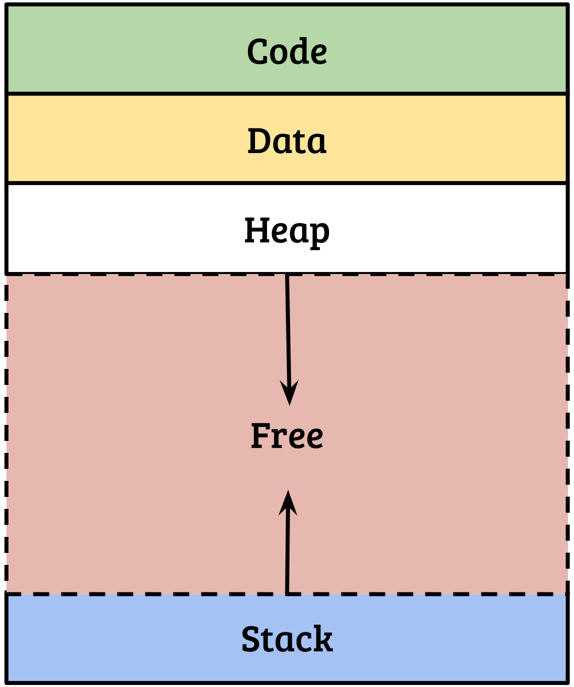
CS330: Operating Systems

Virtual memory: Memory API

Recap: Process address space

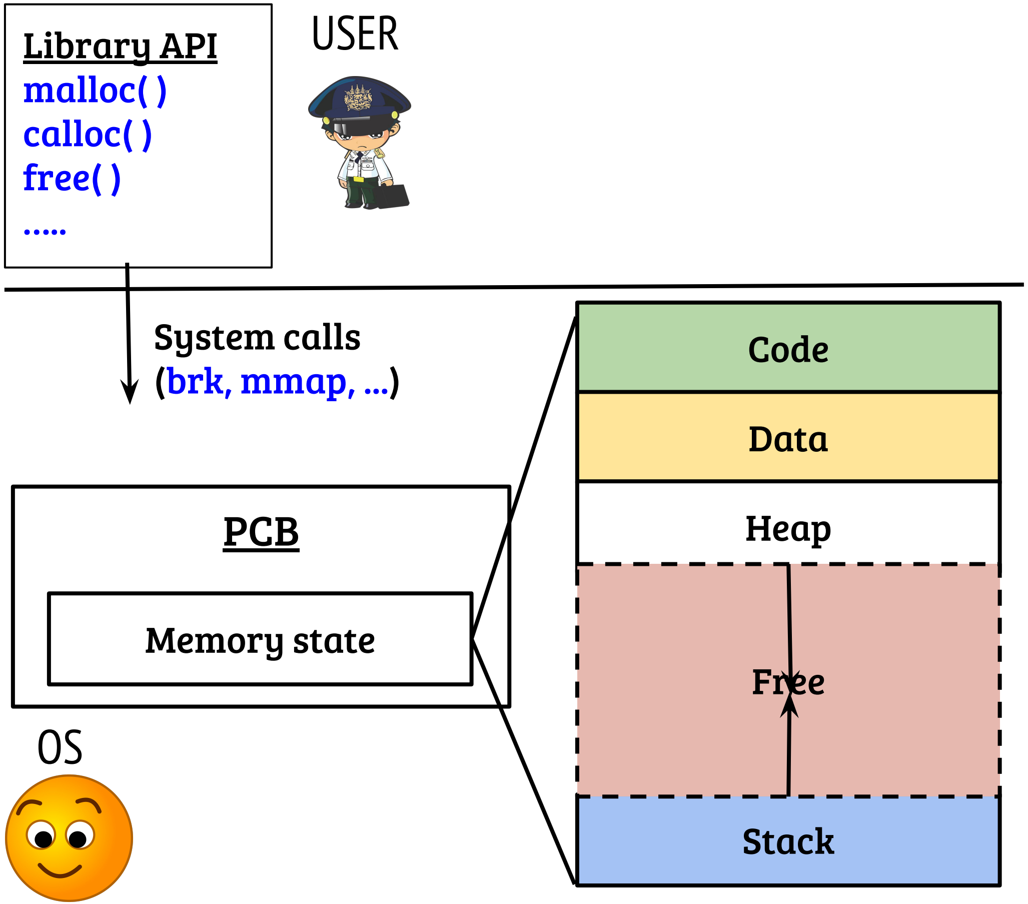


- Address space presents the same view of memory to *all processes*
  - Address space is virtual
  - OS enables this virtual view

Recap: Process address space

- If all processes have same address space, how they map to actual memory?
- Architecture support used by OS techniques to perform memory virtualization i.e., translate virtual address to physical address (will revisit)
- What are the responsibilities of the OS during program load?
  - How CPU register state is changed?
- Creating address space, loading binary, updating the PCB register state
- What is the role of OS in dynamic memory allocation?

