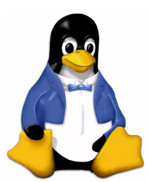
Managing memory in variable sized chunks

Christopher Yeoh <cyeoh@au1.ibm.com>

Linux.conf.au 2006



Outline

- Current Linux/K42 memory management
- Fragmentation problems
- Proposed system
- K42 Architecture/Implementation
- Experiment results
- Future work

Current OS memory management

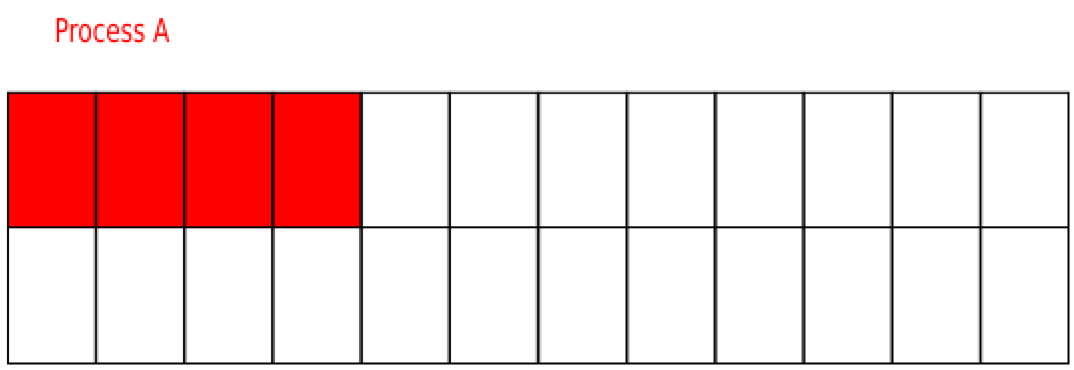
- Physical memory split into frames
  - Typically 4Kbytes
- Applications work with virtual addresses
  - OS manages
    - allocation of physical frames
    - loading of frame contents
    - virtual to physical address mapping

Virtual memory

- Virtual to physical mapping must be very fast
  - Hardware support
    - TLB
- Effect of caches degraded as memory sizes increase
  - Large pages increase effectiveness of caches
    - requires physically contiguous memory

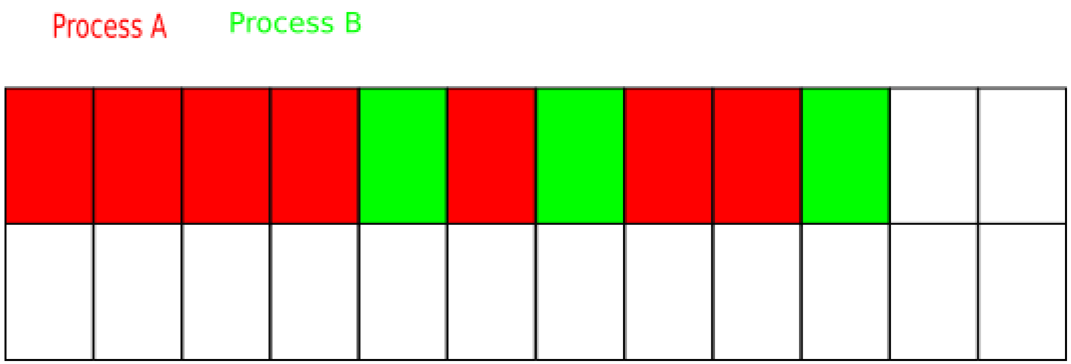
Causes of fragmentation

- Page allocation
  - Memory allocated a frame at a time for a process or file
- Fragmentation builds up over time as processes allocate memory and then exit

