CSCE 313-200
Introduction to Computer Systems
Spring 2024

Memory

Dmitri Loguinov
Texas A&M University

April 12, 2024

Quiz 5

- Write proper synchronization for a train tunnel

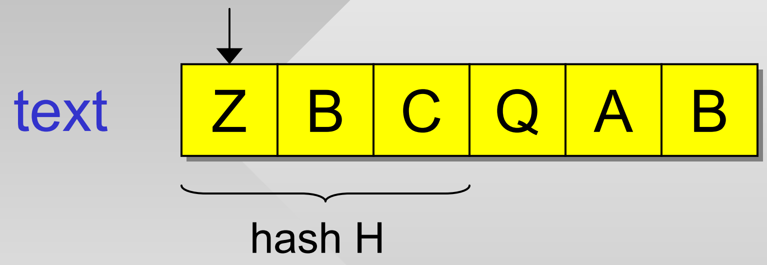
```
                        Train
TryEnteringTunnel (int dir) {
    mutex[dir].Lock();
    if (trains[dir]++ == 0)
          occupied.Wait();
    mutex[dir].Unlock();

    semaMaxN.Wait();
    PassThruTunnel(x, dir);
    semaMaxN.Release();

    mutex[dir].Lock();
    if (--trains[dir] == 0)
          occupied.Release();
    mutex[dir].Unlock();
}
```

Hw3

- Why was homework #3 so inefficient?



keyword list
apple
banana
mango
zebra
…

looking at 'z', no need to attempt a match to apple, banana, mango

- Idea: do not compare current byte to all strings, only to those that can potentially be a match
- Rabin-Karp (RK), 1987
  - Assume M is the smallest keyword length
  - Compute a hash H of the next M chars from current location
  - Hit a hash table, compare with words that tie for that hash
  - Speed is only based on the length of collision chains

Hw3

example with M = 3, B = 10

- After hash table lookup, slide by one byte forward, recompute the hash of the next M chars

- Notice that M-1 chars are the same in both hashes
  - Main twist of the algorithm is to use a rolling hash, which obtains H_{i+1} from H_i in O(1) time
- Treating hashes as base-B integers, we have
  - $H_0 = str[0] * B^{M-1} + str[1] * B^{M-2} + \ldots + str[M-1]$
  - $H_{i+1} = (H_i * B + str[i+M]) \% B^M$

Hw3

- Larger M means fewer collisions and faster operation
- With M = 3 and 216K strings, RK runs at 20MB/s
  - 2000 times faster than the naïve method
- Indexing a file with unknown keywords is slightly different, but the idea is similar to RK
  - Homework #4 explores this in more detail
- Main goal is to design code that processes all 4.5B words in large Wikipedia in ~35 sec (135M wps)
  - 3.7M times faster than the method in homework #3
- Homework #4 has 3 checkpoints
  - The first two should be done early
  - Checkpoint #3 is more complex, uses virtual memory

Chapter 7: Roadmap

- 7.1 Requirements
- 7.2 Partitioning
- 7.3 Paging
- 7.4 Segmentation
- 7.5 Security

Part III

Chapter 7: Memory
Chapter 8: Virtual RAM

Requirements

Main memory services of the OS:

