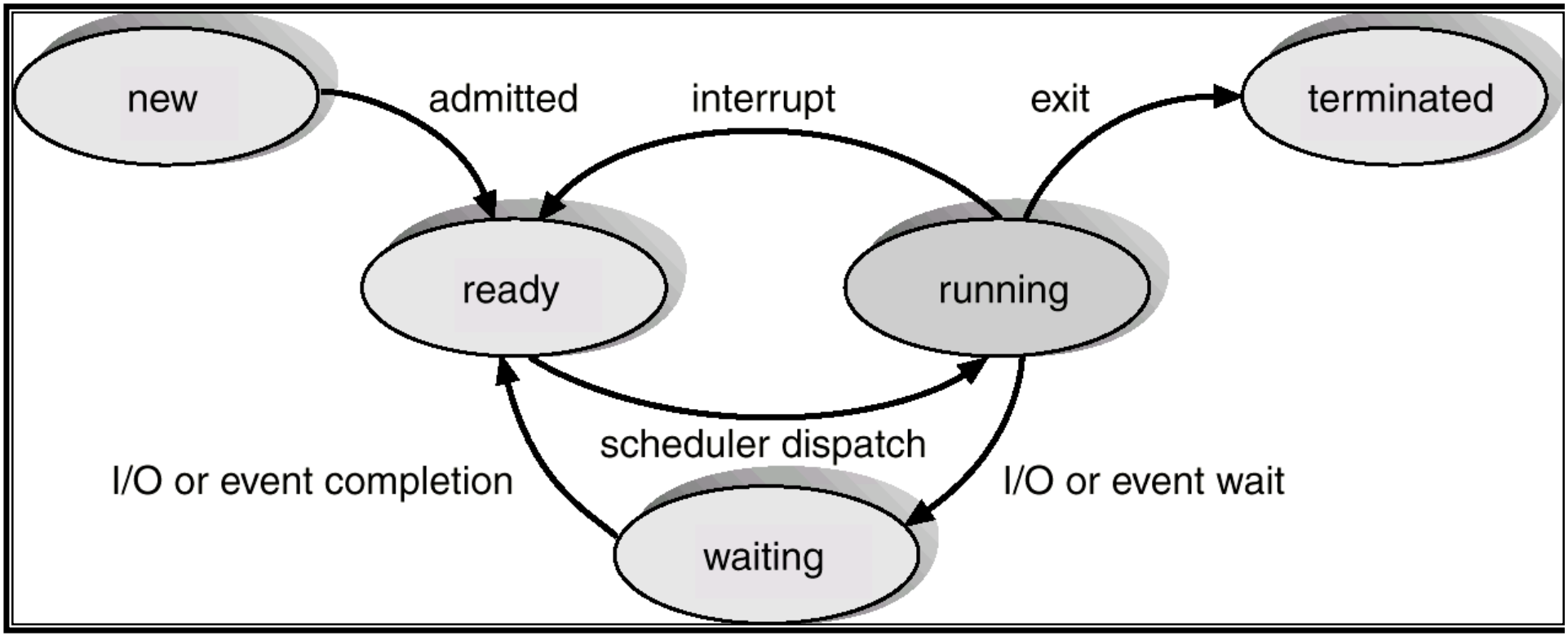
Outline

- Chapter 4: Processes
- Chapter 5: Threads
- Introduction to Threads: Birrell