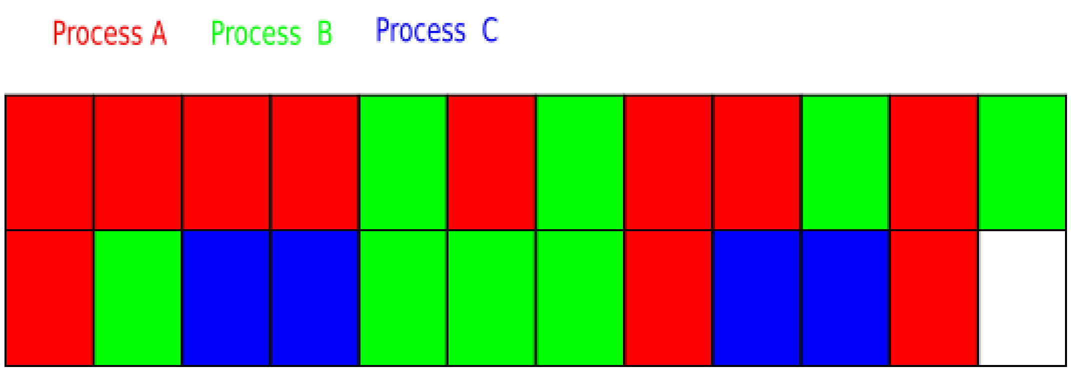


Fragmentation

- Process B starts

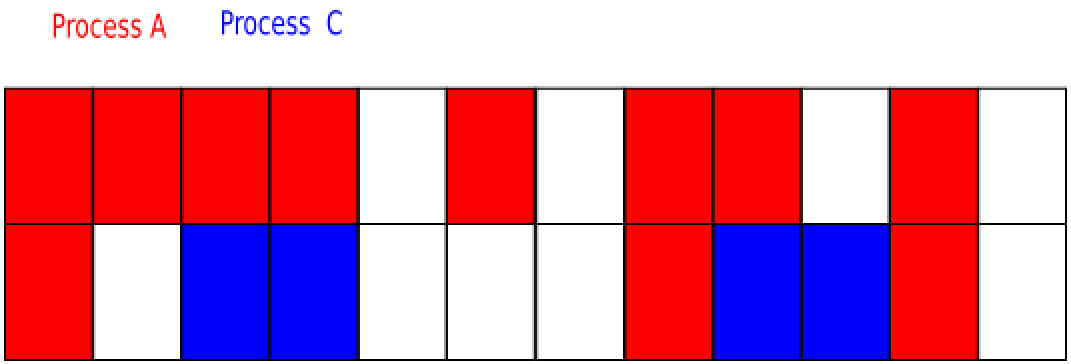


- Process C starts

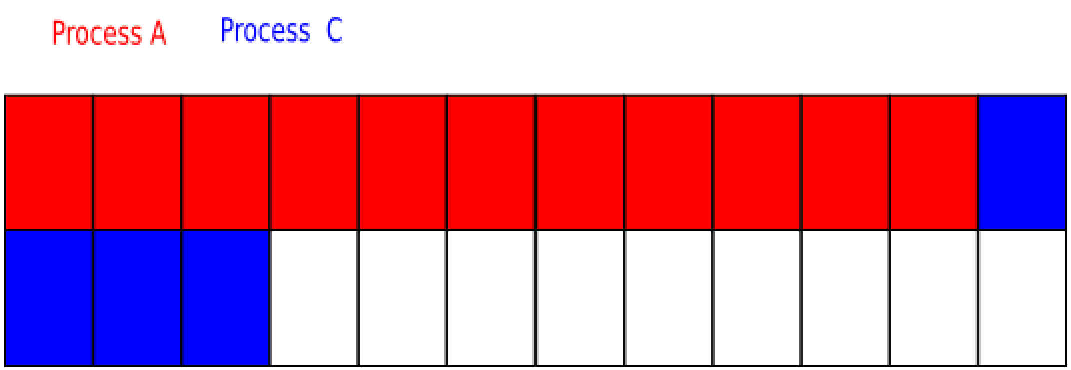


Fragmentation

- Process B exits



- Minimal fragmentation



Problems with fragmentation

- Can't allocate large pages
- Some device drivers need physically contiguous memory
- More difficult to hotplug memory
  - virtual machines as well as real physical memory
- One solution
  - Reserved areas
    - Fixed size
    - Balancing pools

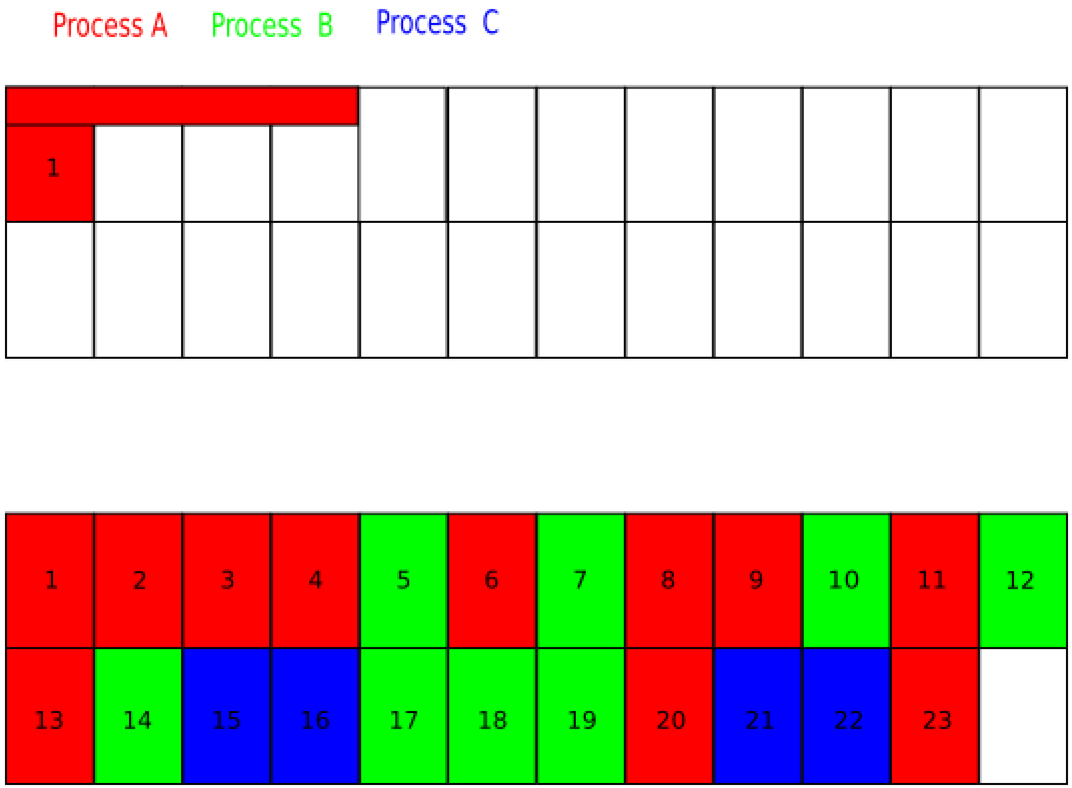
Problems with Fragmentation

- Current Linux approach
  - 3 zones
    - user reclaimable
    - kernel reclaimable
    - kernel non reclaimable
  - Fall-back allocation when a zone is exhausted
  - Copy memory around in reclaimable areas when too fragmented
- Some device drivers have to reserve contiguous physical memory
  - Module loading can fail

