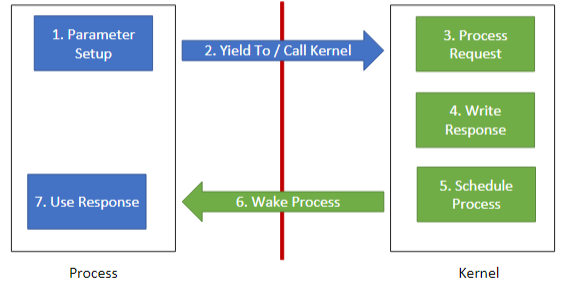
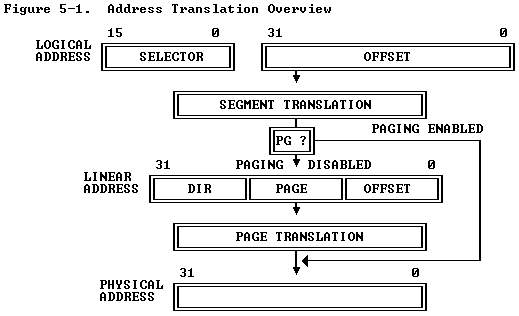
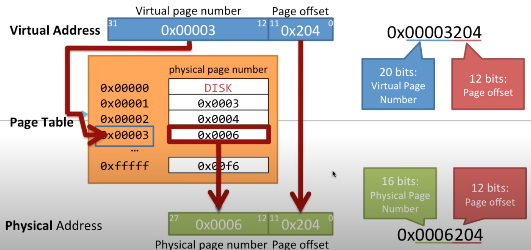
* 1. **List the six common services provided by an operating system. Briefly describe what each does.**
     + *Security* defines access control (system login, file/resource access) and System Integrity (segfault, general protection fault.
     + *Isolation* keeps each process separated. The kernel's memory is not User Memory. User environments are separate.
     + *Communication* talks process to kernel, hardware, abstraction layer, and process.
     + *Resource Management* distributes resources to RAM, CPU Cycles, Bandwidth, Disk space. Handles requests for more. Quotas. Reclaims Resources.
     + *Multiplexing & Scheduling* loads programs, determines which program runs (process yields, pre-emptive tasking), many threads, one processor
     + *Hardware Abstraction* device drivers, common interface, Hardware Abstraction Layer (HAL).
  2. **What is the difference between kernel space and user space, and why is this separation important?**
     + Kernel space has full permission of processes and hardware, where user space is managed by the kernel and must ask to run processes.
  3. **What is the primary unit of isolation in an operating system?**
     + The address space/processes
  4. **What are the three common types of operating system architectures? Briefly describe each.**
     + *Monolithic* all operating system is contained in the kernel
     + *Layered* the kernel contains different layers that can be modified individually
     + *Microkernel* keeps the kernel small by executing some processes outside the kernel (to daemon processes) with low system access levels
  5. **What is a system call?**
     + The primary entry point into the kernel from a user process.
  6. **Briefly describe the series of events that occurs during a system call.**



