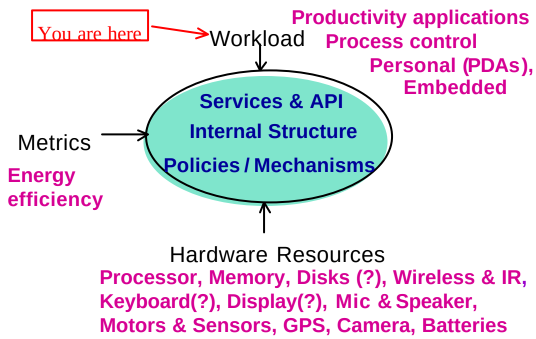
Rethinking OS Design



Workloads & Benchmarks

To discuss workload and measurement issues:

- *representative* benchmarks of workload
- *appropriate metrics* for target workload
- ability to *measure*

Traditional Benchmarks

- Microbenchmarks (Winbench, lmbench)
  - what they do isn't useful by itself. Example: null message roundtrip time
- Synthetic benchmarks
  - artificial programs (possibly designed to match observed behavior patterns). Often to get "controlled" experiments.
  - correlate with user experience?
- Application suites (SPEC95, Sysmark NT)
  - Set of "real" programs. Selection?
  - Inputs that drive them (scripted user interactions)?
  - Source or no source? Instrumentation?

Itsy Results

		Itsy							ThinkPad
#	Micro-Benchmark	Processor Voltage	Power (Watts) at Specified MHz			Energy (Joules) at Specified MHz			Power (Watts)
			59.0	132.7	206.4	59.0	132.7	206.4	
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
1	Sleep mode	normal	0.010	0.010	0.011	—	—	—	0.322
2	Idle mode	normal	0.094	0.128	0.162	—	—	—	3.30
3	Idle mode, LCD enabled	normal	0.134	0.164	0.198	—	—	—	5.15-7.74
4	Busy wait	normal	0.276	0.412	0.606	—	—	—	8.30
5		reduced	0.177	0.324	0.482	—	—	—	
6	Busy wait, LCD enabled	normal	0.263	0.447	0.637	—	—	—	10.4-13.1
7		reduced	0.217	0.363	—	—	—	—	
8	Addition loop	normal	0.314	0.612	0.899	6.388	5.834	5.220	7.43
9		reduced	0.247	0.490	0.729	5.030	4.457	4.181	
Memory Test with instruction cache, MMU, write buffer and data cache enabled									
10	In-cache read test	normal	0.385	0.763	1.123	0.228	0.201	0.191	7.81
11	Out-of-cache read test	normal	0.458	0.719	0.777	8.422	6.419	6.880	8.74
12	In-cache write test	normal	0.383	0.762	1.139	0.227	0.200	0.189	7.43
13	Out-of-cache write test	normal	0.731	1.290	1.573	3.884	3.555	3.877	7.20
Memory Test with only instruction cache enabled									
14	In-cache read-test	normal	0.504	0.800	1.023	3.717	3.291	3.187	
15	Out-of-cache read-test	normal	0.523	0.840	1.063	3.853	3.415	3.330	
16	In-cache write test	normal	0.596	1.076	1.183	3.838	3.018	3.617	
17	Out-of-cache write test	normal	0.596	1.076	1.183	3.574	3.013	3.617	

Application Suites

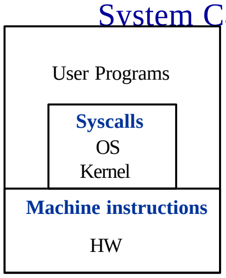
- Fixed demand - constant demand over time. Multimedia. Example: DVD player
  - sufficiency
- Variable demand - interactive. Productivity applications with user input: Word[tm], Excel[tm], Netscape[tm], PhotoShop[tm] Entertainment: Tetris, Quake, MP3 player
  - common to model infinitely fast user
 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



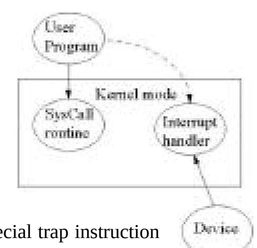
UNIX

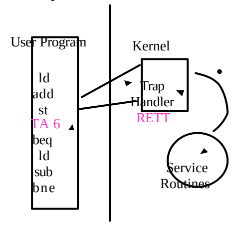
- fork, exec, exit, join
- open, close, read, seek

