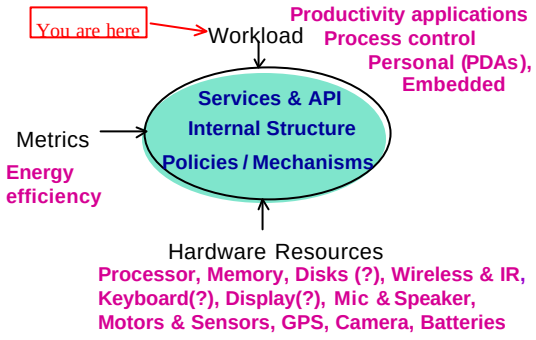
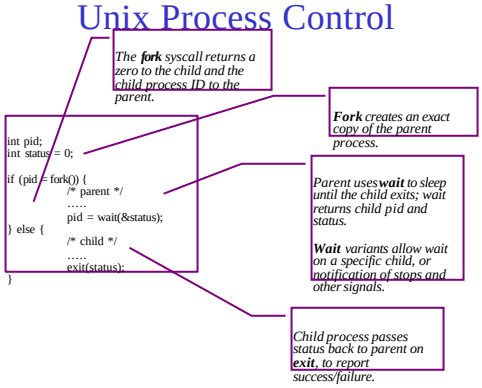
Rethinking OS Design



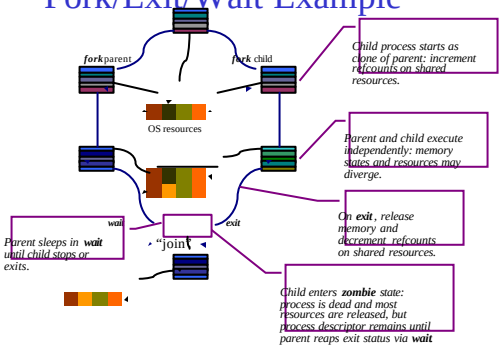
(Traditional) Unix Abstractions

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

Unix Process Control



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

```
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);
        }
}
```

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

*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.*

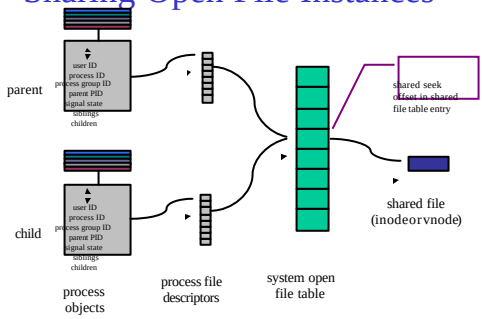
*Standard descriptors (0, 1, 2) for input, output, error messages (**stdin**, **stdout**, **stderr**)*

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.

OSDM: OnNow

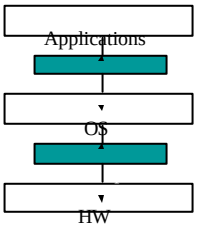
SetSystemPowerState
- initiate sleep state, query apps(?)

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

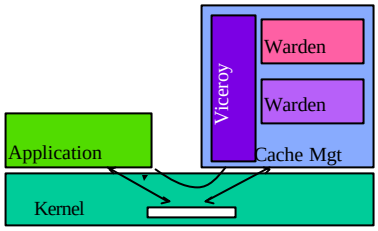
Framework for Adaptation

- Odyssey project - Satya (CMU)
- Odyssey is an attempt to incorporate *application-aware adaptation*
- Noble et al, *Agile application-aware adaptation for mobility*, SOSP 97
  (network bandwidth examples)
- Flinn and Satya, *Energy-aware adaptation for mobile applications*, SOSP 99
  (energy usage examples)

Odyssey Provides

- API - new syscalls to register a window of tolerance for a variable resource (e.g. network bandwidth)
- Notifications of change (upcalls)
  - Implies *detection* of changes. Mechanisms needed.
- Typing - Wardens which handle type-specific functionality
  - Type awareness necessary to evaluate tradeoffs
- Viceroy – centralized resource coordination

Architecture of Odyssey Client



API

Example

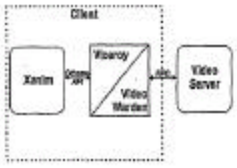


Figure 4: Video Player in Odyssey

- Each movie in multiple tracks at different fidelity levels
- Warden can switch between tracks to fit bandwidth requirements

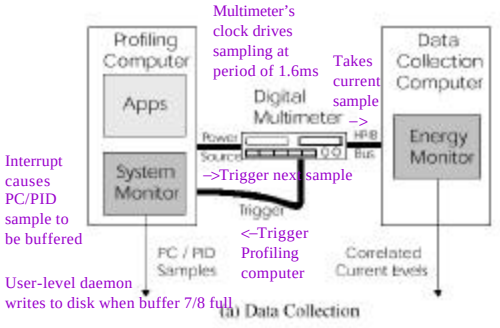
Energy Resource

- Monitoring to detect resource availability Powerscope
- Using Odyssey for adaptation in this domain.