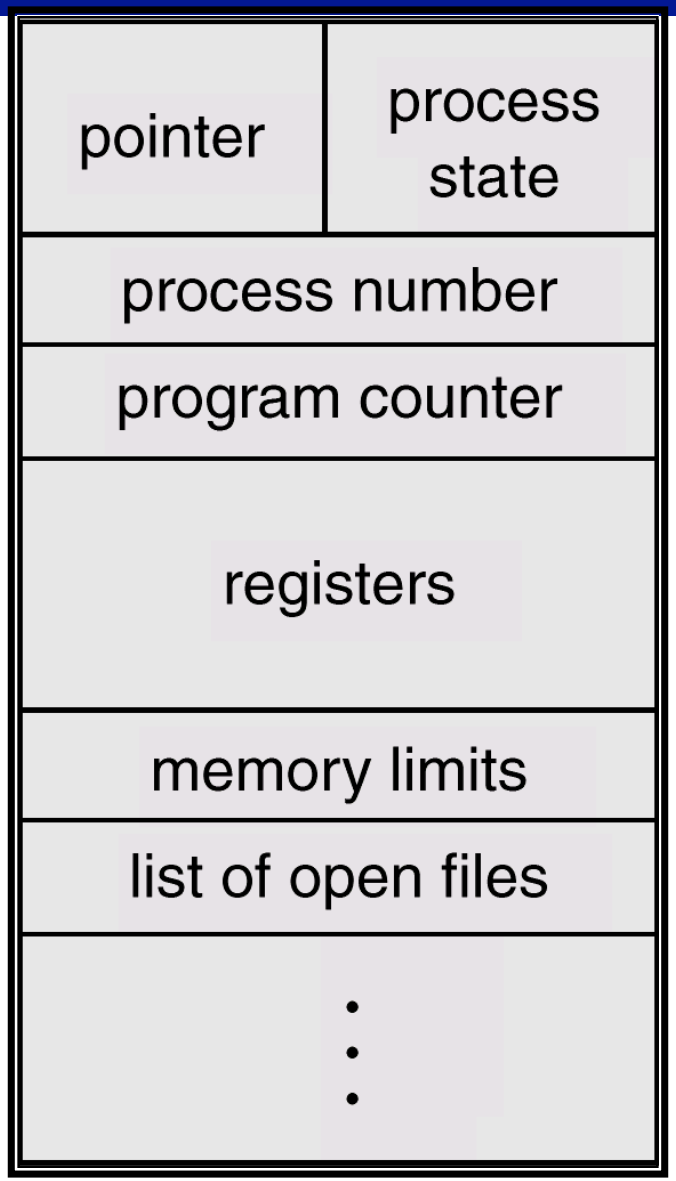
Processes

- 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

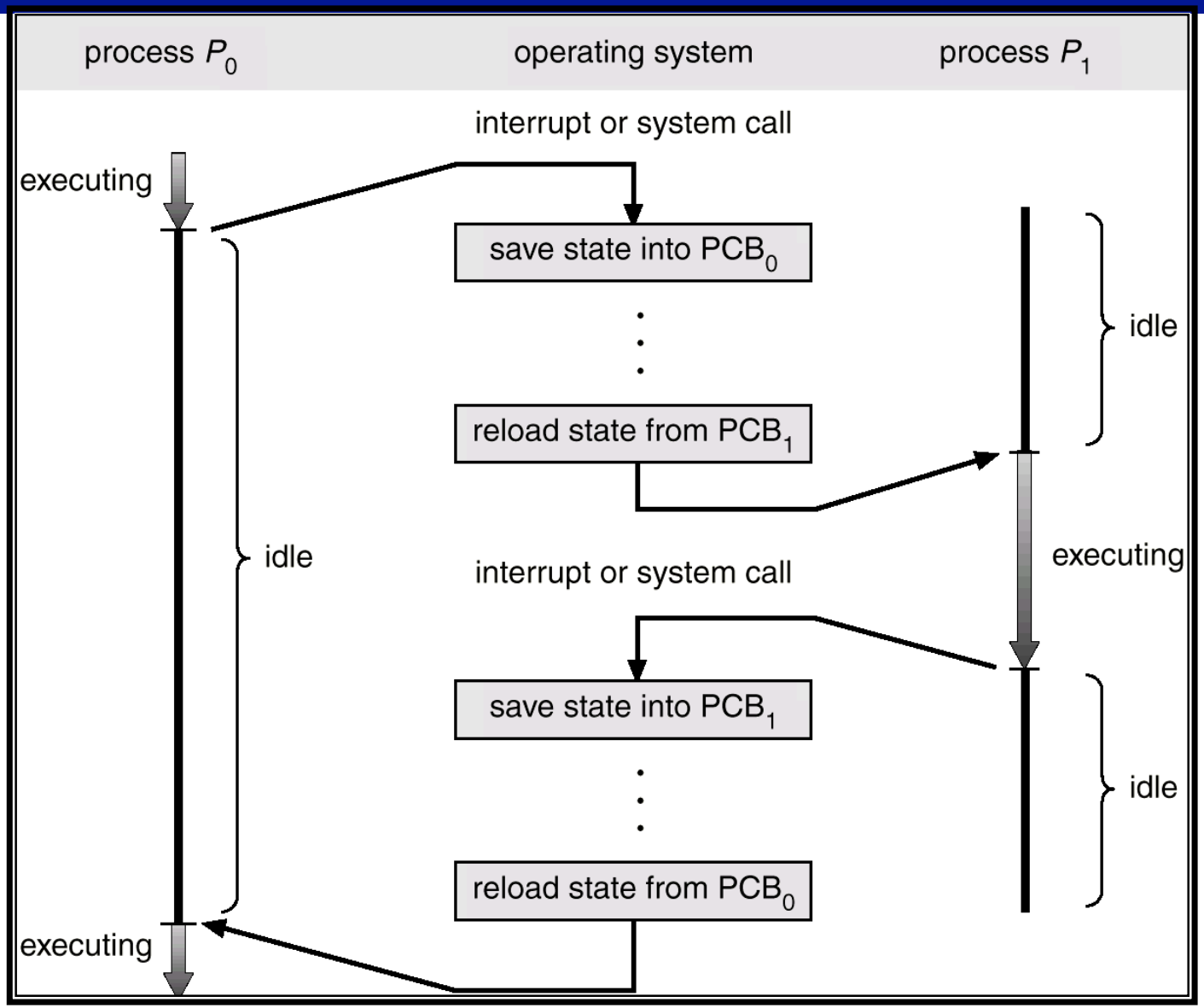
Process States



Process Control Block



Process 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)
- Bounded buffer problem

