Rethinking OS Design

Workloads & Benchmarks
To discuss workload and measurement issues:
• representative benchmarks of workload
• appropriate metrics for target workload
• ability to measure
**Application Suites**

- Fixed demand - constant demand over time.
  - Multimedia. Example: DVD player
  - sufficiency
- Variable demand - interactive.
  - Productivity applications with user input: Word™, Excel™, Netscape™, PhotoShop™
  - Entertainment: Tetris, Quake, MP3 player
  - common to model infinitely fast user
  - variability in load

**OS Abstractions and API’s**

Abstract machine environment. The OS defines a set of logical resources (objects) and operations on those objects (an interface for the use of those objects).

Hides the physical hardware.

---

**Invoking Kernel Services - System Call Interface**

- User Programs
  - **Syscalls**
  - **OS**
  - **Kernel**
- **Machine instructions**
  - **HW**

UNIX
- fork, exec, exit, join
- open, close, read, seek

PalmOS
- EvtGetEvent
- MemHandleLock
- SndPlaySystemSound

- For a user to do something “privileged”, it must invoke an OS procedure providing that service.
  - A System Call.

- Special trap instruction that saves essential context, causes a mode change and transfers to a handler.
Idleness?
Defining the Process Abstraction

- Unit of scheduling
- One (or more*) sequential threads of control
  - program counter, register values, call stack
- Unit of resource allocation
  - address space (code and data), open files
  - sometimes called *tasks or jobs*

Process Mechanisms
Context Switching

- When a process is running, its program counter, register values, stack pointer, etc. are contained in the hardware registers of the CPU. The process has direct control of the CPU hardware for now.
- When a process is not the one currently running, its current register values are saved in a process descriptor data structure (PCB - *process control block*).
- Context switching involves moving state between CPU and various processes’ PCBs by the OS.

Process State Transitions

- Ready
- Running
- Blocked

Process Mechanisms
PCBs on Queues

- PCB data structure in kernel memory represents a process (allocated on process creation, deallocated on termination).
- PCBs reside on various *state queues* (including a different queue for each "cause" of waiting) reflecting the process’s state.
- As a process executes, the OS moves its PCB from queue to queue (e.g. from the “waiting on I/O” queue to the “ready to run” queue).
Unix Process Model

- Simple and powerful primitives for process creation and initialization.
  - fork syscall creates a child process as (initially) a clone of the parent
  - parent program runs in child process to set it up for exec
  - child can exit, parent can wait for child to do so
- Rich facilities for controlling processes by asynchronous signals.
  - notification of internal and/or external events to processes or groups
  - the look, feel, and power of interrupts and exceptions
  - default actions: stop process, kill process, dump core, no effect
  - user-level handlers

(Traditional) Unix Abstractions

- Processes - thread of control with context
- Files - a named linear stream of data bytes
- Sockets - endpoints of communication between unrelated processes

Files (& everything else)

- Descriptors are small unsigned integers used as handles to manipulate objects in the system, all of which resemble files.
- open with the name of a file returns a descriptor
- read and write, applied to a descriptor, operate at the current position of the file offset. lseek repositions it.
- Pipes are unnamed, unidirectional I/O stream created by pipe.
- Devices are special files, created by mknod, with ioctl used for parameters of specific device.
- Sockets introduce 3 forms of sendmsg and 3 forms of recvmsg syscalls.
Unix Process Control

- The `fork` syscall copies a zero to the child and the child process ID to the parent.

- Parent waits until the child exits with status.

- Exit returns status.

- Wait variants allow wait on specific child, or notification of stops and other signals.

- Child process passes status back to parent on exit, to report success/failure.

Child Discipline

- After a `fork`, the parent program has complete control over the behavior of its child.

- The child inherits its execution environment from the parent, but the parent program can change it.
  - sets bindings of file descriptors with `open`, `close`, `dup`
  - pipe sets up data channels between processes

- Parent program may cause the child to execute a different program, by calling `exec*` in the child context.

Exec, Execve, etc.

- Children should have lives of their own.
- `Exec*` “boots” the child with a different executable image.
  - parent program makes `exec*` syscall (in forked child context) to run a program in a new child process
  - `exec*` overlays child process with a new executable image
  - restarts in user mode at predetermined entry point (e.g., `crt0`)
  - no return to parent program (it’s gone)
  - arguments and environment variables passed in memory
  - file descriptors etc. are unchanged

Fork/Exit/Wait Example

- Child process starts as clone of parent, with parent environment.

- Parent and child run independently, sharing resources.

- On exit, release memory and other resources.

- When parent reaps child, it can use `wait` to report success/failure.
Join Scenarios

• Several cases must be considered for join (e.g., exit/wait).
  – What if the child exits before the parent joins?
  • “Zombie” process object holds child status and stats.
  – What if the parent continues to run but never joins?
  • How not to fill up memory with zombie processes?
  – What if the parent exits before the child?
  • Orphans become children of init (process 1).
  – What if the parent can’t afford to get “stuck” on a join?
  • Unix makes provisions for asynchronous notification.