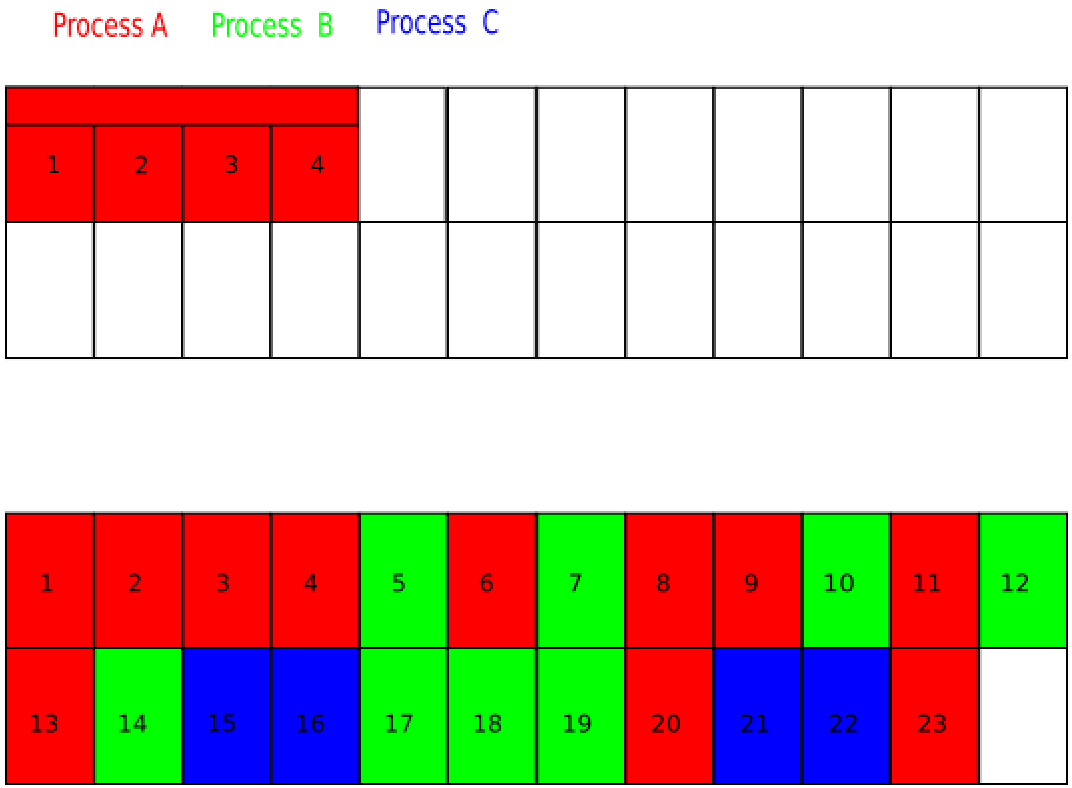
Proposed system

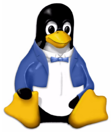
- Allocation of memory in chunks
  - Chunks allocated for processes/files
    - allocation for process done from chunk
    - when exhausted another chunk allocated
  - Chunks can be of variable size
- Allocate array of page descriptors which refer to a chunk