PalmOS

- EvtGetEvent
- MemHandleLock
- SndPlaySystemSound

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



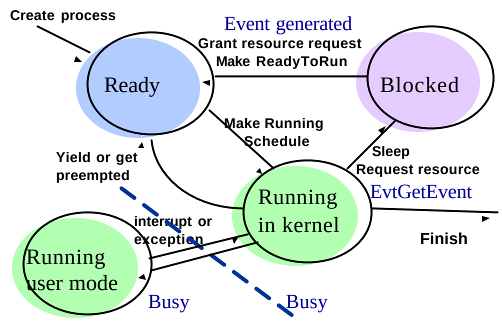


- Special trap instruction that saves essential context, causes a mode change and transfers to a handler.

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*
- Operations on Processes: fork (clone-style creation), wait (parent on child), exit (self-termination), signal, kill. Process-related System Calls.

Process State Transitions



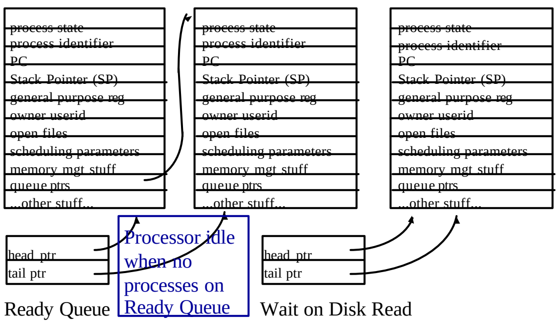
Process Mechanisms
Context Switching

- 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 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).

PCBs & Queues



(Traditional) Unix Abstractions

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

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

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 returns a zero to the child and the child process ID to the parent.*

Fork creates an exact copy of the parent process.

```
int pid;
int status = 0;

if (pid = fork()) {
        /* parent */
        …..
        pid = wait(&status);
} else {
        /* child */
        …..
        exit(status);
}
```

*Parent uses **wait** to sleep until the child exits; wait returns child pid and status.*

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

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

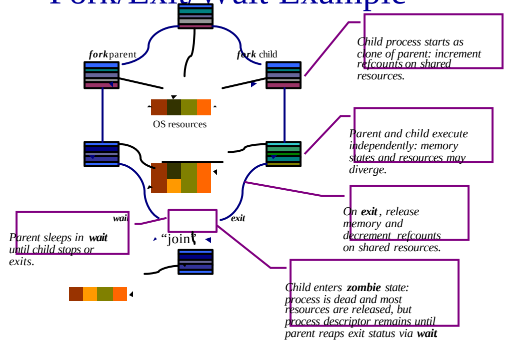
Child Discipline

- 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



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

signal == "upcall"

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

```
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");
}
```

Sets up signal handler

Instructs kernel to send SIGALRM in 3 seconds

Suspends caller until signal

Yet Another User's View

```
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;
}
```

Collects status

SIGCHLD sent by child on termination; if SIG_IGN, dezombie

File System Calls

*Open files are named to by an integer **file descriptor**.*

```
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 passes status back to parent on **exit**, to report success/failure.*

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

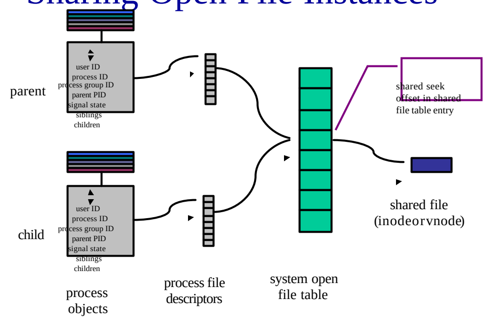
*Standard descriptors (0, 1, 2) for input, output, error messages (**stdin, stdout, stderr**).*

File Sharing Between Parent/Child