* 1. **What is hardware abstraction, and why is this an important feature of operating systems?**
     + A set of routines that provide programs with access to hardware resources through programming interfaces. It provides a device driver information, allowing a program to communicate with hardware.
  2. **What are some of the typical services provided by the kernel?**
     + *Processes* Loading/Unloading, Management, Scheduling
     + *Input/Output* Inter-process Communication (IPC), Hardware, Network Transmission
     + *Timers, Memory Management, Filesystems*
  3. **How does the PC bootstrapping process work? Briefly describe the series of events that occur during this process.**
     + CPU begins at hardwired address that usually holds a ROM segment. The ROM loads the loader which loads the kernel.
     + CPU beings at 0x000ffff0, BIOS initializes devices, BIOS loads 512 byte boot sector at address 0x7c00 and then jumps to 0x7c00.
  4. **What is the underlying metaphor for the UNIX kernel?**
     + Everything is a file (add: really, everything is a buffer that the kernel abstracts as a data stream).
  5. **What is a file descriptor?**
     + It's an integer that indicates which buffer we're using. It starts at 0 and will use the lowest descriptor.
  6. **What are the standard file descriptors which are open when a UNIX process is created? Briefly describe what each is for.**
     + 0 - STDIN, 1 - STDOUT, 2 - STDERR
  7. **How are processes created in UNIX? What is the sequence of system calls to load a new program?**
* System initialization - kernel loads initial process that will create processes for system services (daemons), Execution of process-creating system-call by a running process - Unix will fork() then exec(), User request to create a new process - User shell environment (GUI, Command line shell), initiation of a batch job.
  1. **How do UNIX pipes work? How can you establish a pipeline between a parent and child process?**
     + Pipes are chains of processes put together by standard streams, the output text of each process (STDOUT) is passed directly as input (STDIN) to the next. When you fork the process, it copies it and returns the PID of the child to the parent and 0 to the child.
  2. **What happens when UNIX opens a file?**
     + int open(char \*path, int flags, mode\_t mode)
     + It opens a file descriptor in the lowest available disk, flags are O\_APPEND, O\_CREAT, O\_RDWR…, mode is the octal encoding of file permissions, returns the file descriptor on success and -1 on failure.
  3. **Briefly describe each of the following UNIX system calls:**
     + **Read** Read bytes from the file buffer at specified fd. Reads AT MOST count bytes into buf. Same return as write. *ssize\_t read(int fd, const void \*buf, size\_t count)*
     + **Write** Writes bytes to the files buffer specified by fd. Writes nbytes beginning at address buff. Returns number of bytes written on success and -1 if not. *ssize\_t write(int fd, const void \*buf, size\_t nbytes)*
     + **Exit** Terminates a process, sets the process exit status to the given code. *void exit (int code)*
     + **Pipe** Separated by '|'. Returns two fd's, both open for reading and writing. A read from fd[0] access fd[1]. *pipe(p).*
     + **Dup** Creates a duplicate fd at first available (basically a pointer). *int dup(int oldfd)*
     + **Close** Closes the given file descriptor, freeing it for use, returns 0 on success and -1 on error. *int close (int fd)*
     + **Fork** Creates copy of current process, returns PID of child to parent, returns 0 to child. *int fork()*
     + **Exec** Replaces current process with the program in the file specified by path. argv is argument list, anvp is the environment list, both vectors are terminated with a null pointer. Returns -1 on fail and not otherwise. *int execve(char \*path, char \* const argv[], char \*const envp[])*
  4. **Be able to write code which uses the above system calls to accomplish some task.**
  5. **What is the philosophy for UNIX system programs?**
     + Minimalist, modular software development.
  6. **What does a shell do?**
     + It gets user input and translates it so the OS can understand it and perform the functions. It has a parser that creates a data structure from user input and command execution that establishes pipelines, creates file redirects, and creates new processes.
  7. **Is a shell typically part of the kernel?**
     + No
  8. **How does UNIX IPC work?**
     + It uses pipe and dup commands. int pipe(int pipefd[2]); int dup(int oldfd);
  9. **Describe the memory hierarchy and briefly discuss what each level is and how it behaves.**
     + Volatile Cache is the fastest, Volatile RAM, Non-volatile storage (HDD)
  10. **What job is performed by the memory manager?**
      + Allocating memory among processes, controlling access to memory, preventing process interference.
  11. **Briefly describe the operation of:**
      + **Absolute Access** Programs contain physical addresses, access to all memory, no hardware abstraction.
      + **Address Spaces** Range of addresses giving the illusion of one program which allows programs to coexist.
      + **Base and Limit Registers** Dynamic relocation, physical address space: [Base, Limit], Address Translation: Physical = Addr + Base. Access Canceled if Physical > Limit
  12. **What are some of the common ways of managing free memory? Briefly describe their function.**
      + Bitmaps allocate fixed blocks with a bit (0-free, 1-allocated)
      + Linked Lists allocate variable width blocks of linked list of holes with Algorithms: first fit, next fit, worst fit, quick fit.
  13. **What is virtual memory, and why is it important?**
      + Virtual memory is the CPUs mapping of physical addresses that can be used by programs for security and more advanced MMU features.
  14. **What is memory paging, and what does this allow us to do?**
      + Paging is a way to retrieve data from secondary storage for use in main memory which lets programs exceed the size of available RAM.
  15. **What is a page fault? Briefly describe the scenarios in which a page fault would occur.**
      + When a program tries to R/W a page not in memory (swap page into memory), Execute Outside of Privilege (General Protection Fault), and Write To a Read Only Page (Illegal Instruction).
  16. **What are the steps in the i386 address translation process?**
      + Logical Address from Instruction, Segment Translation, Page Translation, Physical Address