User API for memory management



- Generally, user programs use library routines to allocate/deallocate memory
- OS provides some address space manipulation system calls (week's agenda)

User API for memory management

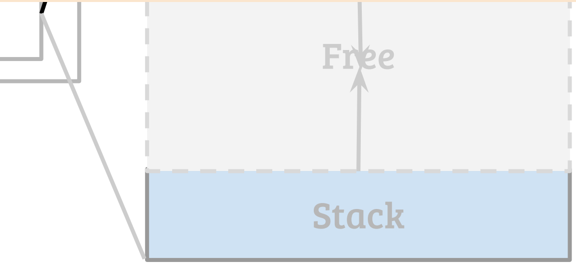
Library API
malloc()
calloc()

USER

- Generally, user programs

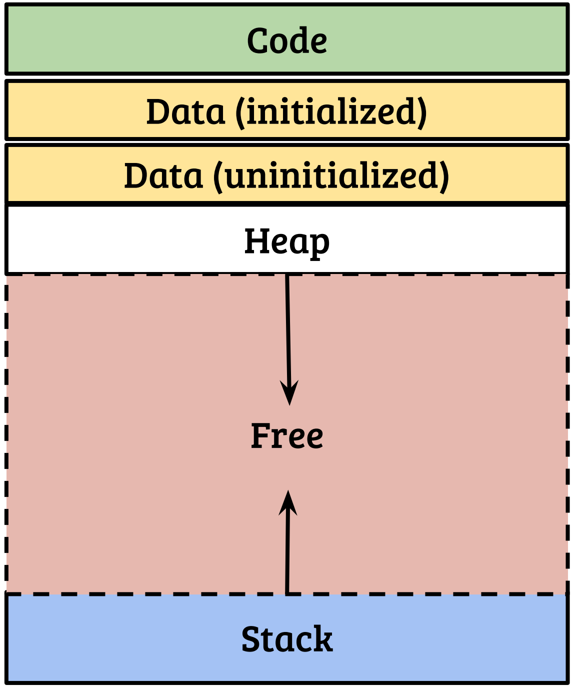
- Can the size of segments change at runtime? If yes, which ones and how?
- How can we know about the segment layout at program load and runtime?
- How to allocate memory chunks with different permissions?
- What is the structure of PCB memory state?

calls (today's agenda)



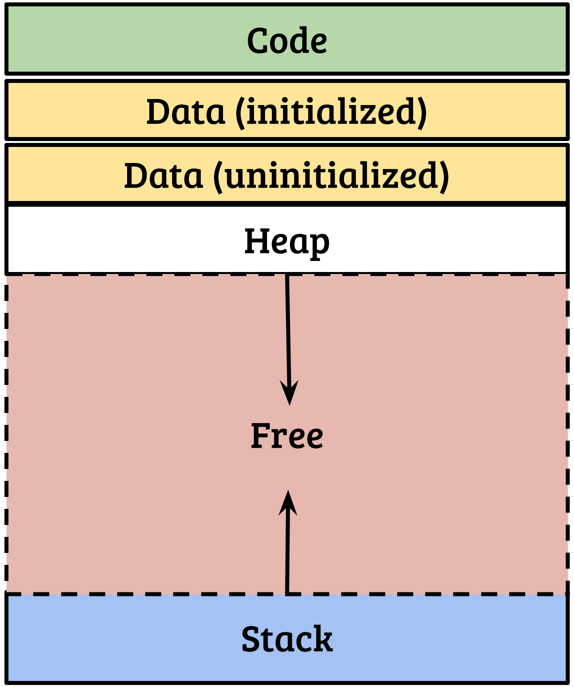
OS

Dynamically sizing the segments (UNIX)