Chunk allocation

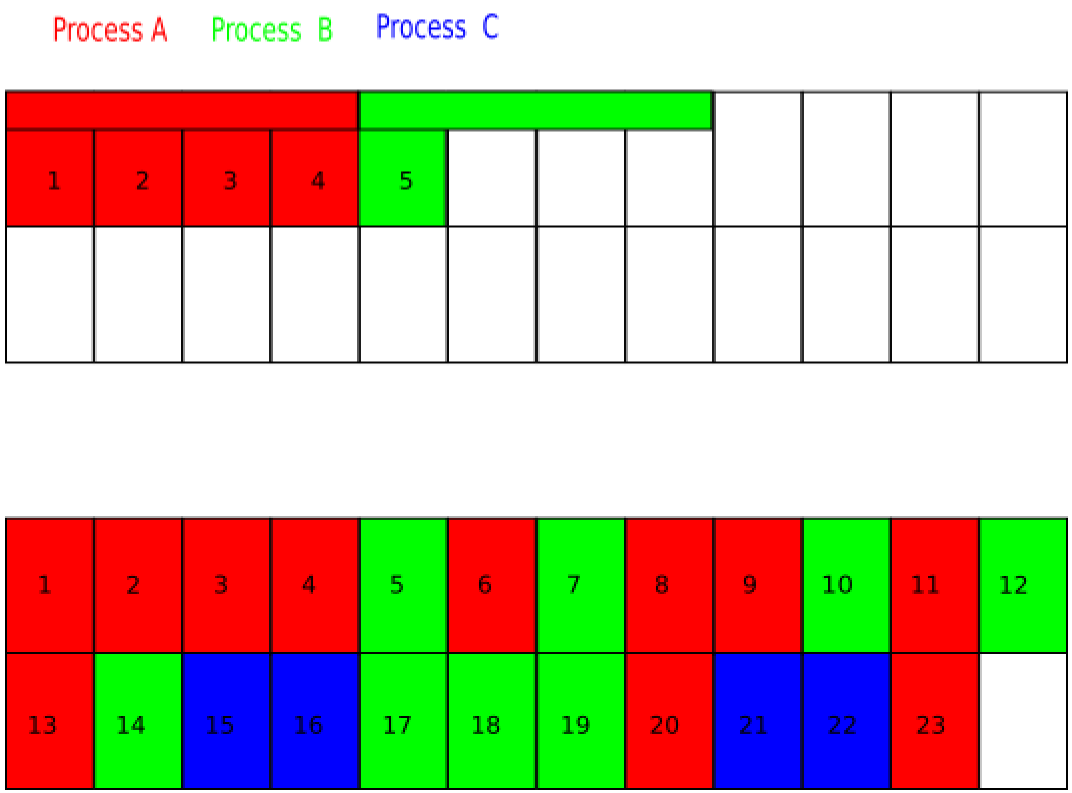


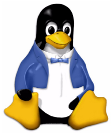
Chunk Allocation



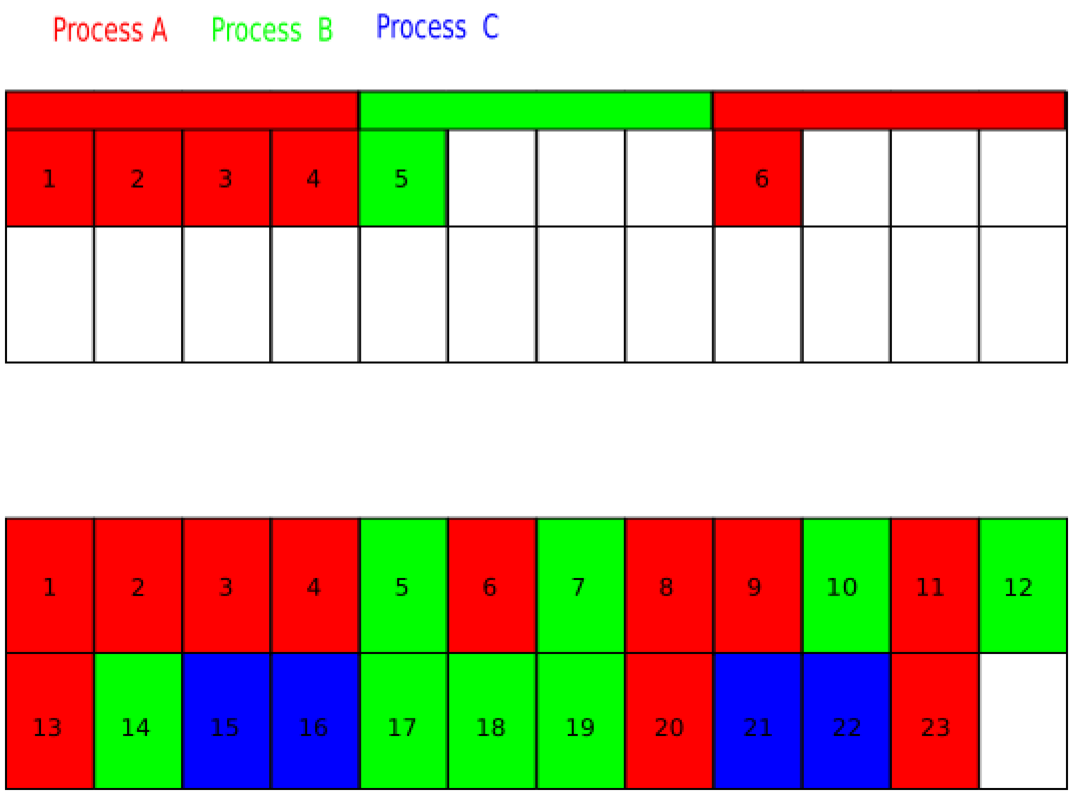


Chunk Allocation

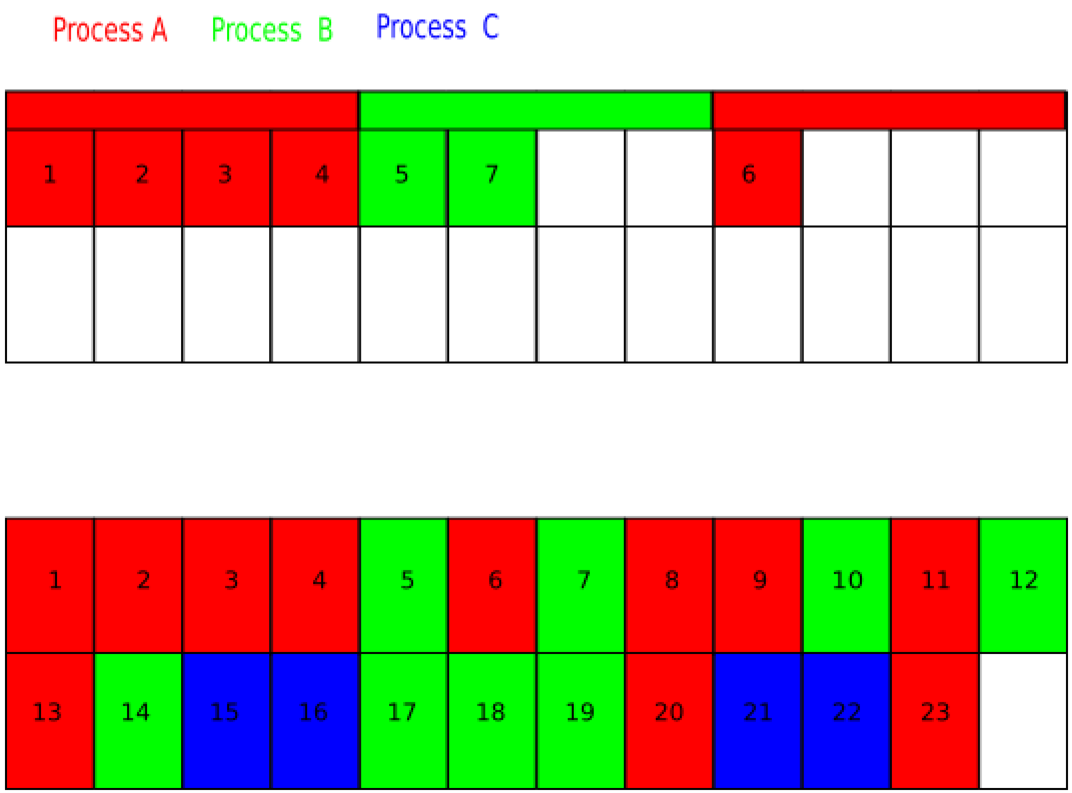




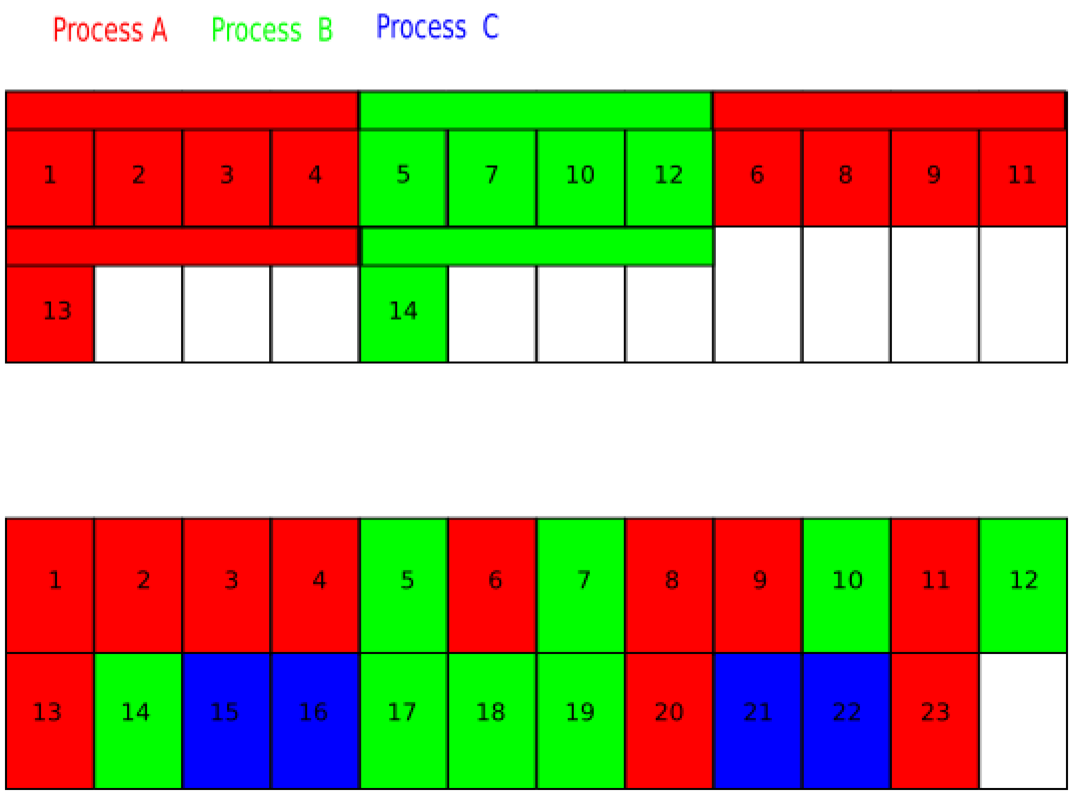
Chunk Allocation



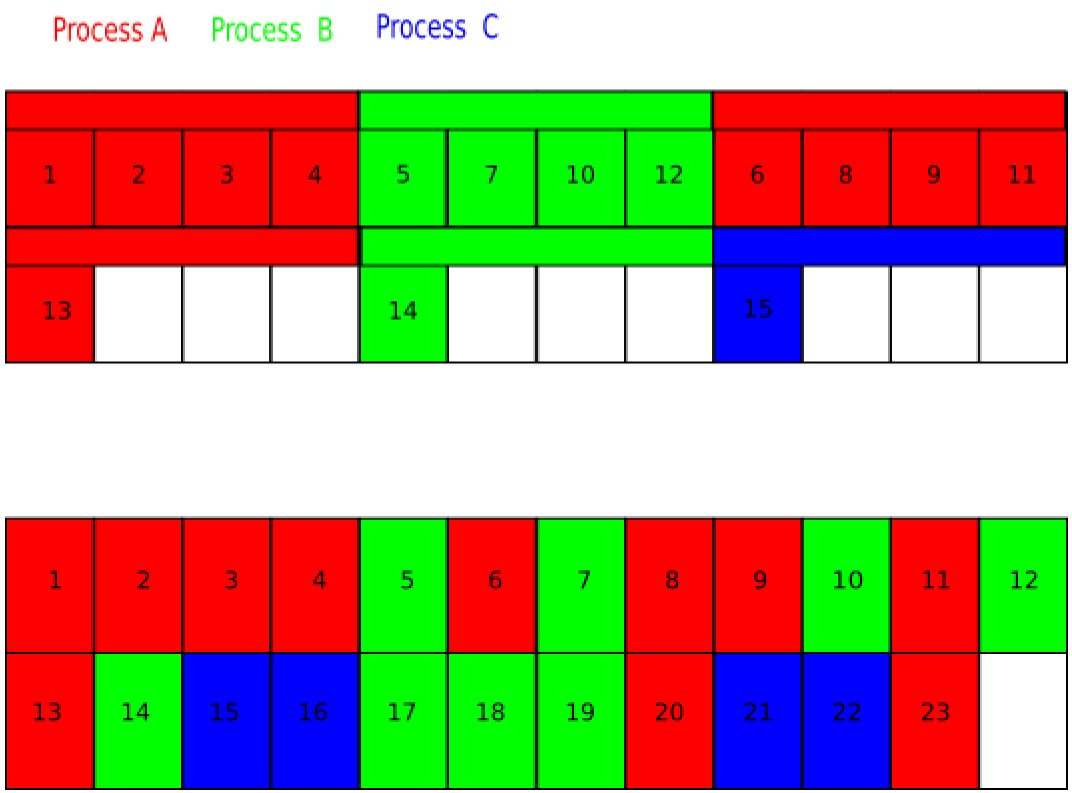
Chunk Allocation



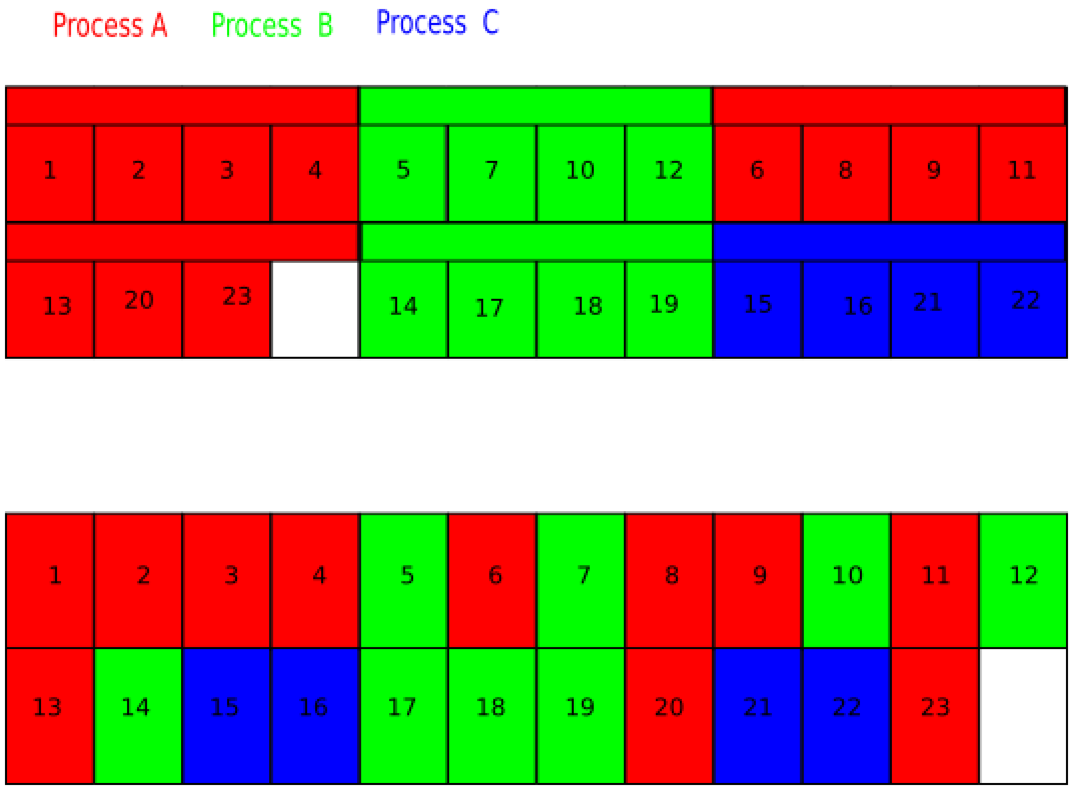
Chunk Allocation



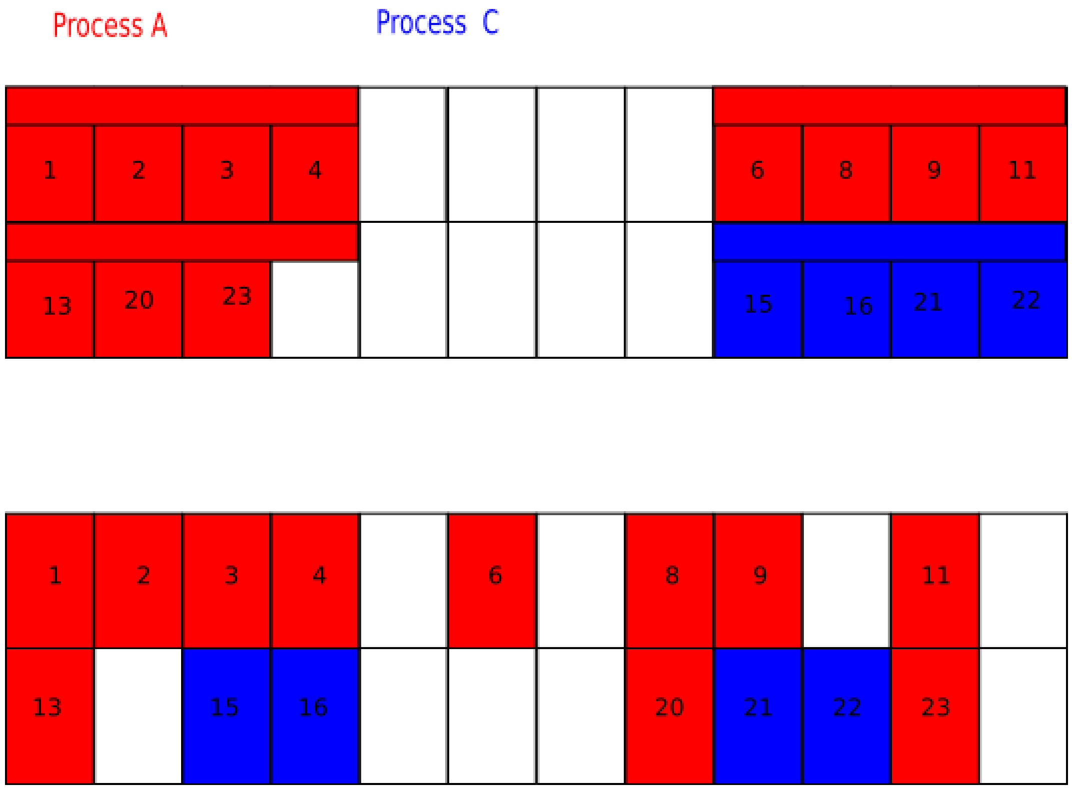
Chunk Allocation



Chunk Allocation



Chunk Allocation

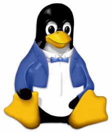


Pros/Cons

- Advantages
  - Naturally reduces fragmentation
  - Naturally scalable
  - Cache friendlier data structures
  - Operate on groups of pages
  - No need to continuously initialise/free page descriptor objects
  - Increase potential to promote to large pages
- Disadvantages
  - don't always allocate “hot page”
    - maybe doesn't matter on PPC (dcbz)

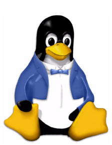
What is K42

- Open source research kernel (64 bit, cache coherent systems)
- Focus on performance, scalability, customizability, maintainability
- Supports Linux API/ABI
- Uses Linux device drivers, filesystems, ...
- Userspace servers (NFS, socket, pipe server)
  - Application level libraries
- Pageable kernel data, Userspace thread scheduling, ...

