Basics of Memory Management

- Virtual memory - a memory management technique which allows the programmers with illusion of an address space which is not shared with other processes, and can be substantially larger than the physical main memory.

- Address space of programs is divided into *pages*

- Physical memory is divided into *page frames*

- Pages and Page frames have fixed size, which is also identical
More Basics

• Virtual memory - address space of a program

• Physical memory

• Need virtual to physical address translation for every memory operation

• Should be fast !!

• *Page fault:* Need to allocate physical pages - use *page replacement algorithm*
OSP Specifics

- Every process has an associated page table, pointed to by page table base register (PTBR)
- Physical memory is divided into MAX_FRAME page frames of size PAGE_SIZE each
- Divide virtual address by PAGE_SIZE to get virtual page number
struct page_entry_node {
    int frame_id; /* frame id holding this page */
    BOOL valid; /* page in main memory: valid = true;
                  not: false */
    int *hook; /* can hook up anything here */
};
More OSP data structures

struct frame_node {
    BOOL free;       /* = true, if free   */
    PCB *pcb;
    int page_id;     /* virtual page id - an index to the 
                      PCB's page tbl */
    BOOL dirty;
    int lock_count;
    int *hook;
};
More OSP data structures

struct page_tbl_node {
    PCB    *pcb;     /* PCB of the process in question */
    PAGE_ENTRY page_entry[MAX_PAGE];
    int    *hook;    /* can hook up anything here */
};
Memory module functions

void memory_init()

void prepage pcb;

PCB *pcb;

int start_cost pcb;

PCB *pcb;

void deallocate pcb;

PCB *pcb;
Memory Module Functions Contd.

void get_page(pcb, page_id)
          PCB *pcb; int page_id;

void lock_page(iorb)
           IORB *iorb;

void unlock_page(iorb)
           IORB *iorb;

void refer(logic_addr, action)
           int logic_addr; REFER_ACTION action;
External Functions

extern siodrum(/* action, pcb, page_id, frame_id */);

/* IO_ACTION action;
   PCB *pcb;
   int page_id, frame_id; */

extern int get_clock();

extern gen_int_handler();
Simple functions

Refer(): called by SIMCORE everytime there is a memory operation

--> must convert virtual to physical address
--> set dirty bit
--> call page fault handler if required

lock and unlock pages: maintain count of devices that have a page locked, increment/decrement the counter, lock may have to bring the page into memory
Page Fault

struct int_vector_node {
    INT_TYPE cause;   /* cause of interrupt */
    PCB *pcb;        /* pcb that caused page fault */
                    (if pagefault interrupt) */
    int page_id;     /* page causing pagefault */
    int dev_id;      /* device causing devint */
    EVENT *event;    /* event involved in waitsvc and sigsvc calls */
    IORB *iorb;      /* IORB involved in iosvc call */
};
Page Fault

-- Int_Vector.cause is set to pagefault
-- Int_Vector.page_id is set to the page that causes the fault
-- Int_Vector.pcb is set to the pcb of the process owning the page
-- General interrupt handler is called
Get a Page
	his is the main place where page allocation and replacement decisions are made

if free frame is available, simply copy
else,
    need to deallocate a page
    check dirty bit is copying back is required
    lock_count of a page replaced must be zero
Dirty (Laundary)

dirty bit is set to false when a page is initially allocated
set to true in two cases:

---> refer, when the action is store
---> lock_page, when iorb.action is read
---> need to write back a dirty page
The Remaining Business

prepage pcb - leave blank

start_cost pcb - always return 0

deallocate pcb read from the book :-}
Lab 3 Description

- Implement BSD 4.4 page replacement algorithm in OSP
- Have a complete and working OSP memory module
- Compare performance with OSP.demo
BSD 4.4 Basics

- Three list of pages: *wired*, *active*, and *free*
- Wired is the same as locked pages, they cannot be moved out
- By maintaining some pages to be free, avoid page replacement
- Identify pages that can be freed from the active list
More details

- Maintain active pages in a linked list - least recently used
- Can easily find the pages that have not been touched recently
- Allocating a Page on a Page Fault
  - Find a free page
  - If too few free pages, process the list of active pages
  - Move pages from active list to free list if not referenced - check
    if locked (do nothing) and dirty (write-back)
BSD 4.4 things you cannot do

- Cannot swap out a process
- Cannot cluster pages that can be written together