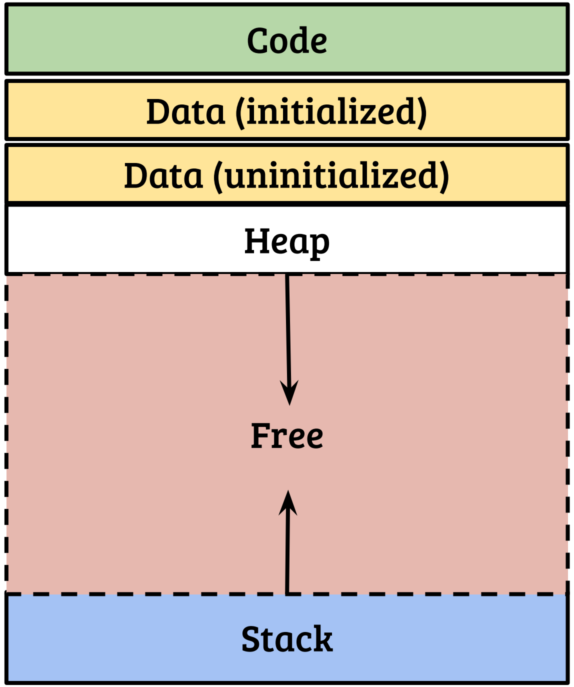
- Code segment size and initialized data segment size is fixed (at exe load)

Dynamically sizing the segments (UNIX)



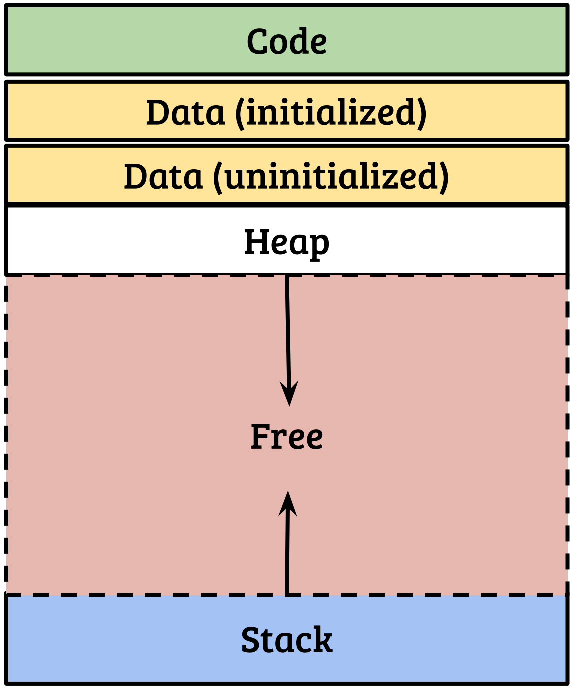
- Code segment size and initialized data segment size is fixed (at exe load)
- End of uninitialized data segment (a.k.a. BSS) can be adjusted dynamically

Dynamically sizing the segments (UNIX)



- Code segment size and initialized data segment size is fixed (at exe load)
- End of uninitialized data segment (a.k.a. BSS) can be adjusted dynamically
- Heap allocation can be discontinuous, special system calls like mmap( ) provide the facility

Dynamically sizing the segments (UNIX)



- Code segment size and initialized data segment size is fixed (at exe load)
- End of uninitialized data segment (a.k.a. BSS) can be adjusted dynamically
- Heap allocation can be discontinuous, special system calls like mmap( ) provide the facility
- Stack grows automatically based on the run-time requirements, no explicit system calls

Sliding the BSS (brk, sbrk)

int brk(void *address);

- If possible, set the end of uninitialized data segment at *address*
- Can be used by C library to allocate/free memory dynamically

void * sbrk (long size);

- Increments the program's data space by size bytes and returns the old value of the end of bss
- sbrk(0) returns the current location of BSS

Finding the segments

- etext, edata and end variables mark the end of text segment, initialized data segment and the BSS, respectively (At program load)
- sbrk(0) can be used to find the end of the data segment
- Printing the address of functions and variables
- Linux provides the information in /proc/pid/maps

User API for memory management

Library API

USER

- Can the size of segments change at runtime? If yes, which ones and how?
- Heap and data segments can be adjusted using brk and sbrk
- How can we know about the segment layout at program load and runtime?
- Using predefined variables, sbrk, proc file system (Linux)
- How to allocate memory chunks with different permissions?
- What is the structure of PCB memory state?