Signals

• Signals notify processes of internal or external events.
  – the Unix software equivalent of interrupts/exceptions
  – only way to do something to a process “from the outside”
  – Unix systems define a small set of signal types
• Examples of signal generation:
  – keyboard ctrl-c and ctrl-z signal the foreground process
  – synchronous fault notifications, syscall errors
  – asynchronous notifications from other processes via kill
  – IPC events (SIGPIPE, SIGCHLD)
  – alarm notifications

Signals

• Signals notify processes of internal or external events.
  – the Unix software equivalent of interrupts/exceptions
  – only way to do something to a process “from the outside”
  – Unix systems define a small set of signal types
• Examples of signal generation:
  – keyboard ctrl-c and ctrl-z signal the foreground process
  – synchronous fault notifications, syscall errors
  – asynchronous notifications from other processes via kill
  – IPC events (SIGPIPE, SIGCHLD)
  – alarm notifications

Process Handling of Signals

1. Each signal type has a system-defined default action.
   abort and dump core (SIGSEGV, SIGBUS, etc.)
   ignore, stop, exit, continue
2. A process may choose to block (inhibit) or ignore some signal types.
3. The process may choose to catch some signal types by specifying a (user mode) handler procedure.
   specify alternate signal stack for handler to run on
   system passes interrupted context to handler
   handler may munge and/or return to interrupted context

Using Signals

```c
int alarmflag=0;
alarmHandler ()
{
  printf("An alarm clock signal was received\n");
  alarmflag = 1;
}
main()
{
  signal(SIGALRM, alarmHandler);
  alarm(3); printf("Alarm has been set\n");
  while (!alarmflag ) pause ();
  printf("Back from alarm signal handler\n");
}
```

Using Signals

```c
int alarmflag=0;
alarmHandler ()
{
  printf("An alarm clock signal was received\n");
  alarmflag = 1;
}
main()
{
  signal(SIGALRM, alarmHandler);
  alarm(3); printf("Alarm has been set\n");
  while (!alarmflag ) pause ();
  printf("Back from alarm signal handler\n");
}
```
Yet Another User’s View

```c
main(argc, argv)
{ int argc; char * argv[];
  { int pid;
    signal(SIGCHLD, childhandler);
    pid = fork();
    if (pid == 0) /* child */
      { execvp(argv[2], &argv[2]); }
    else
      { sleep(5);
        printf("child too slow\n");
        kill(pid, SIGINT);
      }
  }
}

childhandler()
{ int childPid, childStatus;
  childPid = wait(&childStatus);
  printf("child done in time\n");
  exit;
}
```

SIGCHLD sent by child on termination; if SIGHUP, daemon

File System Calls

```c
char buf[BUFSIZE];
int fd;
if ((fd = open("../zot", O_TRUNC | O_RDWR)) == -1) {
  perror("open failed");
  exit(1);
}
while(read(0, buf, BUFSIZE)) {
  if (write(fd, buf, BUFSIZE) != BUFSIZE) {
    perror("write failed");
    exit(1);
  }
}
```

Pathnames may be relative to process current directory.
Process does not specify current file offset: the system remembers it.
Process passes status back to parent on exit, to report success/failure.

File Sharing Between Parent/Child

```c
main(int argc, char * argv[])
{ char c;
  int fdrd, fdwt;
  if ((fdrd = open(argv[1], O_RDONLY)) == -1)
    exit(1);
  if ((fdwt = creat(argv[2], 0666)) == -1)
    exit(1);
  fork();
  for (;;) {
    if (read(fdrd, &c, 1) != 1)
      exit(0);
    write(fdwt, &c, 1);
  }
}
```

Sharing Open File Instances

A process may be allocated file descriptors:
- Standard descriptors (0, 1, 2) for input, output, error messages (stdin, stdout, stderr).
- Open files are named to an integer file descriptor.

Process does not specify current file offset; the system remembers it.

Sharing Open File Instances

Objects shared seek offset in shared file table entry

System open file table

Parent file descriptors

Child file descriptors

Shared file (inode or vnode)

User ID

Process ID

Process group ID

Parent PID

Signal state

Siblings

Children

File Sharing Between Parent/Child

File System Calls

Yet Another User’s View

Sharing Open File Instances
File Directories

- Directories are (guess what?) a type of file.
- A hierarchy of directories - a filesystem - has a root (/)
- Pathnames are *absolute* or *relative* to working directory, .....
- Root filesystem may have roots of other filesystems mounted into the hierarchy.
- Directories manipulated by `link()`, `unlink()`, `mkdir()`, `rmdir()`.

Devices

Various devices are abstracted as *special files*.

- Named by a filename.
- Accessed via `open()`, `close()`, `read()`, and `write()`
- Idiosyncratic operations of the device are access through `ioctl()` calls.