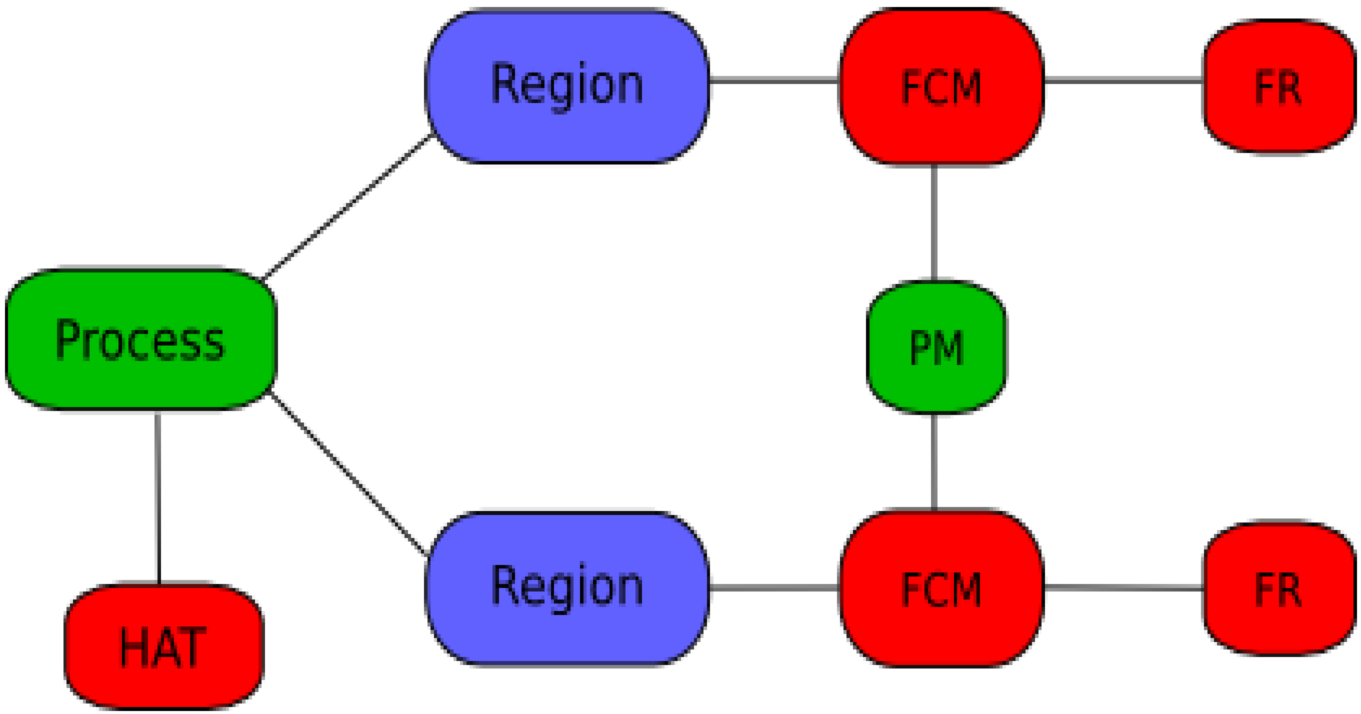


Why K42?

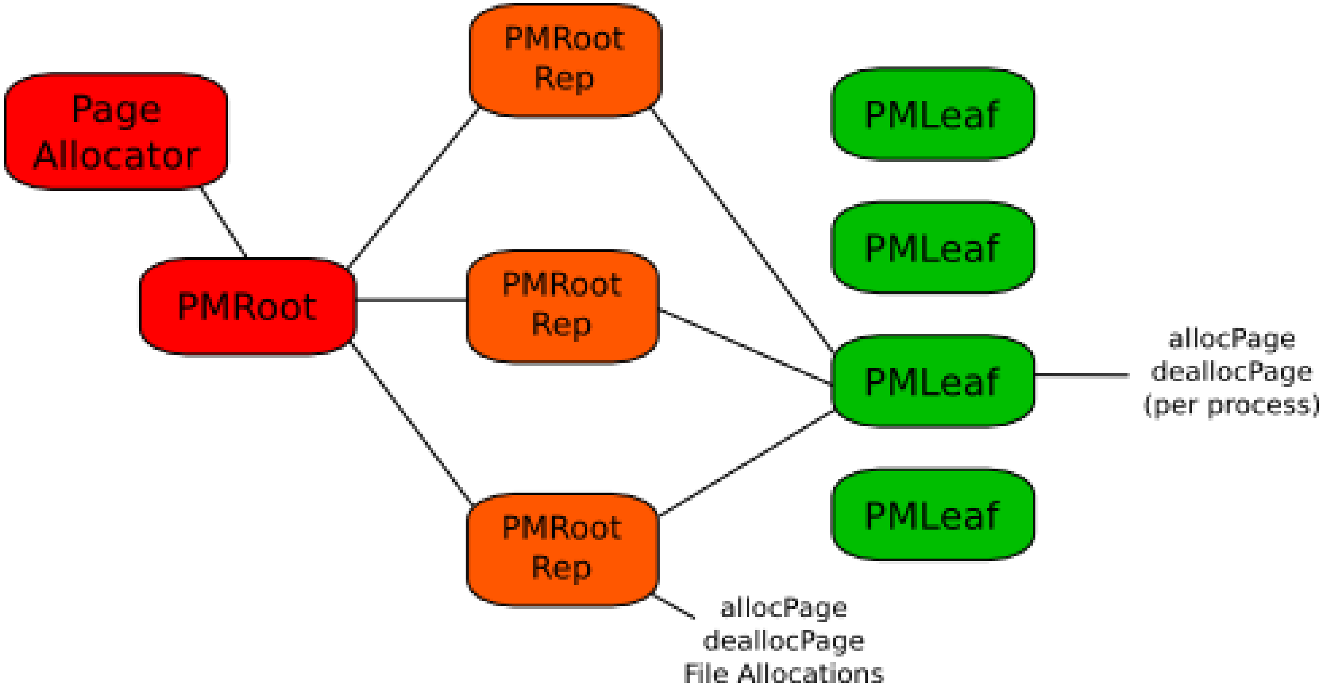
- Designed for experimentation/prototyping
  - OO design – easy to add alternative implementations
  - Supports Linux API/ABI (64-bit PowerPC)
  - Allocation occurs through per process object
  - Has infrastructure for experimentation