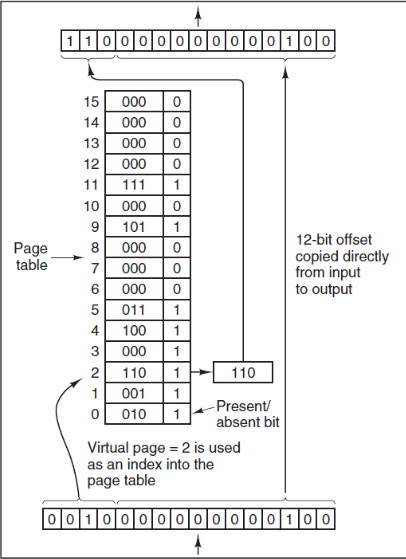
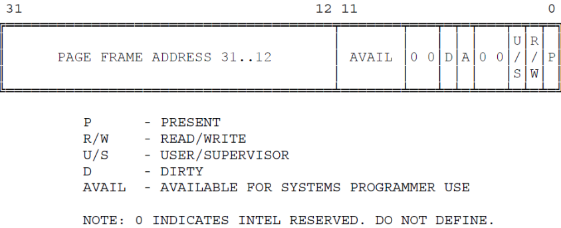


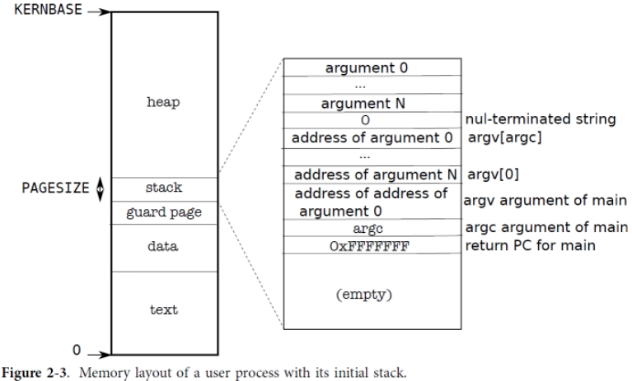
* 1. **Given an i386 page table, segment registers, and an address; translate a virtual address into a physical address.**



* 1. **How big are the physical page frames in the i386?**
     + 4096 Bytes
  2. **Draw a sketch of the i386 page table. What does each level of the table represent?**

Two levels of tables are used to address a page of memory. The higher level is a page directory, which can address up to 1K page tables of the second level. A page table of the second level addresses up to 1K pages for a total of 1M pages.

The page frame address specifies the starting address of a page. Because pages are located on 4K boundaries, the low-order 12 bits are always 0. The present bit indicates whether a page table entry can be used in address translation, where P = 1 indicates it can be used.

* 1. **What register is used to set the current privilege level in the i386?**
     + Data Segment register
  2. **What are the two privilege levels used by xv6 and the SEMOS kernel? What ring numbers do they use, and how do they differ?**
     + Ring 0 and 3. Ring 0 - data segments at all privilege levels are accessible. Ring 3 - Only data segments at privilege level 3 are accessible.
  3. **How do we protect access to pages in the i386?**
     + Privilege is broken into two parts: supervisor (U/S=0) and user (U/S=1). These are assigned based on the CPL: 0, 1, 2 will be supervisor and 3 will be user.
  4. **What are the major parts of the UNIX memory model? Briefly describe each part.**
     + Text (RO, runnable code), Data (initialized data), Heap (space for dynamic allocation), Stack (process calling/local variables).
  5. **What are the major states of a process? Describe the function of each state.**
     + Running (The process is running in the CPU at this instant). Blocked (The process is waiting on something). Ready (The process can run but is currently not in the CPU).
  6. **What is an interrupt?**
     + A response by the processor to an event. It breaks the normal loop, calls the Interrupt Service Routine, and then runs IRET. The processor loop resumes.
  7. **How does privilege translation occur in the i386 processor?**
  8. **What is the difference between an interrupt and an exception in the i386 processor?**
     + Exceptions are triggered by software. Interrupts are generated by hardware.
  9. **What are the major types of exceptions? Describe each and how they differ from each other.**
     + Faults are reported "before" the instruction. Fault Detection: Fault raised during an instruction, CPU reverts to a state that allows the instruction to be repeated, After the Interrupt Service Routine, the instruction is repeated.
     + Traps are reported immediately after the instruction. Trap Detection: Trap is detected at the instruction boundary, CPU updates the EIP, and then enters the ISR, then control is returned to the next instruction.
     + Aborts provide neither instruction location nor a means of resuming. It's a serious hardware error e.g. invalid Page Table.
  10. **How does the i386 processor find the location of the routine which handles an interrupt?**
      + Interrupt Descriptor Table (IDT). The location of the IDT is kept in the IDTR register of the CPU, which can be loaded/stored with LIDT, SIDT instructions.