PowerScope [Flinn] as a Tool

- 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



(a) Data Collection

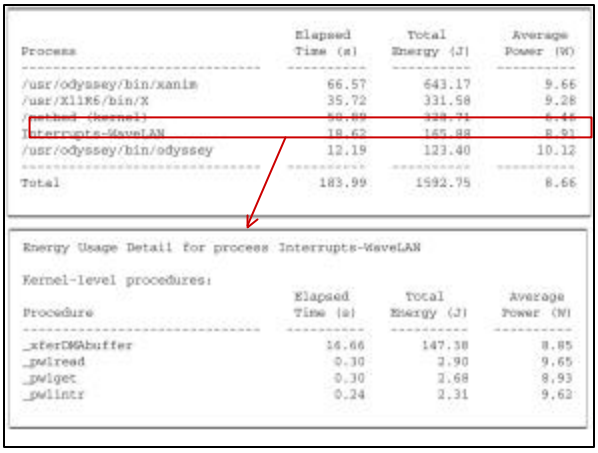
System Monitor Kernel Mods

- 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
/vmlinuz (kernel)	50.89	338.71	6.66
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

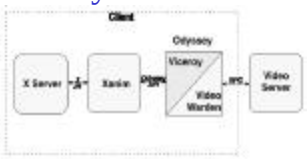
Kernel-level procedures:

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

Fidelity for Energy?

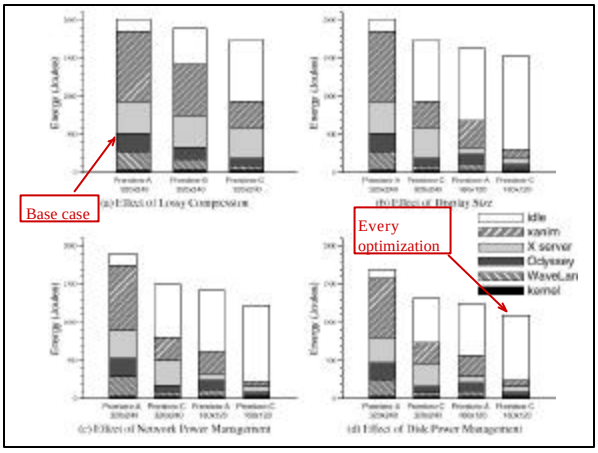
- Before investing in incorporating energy into Odyssey for adaptation, first determine whether Odyssey's model of fidelity as the way to adapt has potential for energy savings.
- Experiments showing that potential, hand-tuned based on Powerscope information.

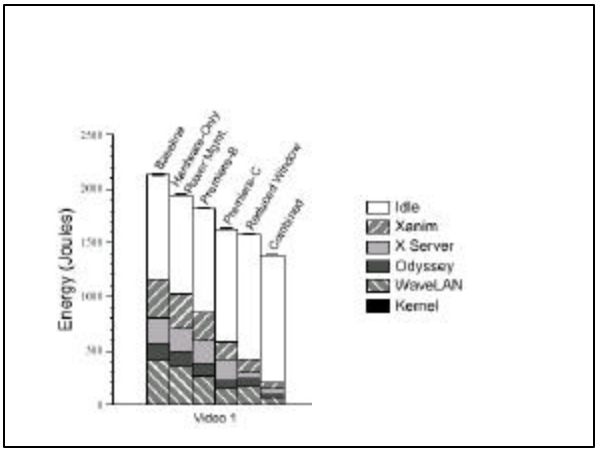
Case Study



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





Conclusions about Fidelity as Energy Saving Adaptation

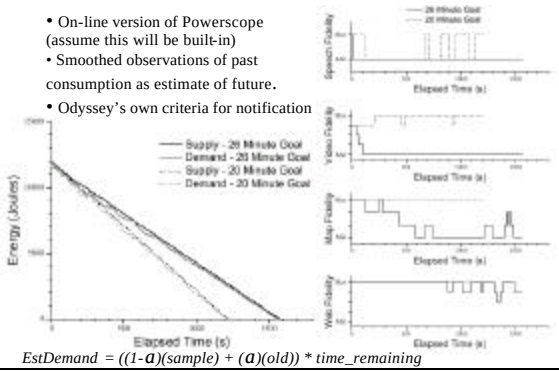
- Significant variation in effectiveness of fidelity reduction among objects
- And among applications
- Combining hardware power management with fidelity reductions is good.

Can Odyssey Automate This?

- User specifies target battery lifetime.
- Odyssey is to monitor energy supply and demand
- Notify applications to change fidelity if estimate future demand and supply don't match to achieve desired lifetime.

Goal-directed Energy Adaptation

- On-line version of Powerscope (assume this will be built-in)
- Smoothed observations of past consumption as estimate of future.
- Odyssey's own criteria for notification



*EstDemand = ((1-α)(sample) + (α)(old)) * time_remaining*

Results

- Goals:
  - Meet specified battery lifetime
  - Highest fidelity within that constraint
  - Infrequent adaptations
  - Small leftover battery capacity at end of period.

Specified Duration (s)	Goal Met	Residue		Number of Adaptations			
		Energy (J)	Time (s)	Speech	Video	Map	Web
1200	100%	145.2 (25.3)	15.3 (1.5)	10.8 (1.6)	11.0 (4.0)	0.4 (0.5)	0.0 (0.0)
1320	100%	107.5 (61.5)	12.9 (7.2)	2.8 (0.4)	28.2 (5.2)	1.6 (2.6)	0.0 (0.0)
1440	100%	101.2 (32.3)	13.0 (4.5)	5.0 (7.9)	22.6 (6.8)	9.6 (3.8)	1.2 (1.8)
1560	100%	60.2 (28.7)	8.7 (5.9)	1.0 (0.0)	6.0 (2.8)	15.4 (4.6)	7.6 (5.9)