K42 Memory Management System



K42 Memory allocation

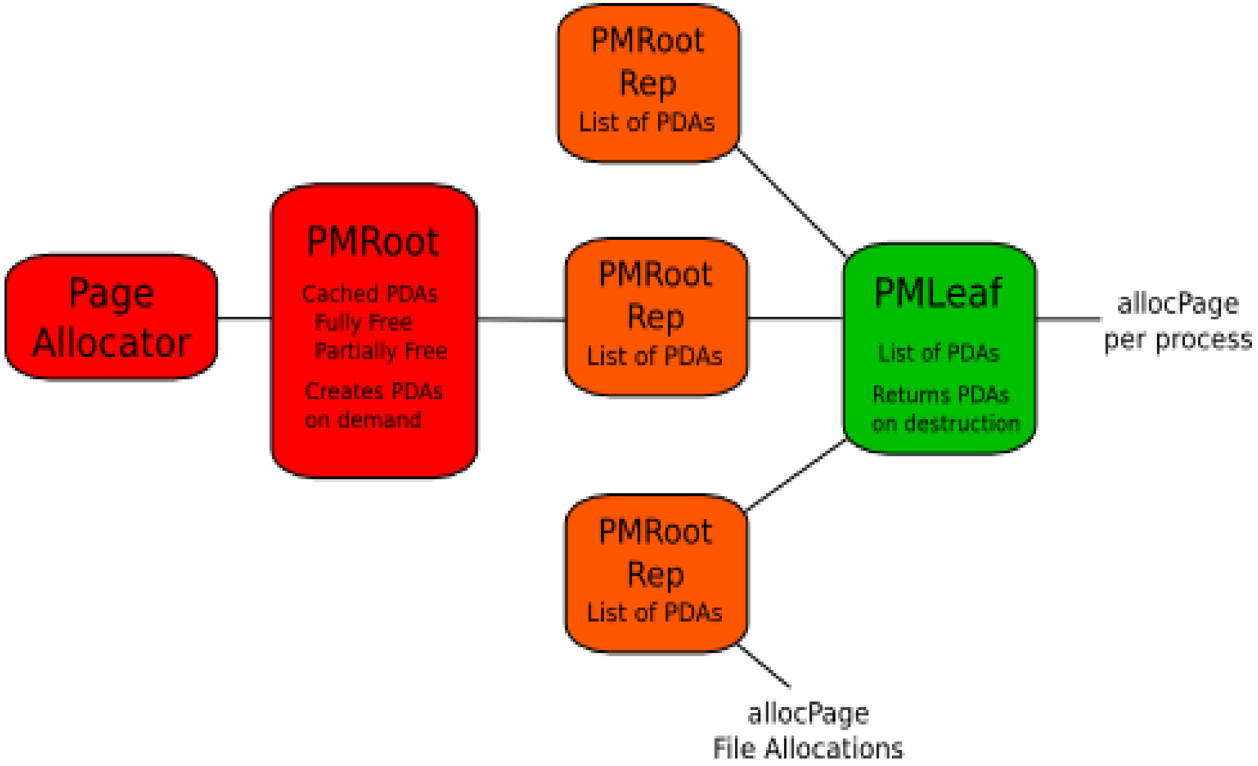


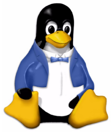
- allocations
  - page allocator, root rep, pmleaf
  - caching of pages in PMRoot and PMRoot reps
  - freeing through same places
    - doesn't have to be from original object

Implementation in K42

- Page Descriptor Array (PDA) Header
  - Page size, Array Size, Chunk start address
  - Bitmap of free pages
- PDA header and chunk allocated separately
- allocPage – returns address **and** pointer to PDA
  - Page descriptors have a field added to store PDA pointer
  - deallocPage removed – frees done directly to PDA
- Under memory pressure fully freed PDAs are freed back to page allocator
  - can re-use partially populated PDAs

K42 PDA memory allocation





Experimental results

- Very preliminary
  - Implementation still under development
  - Still debugging/optimising
- Fixed (per boot) size chunks

Performance results

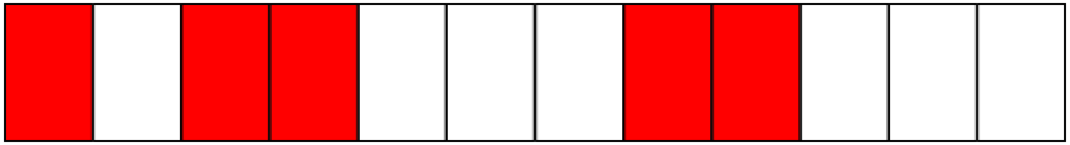
- Performance
  - SDET
    - “system” benchmark
    - commonly used with scalability testing
- 0.5% degradation UP
- Large degradation for SMP
  - have not optimised for SMP yet
  - some obvious places to fix

Fragmentation definition

- Definition of fragmentation
  - Measure amount of free memory
  - For a given page size, calculate number of pages that you should be able to allocate
  - For a given page size, calculate number of pages you can allocate

$$Fragmentation_{PageSize} = \left(1 - \frac{Actual \;\; allocate}{Theoretical \;\; allocate}\right) * 100$$

Example



- Each block is 4kb
- Free memory: 28kb
- For 8kb pages
  - theoretical – 3 x 8kb pages
  - actual – 2 x 8kb pages
  - 33% fragmentation

Fragmentation results

- Test load
  - Long lived processes
    - small allocations/deallocations
  - Short lived processes (forked from long lived ones)
- Simple simulation of web server
- Modified kernel to dump bitmap of all pages in memory marking free/used state

Fragmentation results

- Table of fragmentation vs page size
- Reduced fragmentation for page size <= chunk size
- Increased fragmentation for page size > chunk size

Page Size (kb)	Fragmentation % Normal	Fragmentation PDA (256kb)
4	0	0
8	1.3	0.4
16	3.5	1
32	7.3	1.3
64	11.8	1.6
128	16.1	2.1
256	17	2.9
512	17.3	64

Future work #1

- Further debugging/optimisations
  - SMP
- Move more bitmaps into page descriptor array
  - eg dirty pages
- Handling low memory conditions
  - Splitting of page descriptor arrays
  - Swapping out entire chunks

Future work #2

- Variable sized chunks
  - tailor size of chunk to process (CPO)
- PDAs passed through to FCMs
- File allocations grouped (PMLeaf equivalent)
- Reverse mapping in PDAs to point to FCMs
  - paging/defragmentation
- Move technology into Linux

Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- PowerPC is a registered trademarks of International Business Machines Corporation in the United States, other countries, or both.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product and service names may be trademarks or service marks of others.