Assignment 01 explanations:

Vector 2000 →

Proc 2 1000 →

Proc 1 500

File system

Q3.

sp = setup_stack (PL, sp)

deswitch (NULL, sp)

A →

PUSH Rx

POP Rx

11 Rx

*old = sp

Stack

sp

NULL

A

Rx

Ry

Rz

Assignment 02 explanations:

Grad

U?

Undergrad

G?
monitor

fields v1, v2

method 1()

logic 1

logic 2

method 2

v1, v2, --- global

mutex m

method 1()

lock(m)

logic

unlock(m)

- shouldn't have multiple mutexes floating around.
- Indeterminacy is there in real implementation of monitor.

Virtual Memory:

Last lecture we talked about address translation from virtual to physical address.

Some usage of virtual memory:

1. Use a page table to map whole physical memory to virtual, but it is of no non-trivial importance.
2. Use virtual memory for protection.

Program

Process

(C32-bit Linux)
Another process running the same program will have the same address space layout and offsets.

OS does two things to manage virtual memory:
- Setup page tables beforehand,
- Handle page faults, which allows for "lazy allocation."

<table>
<thead>
<tr>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW</td>
</tr>
</tbody>
</table>

Memory Map

- Regions

Page Table

Stack: (lo, hi)
- Zero-fill
- (extend down)

Page fault responses:
- Stack is a resource which needs to be extended and OS doesn't have any mechanism to handle this as page faults are hardware instruction dependent so a mechanism at hardware level is required to handle page fault.
- Hardware communicates to OS about page fault details to make OS look into memory map to resolve the page fault. ("lazy allocation")

- Some OSes: OSes allocate extra virtual memory to and point the extra pages to a zero-filled page and when a write happens on any of these pages, the page gets mapped to extra allocated memory. ("Write and Allocate")
- Two process's page tables pointing to same physical pages.

(COPY-ON-WRITE)
- Reference counting of pages is required to manage COPY-ON-WRITE

Fetch from storage:

demand-paging
OS: VM area → file
page-handler ⬆

offset = add-vma-base
allocate page
Fetch offset from vma-file

- On writable mapping from VM area → file OS has to manage upath in updating of file i.e. write the page back file if it has been modified and make it inaccessible to process while writing on file happens.

"Anonymous" Memory

vma

page file/partition

* saves memory b/c of partial loading of a program
As disk access time has reduced a lot and more VM is available. OS has been able to run large executable files by allocating memory for more frequently accessed part of the program.

There are two views of VM,

HW view :- TLB + Page Table which are hopefully in sync.
OS view :- which handles page faults and manages mapping of pages to program file on disk.
Lecture 8 2/5/10

Homework #1: relate back to 1st lectures. PIC, loading, code/stack, context switch, I/O. Mention put/get term error.

Homework #2: draw picture, example timeline. Draw flow chart of related problem. (2-Student)

Virtual memory - what to do with page translation?

- tlb map. bigness. allows permissions. page fault always crashes.

addr \rightarrow address space per process, context switch

ref addr \rightarrow options - fail, allocate, file I/O

proc \rightarrow mmu \rightarrow virtual addr... example file: (text, ro, rw, bss) \times pages, libs + files

1. zero fill 2. lazy allocate (stack) 3. demand load (executable)
4. paging (anonymous) 5. copy on write. a - bss, b - process

Digression - allocators. sort list? first, best, worst, next fit?

buddy allocator \leftarrow binary tree, thread size n. round to 2

paging gives us:

G.P. lang-independent allocator
multiplexing of memory
arbitrary functionality - e.g. compression