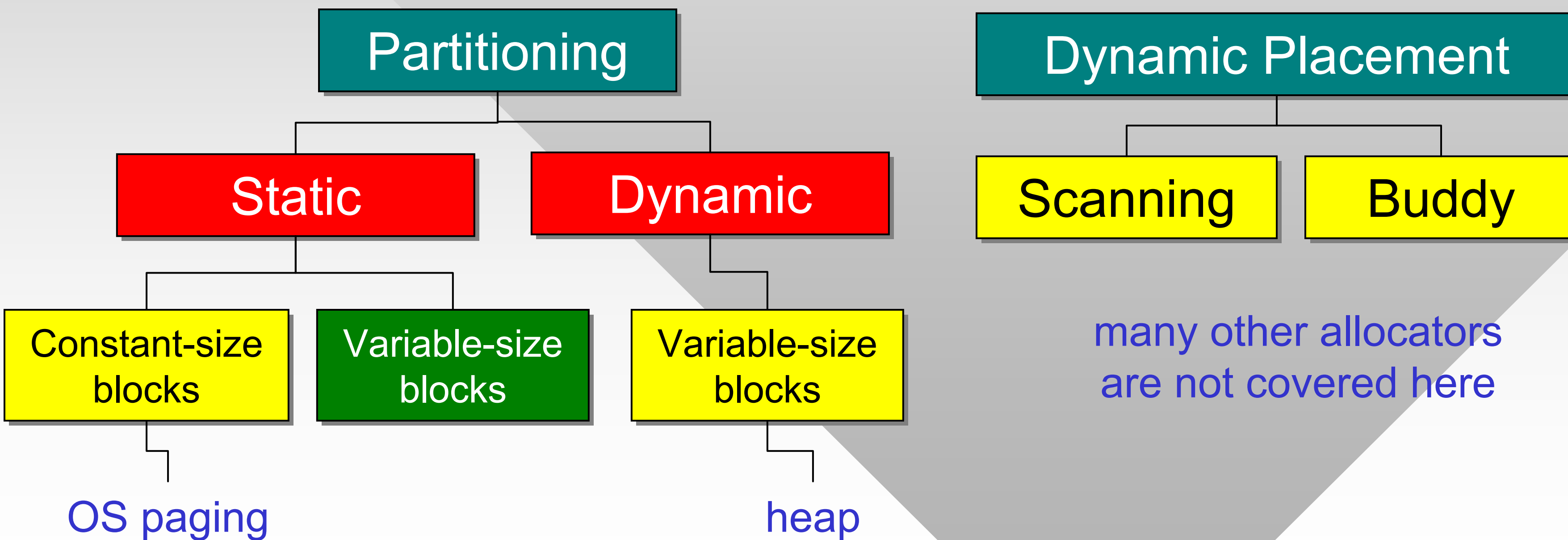
- 1) Dynamic allocation/deletion
- 2) Process & data relocation
  - Transparent fragmentation of process data/code within RAM and swapping to disk as needed
- 3) Protection
  - No unauthorized access to space of other processes
- 4) Sharing
  - Ability to map portions of RAM between different processes

} Memory manager, address virtualization, hardware support

Chapter 7: Roadmap

Memory Management

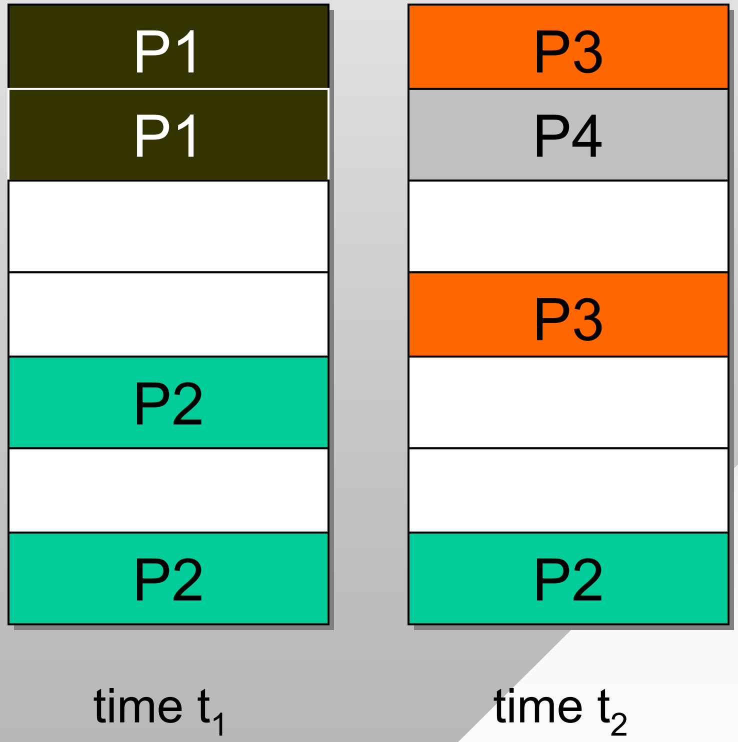
- Memory allocation is a complex problem
  - We examine only the most basic approaches
- Partitioning: type of RAM segmentation into blocks
- Placement: actual block allocation algorithms