Stack

Discontiguous allocation (mmap)

- mmap( ) is a powerful and multipurpose system call to perform dynamic and discontiguous allocation (explicit OS support)
- Allows to allocate address space
  - with different protections (READ/WRITE/EXECUTE)
  - at a particular address provided by the user
- Example: Allocate 4096 bytes with READ+WRITE permission

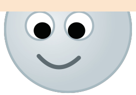
```
ptr = mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_ANONYMOUS
|MAP_PRIVATE, -1, 0);  // See the man page for details
```

User API for memory management

Library API

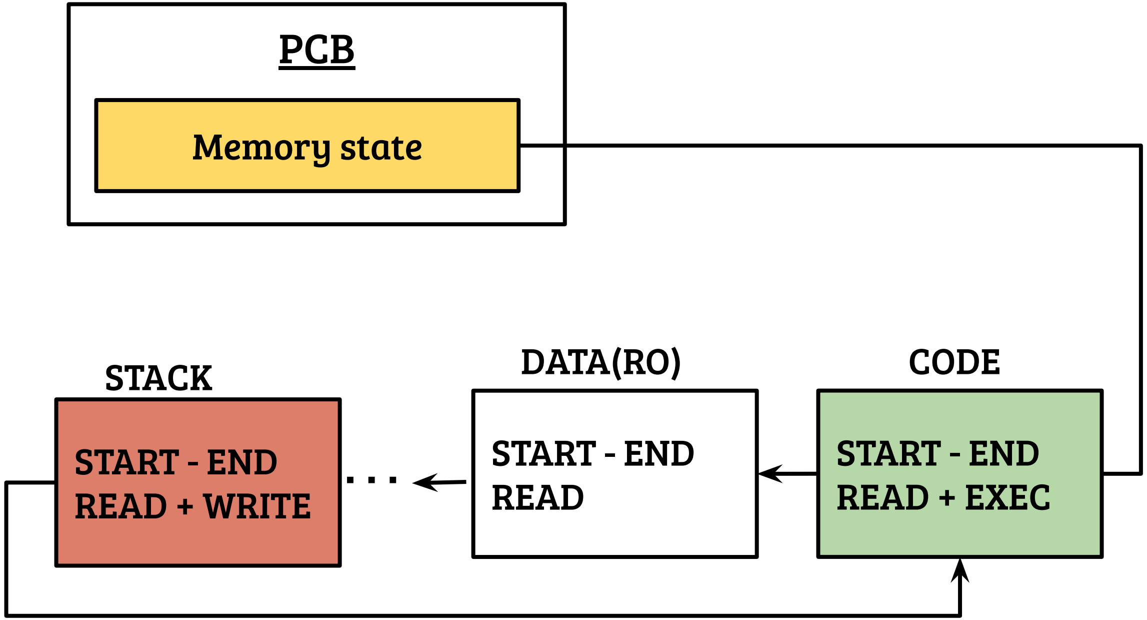
USER

- Can the size of segments change at runtime? If yes, which ones and how?
- Heap and data segments can be adjusted using brk and sbrk
- How can we know about the segment layout at program load and runtime?
- Using predefined variables, sbrk, proc file system (Linux)
- How to allocate memory chunks with different permissions?
- mmap( ) supports discontinuous allocation with different permissions
- What is the structure of PCB memory state?



Stack

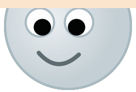
Memory state of PCB (example)



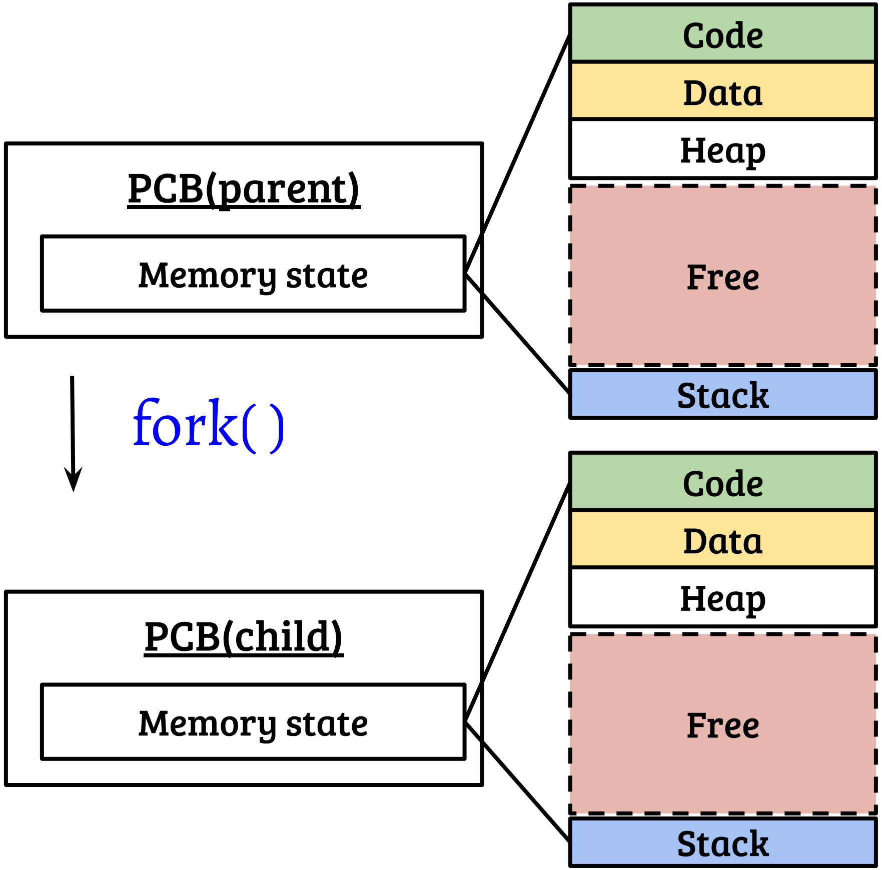
- Maintained as a sorted circular list accessible from PCB
- START and END never overlap between two segment areas
- Can merge/extend areas if permissions match

