Red Terms you should know

Review (by subject)
Dynamic Memory Allocation

(what you implemented in Program 5)

Obtaining and managing memory space (heap) while a program is running.

Provided:

```c
void *malloc (size_t size);  // in bytes
void free (void *ptr);
```

Notes:

→ heap space is not unlimited.

→ this interface permits variable-sized allocations.

→ for C programs, the programmer is expected to do memory management (free heap-allocated space when done using it).
Implementation (of malloc() + free()):

We want to maintain a list of free/allocated heap space. But, implementing this with **nodes in a linked list** would require dynamically-allocated space.

Solution: use the heap space, and embed the linked list nodes into this heap space.
Many algorithms exist for the allocation. They vary in
- efficiency of malloc + free
- amount of space used for the list info.

4 algorithms to search for and choose a free block (that we discussed):

best fit
closest fit
first fit
next fit

Each has good/bad aspects.

When we free a block, we need to coalesce adjacent free blocks, so that we can re-use memory for further allocation and still do an efficient search.
Caches

Memory accesses take too much time. So, we specialize the design of the memory system to make lots of accesses faster.

Since accesses tend to exhibit
1. temporal locality
2. spatial locality

we implement a cache, a smaller, faster memory to hold copies of what is already in main memory.

on chip, for speed

logical grouping of the memory system
Accesses first go to the cache. A look up determines hit or miss.
On a miss, access goes to main memory, and the required block of data is placed into the cache.
direct mapped cache

uses address

| tag | set # or index # | byte within block |

valid tag

data block

look up uses the address

index # determines the block frame

if (frame contains valid data AND tag in frame matches tag in address)

hit — use LSBs of addr to extract desired byte within block

else

miss — bring block in from memory put in correct frame mark valid & set correct tag
We can attempt to quantify cache performance with statistics.

hit ratio
miss ratio
AMAT
To further increase cache performance, classify the types of misses:

1. compulsory
2. capacity
3. conflict

By increasing the number of sets per line, we often decrease conflict misses.

4-way set associative cache

```
<table>
<thead>
<tr>
<th>address</th>
<th>tag</th>
<th>index# or set#</th>
<th>byte within block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vtag</td>
<td>data</td>
<td>Vtag</td>
<td>data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Caching works so well that we often have 2 separate caches:

- **I-cache** for instructions only
- **D-cache** for everything else

They can be accessed in parallel.

AND, multiple levels of caches:

```
P <-> L1 <-> L2 <-> L3 <-> main memory
```
Virtual Memory

We looked at 3 implementations:

1. base and bounds
2. segmentation
3. paging
Base and Bounds

The entire, contiguous address space of a program is swapped into physical memory.

```
virtual addr

+---------------------+-------------------+
| BASE               | physical addr     |
+---------------------+-------------------+

physical memory

| code  | data  |
+-------+-------+
|       |       |

heap↓

stack↑

-----

yes fault/
no OK addr
```
Segmentation
Base and bounds applied on a per segment basis.

Segment table has one entry per segment. Each entry has a base & bounds.
virtual address

| segment number | offset |

index

Segment table

| base | bounds |

Notes

- Segment table is in memory

- Use of bounds not shown in this diagram
Paging

Swapping is done for fixed-size pages.

A virtual page may be placed into any physical page frame.
virtual address

virtual page number 
offset

PTBR

page table

index

PFN

Notes

- page table is in physical memory
- PFN is physical frame number
- PTBR - physical address of start of page table
Linking

Forms a single executable from pieces of object code that result from separate assembly.

The code is incomplete due to absolute and relative addresses that are unknown at assembly time because of unresolved symbols. Fixes and completes the code.
Networks & Internet Programming

We just finished this subject in lecture. It should be fresh enough to not need review.