UNIX Design Principles

- Designed to be a time-sharing system.
- Has a simple standard user interface (shell) that can be replaced.
- File system with multilevel tree-structured directories.
- Files are supported by the kernel as unstructured sequences of bytes.
- Supports multiple processes; a process can easily create new processes.
- High priority given to making system interactive, and providing facilities for program development.

Programmer Interface

Like most computer systems, UNIX consists of two separable parts:

- Kernel: everything below the system-call interface and above the physical hardware.
  - Provides file system, CPU scheduling, memory management, and other OS functions through system calls.
- Systems programs: use the kernel-supported system calls to provide useful functions, such as compilation and file manipulation.

4.3BSD Layer Structure

	(the users)	
	shells and commands compilers and interpreters system libraries	
	system-call interface to the kernel	
signals terminal handling character I/O system terminal drivers	file system swapping block I/O system disk and tape drivers	CPU scheduling page replacement demand paging virtual memory
	kernal interface to the hardware	
terminal controllers terminals	device controllers disks and tapes	memory controllers physical memory

Process Control Blocks

- The most basic data structure associated with processes is the *process structure*.
  - unique process identifier
  - scheduling information (e.g., priority)
  - pointers to other control blocks
- The *virtual address space* of a user process is divided into text (program code), data, and stack segments.
- Every process with sharable text has a pointer from its process structure to a *text structure*.
  - always resident in main memory
  - records how many processes are using the text segment
  - records where the page table for the text segment can be found on disk when it is swapped

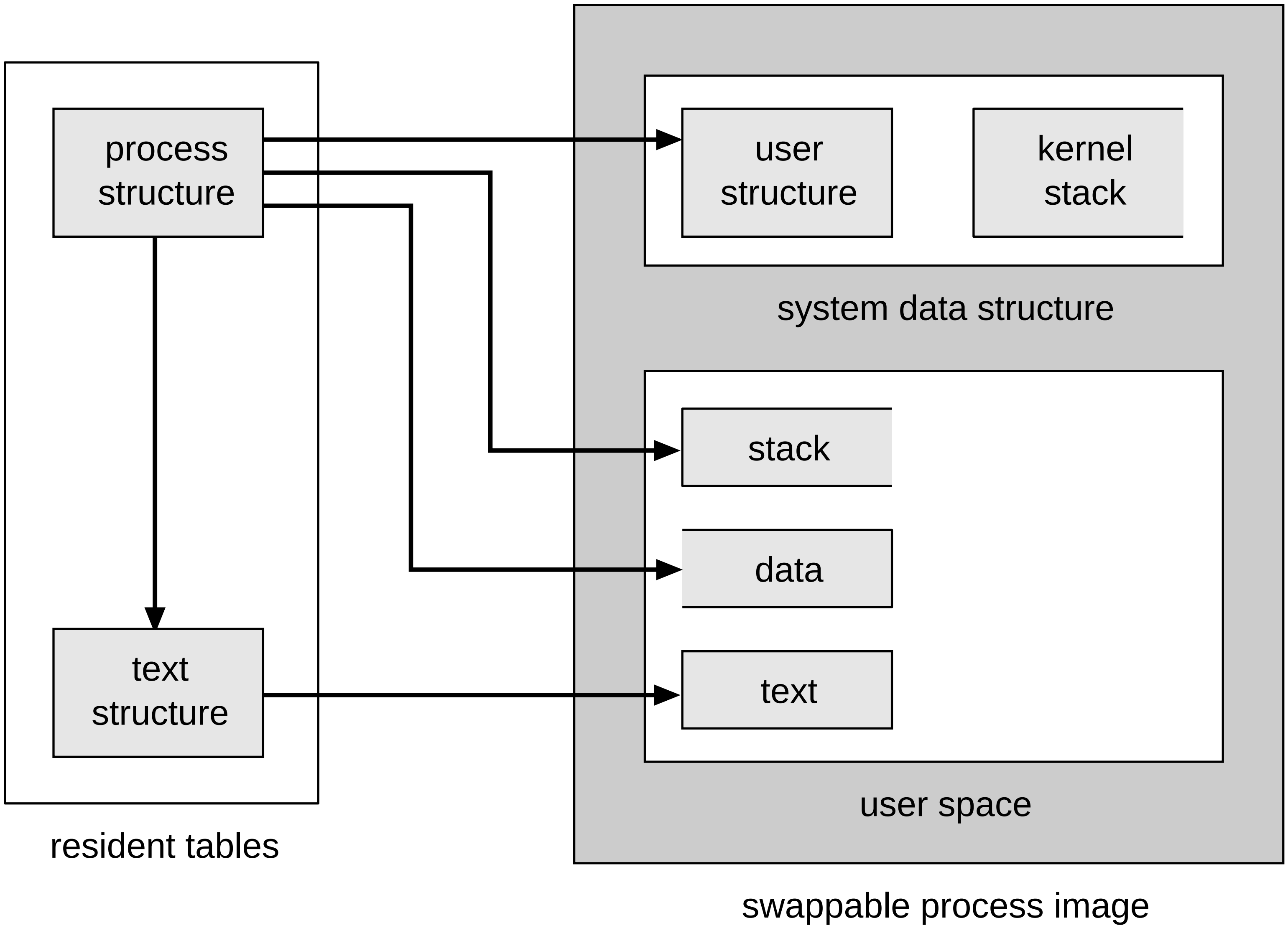
Process Control Blocks (Cont'd)