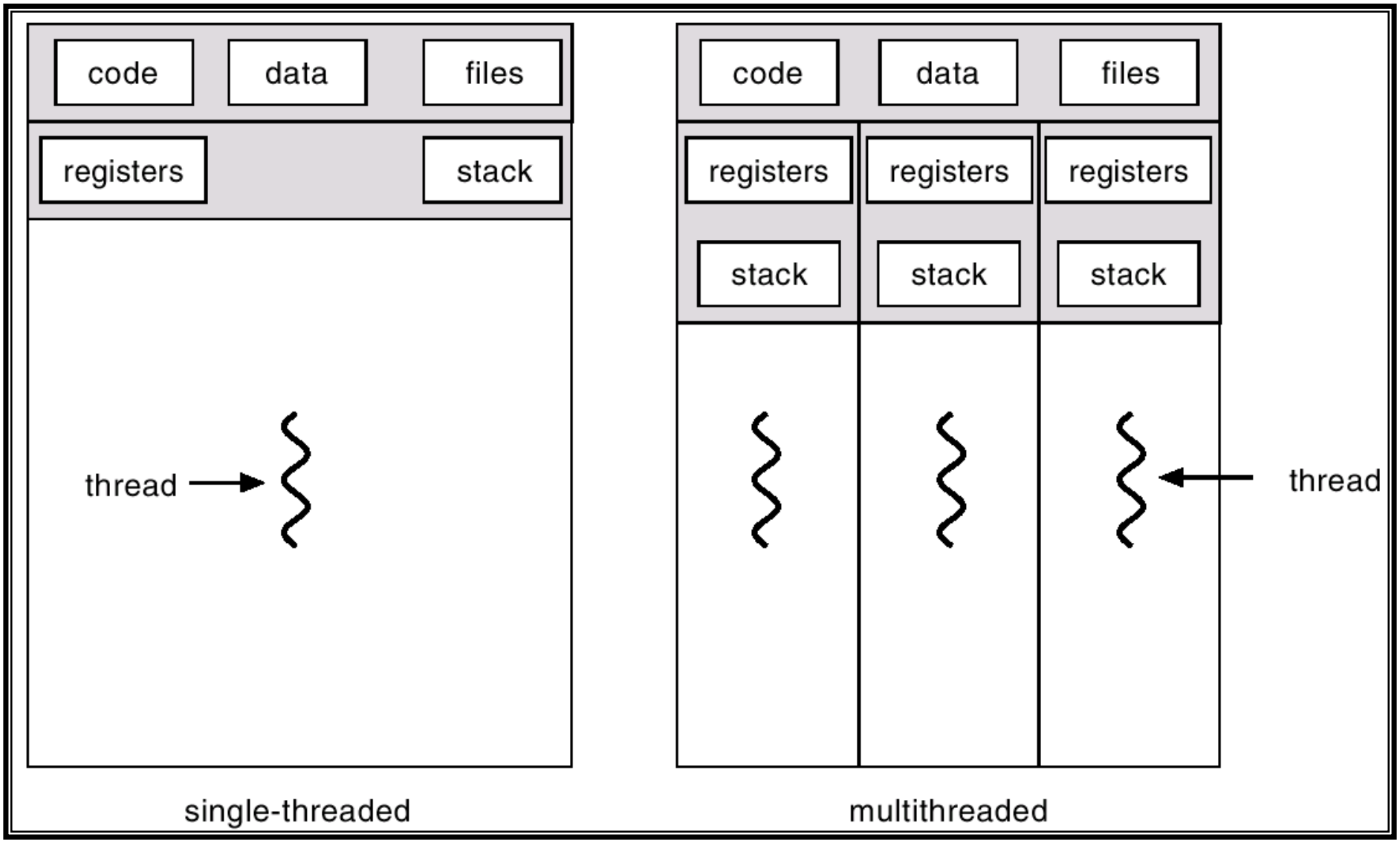
CPU scheduling

- 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

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

Threaded application



Threads - Benefits

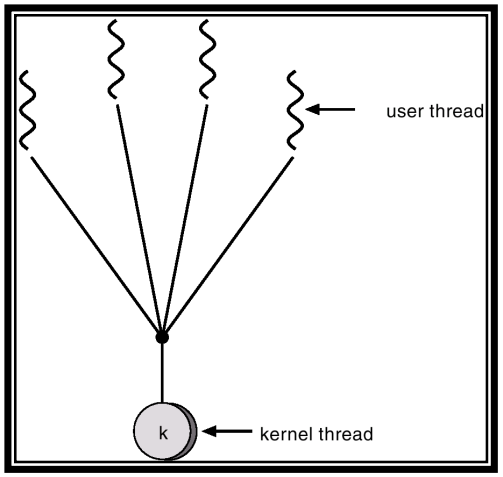
- Responsiveness
  - If one “task” takes too long, other “tasks” can still proceed
- Resource sharing:
  - Grammar checker can check the buffer as it is being typed
- Economy:
  - Process creation is expensive (spell checker)
- Utilization of multiprocessor architectures:
  - If we had four processors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