```
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);
        }
}
```

[Bach]

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(), ulink(), mkdir(), rmdir().

Devices

Various devices are abstracted as *special files*.

- Named by a filename.
- Accessed via open(), close(), read(), and write()
- Idiosynchratic 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)

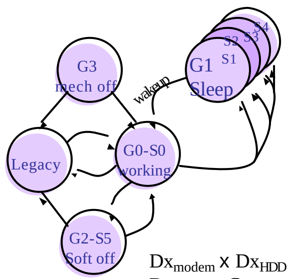
ACPI
Advanced Computer Power Initiative

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



$Dx_{modem} \times Dx_{HDD} \times Dx_{cdrom} \times Cx_{cpu}$

Transmeta Crusoe ACPI Power States

ACPI System State		Processor Power State	DDR, SDR SDRAM	Clock Generator
G0/S0 (Working)	C0	Normal	Normal	Running
	C1	Auto Halt	Normal	Running
	C2	Quick Start	Self refresh	Running
	C3	Deep Sleep	Self refresh	Clocks stopped
G1/S1 (Sleeping)		Deep Sleep	Self refresh	PLL shut down
G1/S2 (Suspend to RAM)		Off	Self refresh	PLL shut down
G1/S3 (Suspend to RAM)		Off	Self refresh	PLL shut down
G1/S4 (Suspend to disk)		Off	Off	Off
G2/S5 (Soft off)		Off	Off	Off
G3 (Mechanical off)		Off	Off	Off

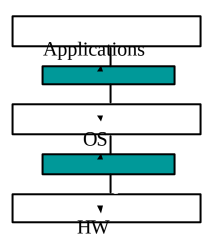
Transmeta Crusoe Power

Crusoe Processor Typical Power Dissipation - Model TM5400

Parameter	500-700 MHz 1.2-1.6V	Notes
DVD operating power	1.8 W	1,2
MP3 operating power	1.0 W	1,3
Auto Halt power	0.9 W	1,4
Quick Start power	0.3 W	1,5
Deep Sleep power	0.03 W	1,6
Off / Instant On power	0 W	1,7

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



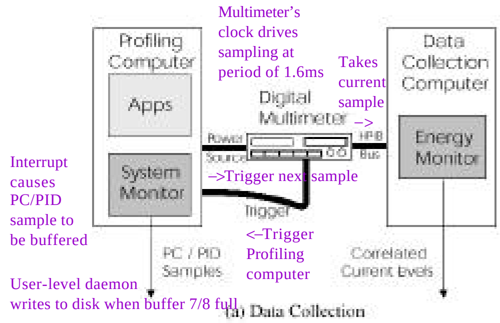
RequestWakeupLatency - to specify latency requirements

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

Experimental Setup
Data Gathering



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)

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

$$E = V_{meas} \sum_{t=0}^{n} I_t \, \Delta t$$

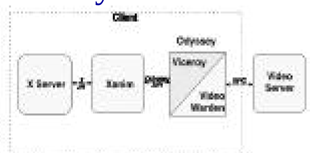
Process	Elapsed Time (s)	Total Energy (J)	Average Power (W)
/usr/odyssey/bin/xanim	66.57	643.17	9.66
/usr/X11R6/bin/X	35.72	331.58	9.28
/netbsd (kernel)	50.89	328.71	6.46
Interrupts-WaveLAN	18.62	165.88	8.91
/usr/odyssey/bin/odyssey	12.19	123.40	10.12
Total	183.99	1592.75	8.66

Energy Usage Detail for process Interrupts-WaveLAN

Kernel-level procedures:

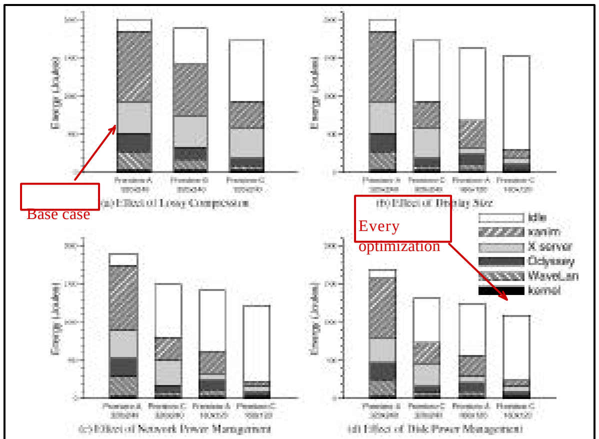
Procedure	Elapsed Time (s)	Total Energy (J)	Average Power (W)
_xferDMAbuffer	16.66	147.38	8.85
_pwlread	0.30	2.90	9.65
_pwlget	0.30	2.68	8.93
_pwlintr	0.24	2.31	9.62

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_{small}: 4.9MB, C_{small}: 1MB
- Step 3: WaveLAN put into standby mode when not used
- Step 4: Disk powered off



(a) Effect of Lossy Compression
(b) Effect of Display Size
(c) Effect of Network Power Management
(d) Effect of Disk Power Management