

Announcements

- Assignment 1 will be returned in labs this week and marks will become available in **StReaMS** tomorrow (Tuesday)
- Some students will not see any marks (but will receive an e-mail...) :-(
 - Lab 5 this week (PeANUt bit operations and address parameters, and PeANUt virtual memory)
Preparation is important for this lab - download, print and read the lab sheet before you go into the lab!
- Lab 5 is the **last** supervised PeANUt lab

Virtual memory in PeANUt

- Virtual memory implementation
 - Page table
- Virtual memory in PeANUt
 - Page replacement
 - Paging behaviour
 - Working set
- References:
 - Specification of the PeANUt computer (Section 3)
- Additional reading: 10.1 – 10.7 in Bryant and O'Hallaron

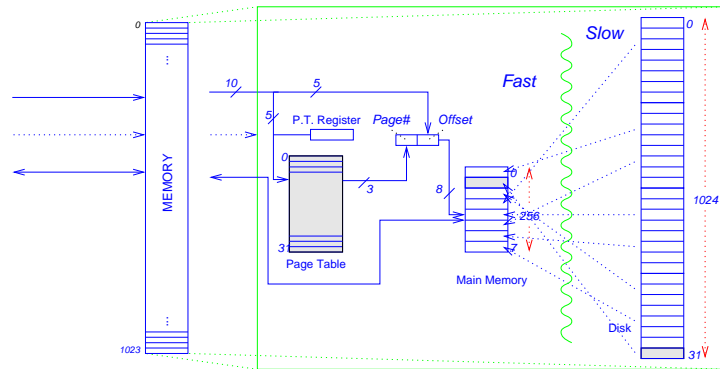
Virtual memory in PeANUt (1)

- Our previous model of the PeANUt considered memory to be a *black box*
 - Data was written or read by the memory automatically
 - Suppose paging is implemented: How would you know if the data is in main memory or on the disk?
- A closer look at PeANUt memory
 - Main memory (MM): 256 cells, in 8×32 -cell page frames
 - Secondary memory (SM): 1024 cells, in 32×32 -cell pages

Virtual memory in PeANUt (2)

- Memory Management Unit (MMU)
 - Not explicitly shown in PeANUt architecture
 - Uses a page table with 32 entries (one per page)
 - Translates virtual addresses into physical addresses
 - Moves pages between main memory and disk whenever necessary

Virtual memory in PeANUt (3)

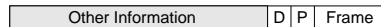


5

COMP2300, 2006

Page table

- Purpose: Give information on the status of each of the pages of the virtual memory
 - Is it present in main memory? **P**resent bit
 - If so, in which page frame is it? Frame number field (3 bits)
 - If so, has data been written to it? **D**irty bit



- Where does this page table reside?
 - Possibly in special hardware
 - In PeANUt: Part of main memory, at VM addresses **0–31** (large: 1/8 of main memory!)

6

COMP2300, 2006

How does PeANUt virtual memory work?

- To read some data (at say address **00101 01010**)
 1. The address is split into two parts – a **page number (00101)** and a **page offset (01010)**
 2. The status of the page is checked with the page table
 3. A) If page is in main memory (**P** set):
 - Produce its page frame address (say page frame 3 (**011**))
 - B) Otherwise, a page fault occurs:
 - Fetch page from secondary memory into main memory, possibly removing another page
 - Find the new page frame address of the page
 4. Combine page frame address with the page offset to produce an 8 bit address (**011 01010**), giving the location of the cell in main memory
- To write some data: Almost the same procedure as above (must also set the **dirty bit (D)** in the page table)

7

COMP2300, 2006

Page replacement

- How do we decide which page to throw out?
- Random (cannot get any performance advantages from **locality of reference**)
- **First In First Out (FIFO)**
 - Hardware keeps track of **age** of each page
 - Replace the **oldest** page
 - Problem: Any page continuously required must eventually be thrown out!
- **Least Recently Used (LRU)**
 - Hardware keeps track of **access times** to each page
 - Replace the **least recently used** page
 - Generally performs quite well

8

COMP2300, 2006

Paging policy failure

- A case of: Access pages 0, 1, 2, 3, 4 cyclically

0	4	3	2	
...	1	0	4	3
	2	1	0	4
	3	2	1	0

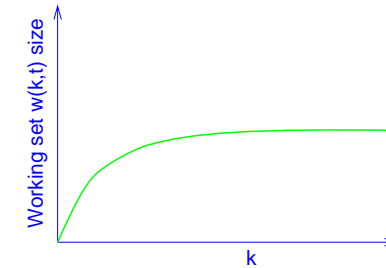
 ... Page table updates
- Here, five pages are needed continuously, but there are only four **page frames**
- Under both **LRU** and **FIFO** policies, the page required next is removed!
 - This problem can only be solved if future **page demand** can be predicted
 - Problems are inevitable when the demand for pages exceeds the number of page frames available

9

COMP2300, 2006

Working set (1)

- The **working set function** $w(k, t)$ is defined as the set of pages accessed during the k most recent memory accesses, up until time t

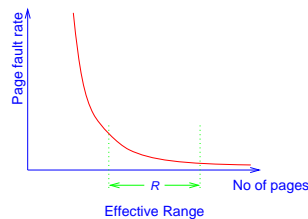


11

COMP2300, 2006

Paging behaviour

- How many pages? (or, how big are the pages?)
 - Too few (too big) can result in un-needed words put in main memory, resulting in **page faults**
 - Too many results in huge **page tables** (and more frequent page faults)



Assumption: page demand < available main memory

10

COMP2300, 2006

Working set (2)

- As k increases, the size of the working set function $w(k, t)$ approaches a (constant) limit
 - This limit is the **working set size** at time t
 - It is roughly independent of t
- The working set size concept can be used to describe the memory needs of a program
 - If the main memory is not large enough to hold the working set, there will be excessive **page faults** (such behaviour is called **thrashing**)

12

COMP2300, 2006

Page table in PeANUt

- Is permanently located in page 0 (addresses **a0** to **a37**)
- It has 32 entries, each relating to one of the 32 VM pages
- 5 bits of a memory address are used to determine the page number, 5 bits for the offset within the page
- Format per page table entry:

Unused	Last used	Swap count	D	P	Frame number				
15	11	10	8	7	5	4	3	2	0

PeANUt virtual memory traps

- **trap #11** (page fault)
Is not user-initiated or user-definable. Can occur in the fetch of an instruction or the evaluation of its operand
- **trap #12** (swap page in)
Move page number **AC** from secondary memory to main memory. The O/S will use page table entry **AC** (at **Mem[AC]**) to determine the destination page frame
- **trap #13** (swap page out)
The O/S will move page number **AC** from main memory to secondary memory

PeANUt page table fields

- **Last used:** Reflects how recently the page has been accessed, a value of **0** indicates that this is the most recently accessed page
- **Swap counts:** Indicates how many page faults have occurred since this page has been loaded into memory (tells how long a page has been in memory for)
- **Dirty flag:** A value of **1** indicates that data has been written to this page since it was swapped into memory
- **Present flag:** A value of **1** indicates the page is present in main memory, if it is not the value is **0**
- **Frame number:** This states which page frame (slot in main memory) a page is in **if** the page is present