- The *page tables* record information on the mapping from the process' virtual memory to physical memory.
- Information about the process that is needed only when the process is resident is kept in the *user structure* (or *u structure*).
  - mapped read-only into user virtual address space
  - writable by the kernel
  - maintains the current directory and the table of open files

System Data Segment

- Most ordinary work is done in *user mode*; system calls are performed in *system mode*.
- The system and user phases of a process never execute simultaneously.
- A *kernel stack* (rather than the user stack) is used for a process executing in system mode.
- The kernel stack and the user structure together compose the *system data segment* for the process.

Finding parts of a process using process structure



Allocating a New Process Structure

- **fork** allocates a new process structure for the child process, and copies the user structure.
  - new page table is constructed
  - new main memory is allocated for the data and stack segments of the child process
  - copying the user structure preserves open file descriptors, user and group identifiers, signal handling, etc.

Allocating a New Process Structure (Cont'd)

- **vfork** does *not* copy the data and stack to the new process; the new process simply shares the page table of the old one.
  - new user structure and a new process structure are still created
  - commonly used by a shell to execute a command and to wait for its completion
- A parent process uses **vfork** to produce a child process; the child uses **execve** to change its virtual address space, so there is no need for a copy of the parent.
- Using **vfork** with a large parent process saves CPU time, but can be dangerous since any memory change occurs in both processes until **execve** occurs.
- **execve** creates no new process or user structure; rather, the text and data of the process are replaced.

Memory Management

- The initial memory management schemes were constrained in size by the relatively small memory resources of the PDP machines on which UNIX was developed.
- Pre-3BSD systems use swapping exclusively to handle memory contention among processes: If there is too much contention, processes are swapped out until enough memory is available.
- Allocation of both main memory and swap space is done first-fit.

Memory Management (Cont'd)

- Sharable text segments do not need to be swapped; results in less swap traffic and reduces the amount of main memory required for multiple processes using the same text segment.
- The *scheduler process* (or *swapper*) decides which processes to swap in or out, considering such factors as time idle, time in or out of main memory, size, etc.
- In 4.3BSD, swap space is allocated in pieces that are multiples of a power of 2 and a minimum size, up to a maximum size determined by the size of the swap-space partition on the disk.

Paging

- Berkeley UNIX systems depend primarily on paging for memory-contention management, and depend only secondarily on swapping.
- *Demand paging* – When a process needs a page and the page is not there, a page fault to the kernel occurs, a frame of main memory is allocated, and the proper disk page is read into the frame.
- A *pagedaemon* process uses a modified second-chance page-replacement algorithm to keep enough free frames to support the executing processes.
- If the *scheduler* decides that the paging system is overloaded, processes will be swapped out whole until the overload is relieved.