8/29/2013 - “THE”

Multiprogramming vs batch

Hierarchy

Proof of correctness

Virtual – segments vs pages: segment here is fixed size (or multiple pages), particular address.

Virtual cpu

Semaphores

Processes

Testing

Software development

No multi task

No security

No prority

Higher level can use lower level, but inverse is not possible.

In real world the hierarchy is hard to achieve.

Network in: 比如 level3: 好处可以用其以下的，坏处其以下不可用network. 若不是用hierarchy，则可以用module，every dependent on other，但可以互相使用。

4: applications

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3:devices

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2:console : act like every process has its own console

(If exchange level 1 and 2, might let not so important message of console occupy in the core page)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1： segment: physical memory

\_\_\_\_\_\_\_\_\_

0: Cpu process (resource management, which process run at cpu, abstraction of process, real-time clock)

9/3/2013 – Multics

Large virtual memory

Dynamic linking – sharing of code

Virtual addresss

Segments:

Data & code for each process, variable size, access list, can have >= 1 symbolic name e.g. “/bin/data”, single level store(no distinction between file and segments), distinct, grow and shrink

Processes:

Composed of segments, each process has its own single address space

Stack & stack pointer

Weakness:

Overhead (e.g. caused by dynamic linking)

Fragmentation

Paging algorithm

**Segments: composed by pages, has a page table for the pages of the segment.**

9/5/2013 – Hints

Simplicity

Speed

Don’t hide the power of underlying things

Procedure arguments

Leave it to the client

.VAX e.g. CISC

.i432 (Intel 432)

Keep interfaces stable

.backwards compatible

Plan on rewriting

**Separate normal & worst cases: ! Very important**

Make things static: static allocation is easier than dynamic allocation

**Caching**

– keep the answer

**Hints**

– hint the answer(this may be wrong), if want exactly right it is too expensive thus we use hints.

Hints is kind of speed up.

Hints need cheap verification

Better mostly right

**Watch constant factors**

.Brute force

.simple

**Do work in the background:** most files do not exist very long, may wait not to write them into disks.

**Batch**

Avoid mistakes

Shed load:

drop out of line if it works bad, if might give other users get the computer works better, don’t do a job on everything

**end-to-end:**

.application-level checks for correctness

.lower-level checks for performance

**Logging**:

.atomic or restartable actions

Dynamic translation

JIT compiling

VMWare

09/10/2013

LFS

Keep it simple

Logs

Caching

Crash recovery

Work in the background

Batching

Separate normal from worst cast

Fast indexing-hint

How to speed up disk performance?

.do not use the disk.

Don't use the disks

Caching

Use the disk efficiently

.scheduling algorithm

Lag data out on disk intelligently

FFS-spatical locality

e.g. all blocks of a disk at the time the file

a.. files in the same directory is written

LFS-temporal locality

.blocks written together are read together written

technology trends:

.CPUus are getting faster than that of disks

.big memories

.disk arrays – small writes

.SCSI- variable diisk georetics

LFS – treat the disk like a tape

* + 1. problems:

1. what happens when the disk fills up?

Solution:

* + - * 1. buy a new disk – Plan 9 file system

2. how do I read from the disk.