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

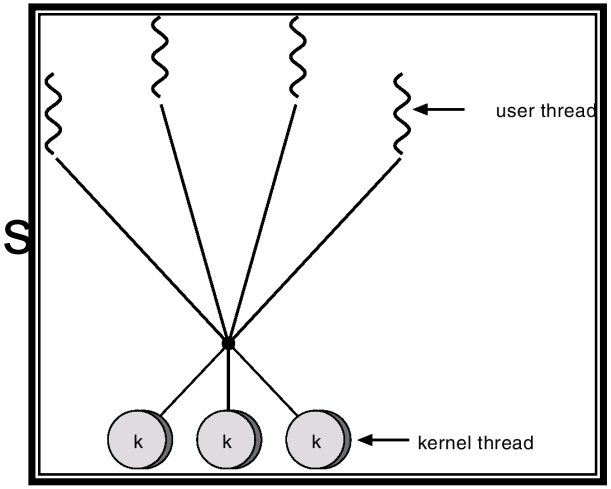
Threading models

- 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
  - …

Windows 2000

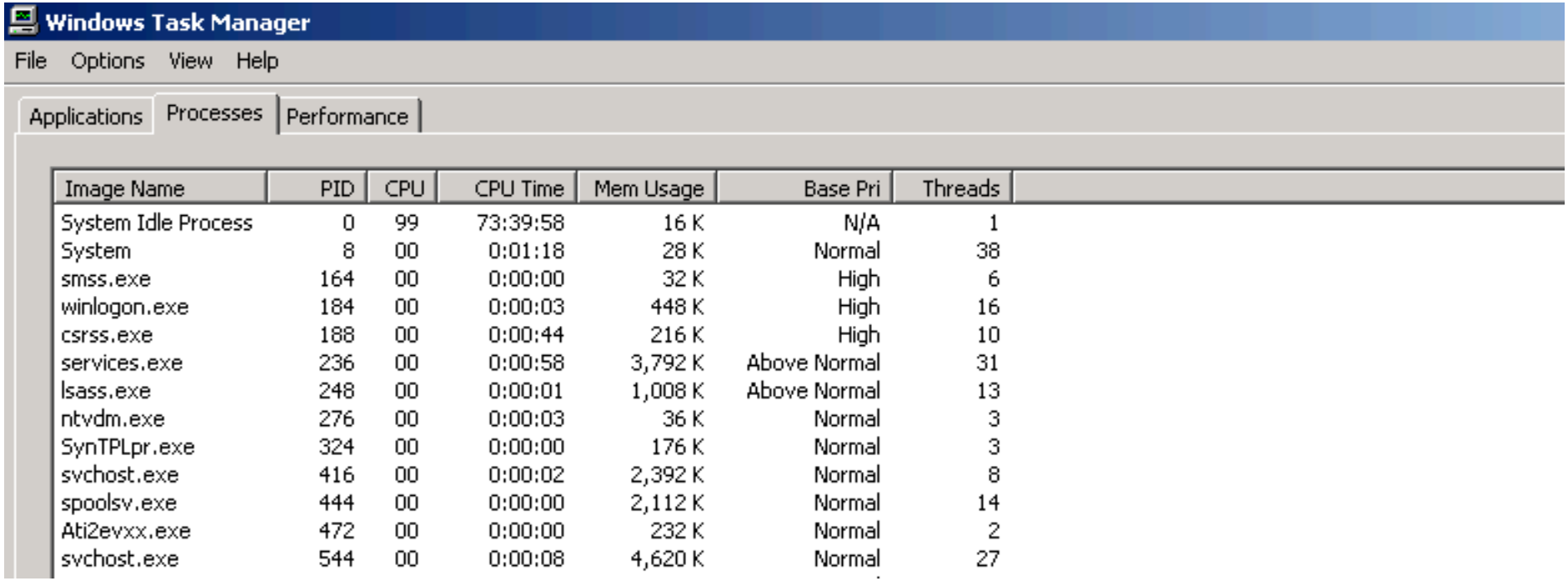


Image Name	PID	CPU	CPU Time	Mem Usage	Base Pri	Threads
System Idle Process	0	99	73:39:58	16 K	N/A	1
System	8	00	0:01:18	28 K	Normal	38
smss.exe	164	00	0:00:00	32 K	High	6
winlogon.exe	184	00	0:00:03	448 K	High	16
csrss.exe	188	00	0:00:44	216 K	High	10
services.exe	236	00	0:00:58	3,792 K	Above Normal	31
lsass.exe	248	00	0:00:01	1,008 K	Above Normal	13
ntvdm.exe	276	00	0:00:03	36 K	Normal	3
SynTPLpr.exe	324	00	0:00:00	176 K	Normal	3
svchost.exe	416	00	0:00:02	2,392 K	Normal	8
spoolsv.exe	444	00	0:00:00	2,112 K	Normal	14
Ati2evxx.exe	472	00	0:00:00	232 K	Normal	2
svchost.exe	544	00	0:00:08	4,620 K	Normal	27

Wizard ‘ps -cfLeP’ output

```
UID        PID  PPID   LWP  PSR   NLWP CLS  PRI   STIME  TTY      LTIME CMD
root         0     0     1    -      1 SYS   96   Aug 03 ?         0:01 sched
root         1     0     1    -      1  TS   59   Aug 03 ?         7:12 /etc/init -
root         2     0     1    -      1 SYS   98   Aug 03 ?         0:00 pageout
root         3     0     1    -      1 SYS   60   Aug 03 ?       275:46 fsflush

root       477   352     1    -      1  IA   59   Aug 03 ??        0:0
                                                                  /usr/openwin/bin/fbconsole -d :0
root        62     1    14    -     14  TS   59   Aug 04 ?         0:00
                                                                  /usr/lib/sysevent/syseventd
```

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)