User API for memory management

- Can the size of segments change at runtime? If yes, which ones and how?
- Heap and data segments can be adjusted using brk and sbrk
- How can we know about the segment layout at program load and runtime?
- Using predefined variables, sbrk, proc file system (Linux)
- How to allocate memory chunks with different permissions?
- mmap( ) supports discontinuous allocation with different permissions
- What is the structure of PCB memory state?
- A sorted data structure of allocated areas can be used

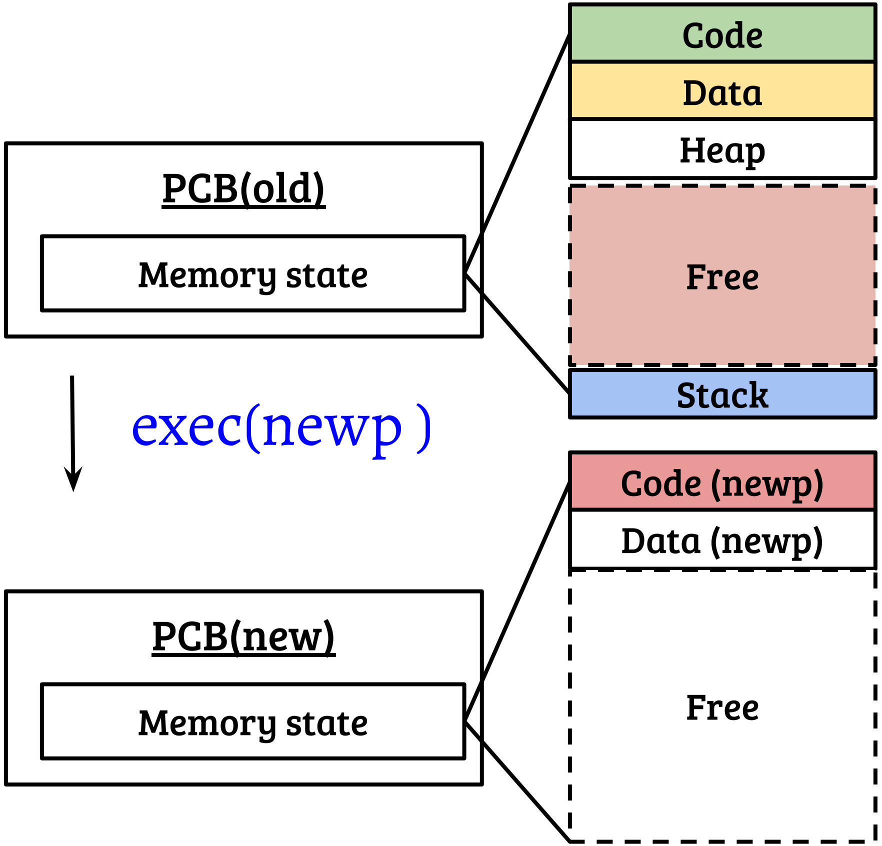


Inheriting address space through fork( )



- Child inherits the memory state of the parent
    - The memory state data structures are copied into the child PCB
- Any change through mmap( ) or brk( ) is per-process

Overriding address space through exec( )



- The address space is reinitialized using the new executable
- Changes to newly created address space depends on the logic of new process