Popular Embedded / RT OS’s

- Microsoft WinCE - WIN32 “lite” API
- WindRiver VxWorks
- pSOS (recently bought out by WindRiver)
- Green Hills INTEGRITY RTOS
- Embedded Linux - e.g., Hard Hat Linux (Montevista software)
- embedded Java platforms with Jini (for access to distr. services)

ACPI

*Advanced Computer Power Initiative*

Brought to you by Intel, Microsoft, and Toshiba and designed to enable OS Directed Power Management (OSPM).

- Goal is to be able to move power management into software for more sophisticated policies
- Abstract OS-HW interface
Power States

G: global states apply to entire system and are visible to user
D: states of individual devices
S: sleeping states within the G1 state
C: CPU states

G1: Sleep
G1: Soft off
S1: S2 S3 S4

Transmeta Crusoe Power

Transmeta Crusoe ACPI Power States

<table>
<thead>
<tr>
<th>ACPI System State</th>
<th>Processor Power State</th>
<th>DDR, SDRAM Power State</th>
<th>Clock Generator</th>
</tr>
</thead>
<tbody>
<tr>
<td>G0/S0 (Working)</td>
<td>C0 Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>Legacy</td>
<td>C1 Normal</td>
<td>Normal</td>
<td>Banishing</td>
</tr>
<tr>
<td></td>
<td>C2 Quiet Sleep</td>
<td>Self refreshed</td>
<td>Banishing</td>
</tr>
<tr>
<td></td>
<td>C3 Deep Sleep</td>
<td>Self refreshed</td>
<td>Clocks stopped</td>
</tr>
<tr>
<td>G1/S1 (Sleeping)</td>
<td>Deep Sleep</td>
<td>Self refreshed</td>
<td>PLL shut down</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G1/S2 (Suspend to RAM)</td>
<td>Off</td>
<td>Self refreshed</td>
<td>PLL shut down</td>
</tr>
<tr>
<td>G1/S3 (Suspend to RAM)</td>
<td>Off</td>
<td>Self refreshed</td>
<td>PLL shut down</td>
</tr>
<tr>
<td>G1/S4 (Suspend to disk)</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td>G2/S5 (S5 off)</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td>G3 (Metrical off)</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
</tr>
</tbody>
</table>

Transmeta Crusoe Power

Crusoe Processor Typical Power Dissipation - Model TM5400

<table>
<thead>
<tr>
<th>Parameter</th>
<th>500-700 MHz 1.2-1.4V</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVD operating power</td>
<td>1.8 W</td>
<td>1.2</td>
</tr>
<tr>
<td>MP3 operating power</td>
<td>1.8 W</td>
<td>1.2</td>
</tr>
<tr>
<td>Auto Halt power</td>
<td>0.9 W</td>
<td>1.4</td>
</tr>
<tr>
<td>Quick Start power</td>
<td>0.3 W</td>
<td>1.5</td>
</tr>
<tr>
<td>Deep Sleep power</td>
<td>0.03 W</td>
<td>1.6</td>
</tr>
<tr>
<td>Off / Instant On power</td>
<td>0 W</td>
<td>1.7</td>
</tr>
</tbody>
</table>

OSDM: OnNow

SetSystemPowerState
- initiate sleep state, query apps(?)

SetThreadExecutionState
- specifies level of support needed (e.g. display required)

WM_POWERBROADCAST
- a message notifying of power state changes to which applications can respond

SetWaitableTimer
- ensure PC is awake at scheduled time

RequestDeviceWakeup

RequestWakeupLatency - to specify latency requirements

GetSystemPowerStatus and GetDevicePowerState
PowerScope [Flinn]

- Statistical sampling approach
  - Program counter/process (PC/PID) + correlated current readings.
  - Off-line analysis to generate profile

- Causality
  - Goal is to assign energy costs to specific application events / program structure
  - Mapped down to procedure level
  - System-wide.
    Includes all processes, including kernel

System Monitor Kernel Mods

- NetBSD
- recording of PC and PID
- fork(), exec(), exit() instrumented to record pathname associated with process
- new system calls to control profiling
- pscope_init(), pscope_start(), pscope_stop(), pscope_read() (user-level daemon, to disk)

Experimental Setup

Data Gathering

Multimeter’s clock drives sampling at period of 1.6ms

- Trigger computer

Profiling computer

- Takes current sample

- Trigger next sample

- Profile

- User-level daemon writes to disk when buffer 7/8 full

Energy Analyzer

- Voltage essentially constant, only current recorded.
- Each sample is binned into process bucket and procedure within process bucket.
- Energy calculated by summing each bucket

\[
E = V_{\text{meas}} \sum_{\text{proc}} I \Delta t
\]
Case Study

Video application
original 12.1MB

- Step 1: lossy compression
  B: 7MB, C: 2.8MB
- Step 2: display size reduced from 320x240 to 160x120
  A\textsubscript{small}: 4.9MB, C\textsubscript{small}: 1MB
- Step 3: WaveLAN put into standby mode when not used
- Step 4: Disk powered off