Note: memory heaps have nothing to do with priority queues

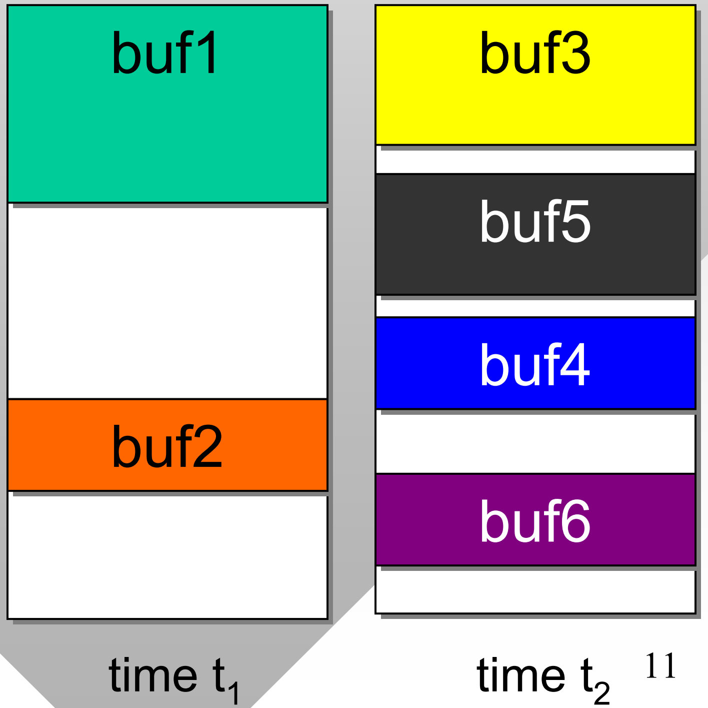
OS Partitioning

- Static partitioning defines block boundaries a-priori
  - Process may hold any number of blocks, which may appear to it as contiguous space
  - Mapping done in hardware
- Suffers from internal fragmentation
- Blocks may be of constant or variable size
  - For simplicity, most kernels have constant-size blocks called pages
- Each page must be a power of 2 (usually 4 KB)



Heap Partitioning

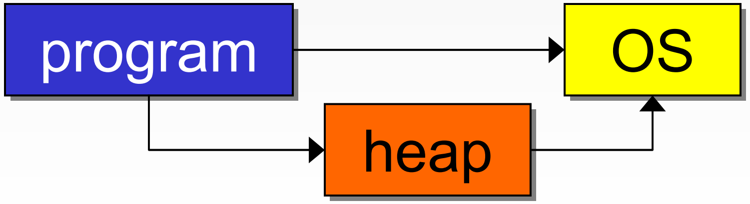
- Tweaking virtual-page tables is slow and a privileged operation; allocation rounded to nearest page size
- Idea: add memory management to user space that can satisfy small buffer request with less overhead
- Dynamic partitioning (heap) grabs pages from the OS, then splits them into smaller chunks in user space
  - Much faster, but leads to external fragmentation
- More difficult to manage due to variable-size blocks



Heap Allocation

```
void f (void) {
    int a;          // on the stack
    // ptr on the stack, buffer on the heap
    char *buf = new char [100];
    // ptr on the stack, buffer from the kernel
    char *OSbuf = VirtualAlloc (...);
}
```

- Memory is typically allocated from:
  - Stack (local variables)
  - Heap (new/malloc)
  - OS (VirtualAlloc)
- We are now concerned with heap
  - OS issues covered in later lectures



- Scanning
  - Linearly search through RAM (or list of blocks) to find empty blocks to allocate
- Search types:
  - First fit: scans from start
  - Best fit: finds the smallest free block that satisfies the request
  - Next fit: searches from the last allocation forward
- E.g., Unix SLOB allocator for simple (embedded) devices