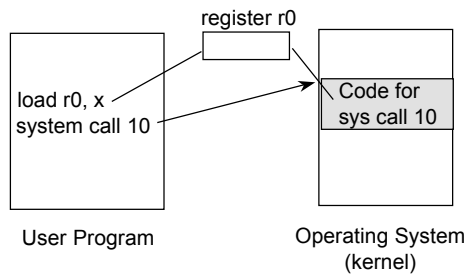


System Calls

- Provide the interface between application programs and the kernel
- Are like procedure calls
 - take parameters
 - calling routine waits for response
- Permit application programs to access protected resources



CMSC 412

1

System Call Mechanism

- Use numbers to indicate what call is made
- Parameters are passed in registers or on the stack
- Why do we use indirection of system call numbers rather than directly calling a kernel subroutine?
 - provides protection since the only routines available are those that are export
 - permits changing the size and location of system call implementations without having to re-link application programs

CMSC 412

2

Types of System Calls

- **File Related**
 - open, create
 - read, write
 - close, delete
 - get or set file attributes
- **Information**
 - get time
 - set system data (OS parameters)
 - get process information (id, time used)
- **Communication**
 - establish a connection
 - send, receive messages
 - terminate a connection
- **Process control**
 - create/terminate a process (including self)

CMSC 412

3

System Structure

- **Simple Structure (or no structure)**
 - any part of the system may use the functionality of the rest of the system
 - MS-DOS (user programs can call low level I/O routines)
- **Layered Structure**
 - layer n can only see the functionality that layer n-1 exports
 - provides good abstraction from the lower level details
 - new hardware can be added if it provides the interface required of a particular layer
 - system call interface is an example of layering
 - can be slow if there are too many layers
- **Hybrid Approach**
 - most real systems fall somewhere in the middle

CMSC 412

4

Policy vs. Mechanism

- Policy - what to do
 - users should not be able to read other users files
- Mechanism- how to accomplish the goal
 - file protection properties are checked on open system call
- Want to be able to change policy without having to change mechanism
 - change default file protection
- Extreme examples of each:
 - micro-kernel OS - all mechanism, no policy
 - MACOS - policy and mechanism are bound together

CMSC 412

5

Processes

- What is a process?
 - a program in execution
 - “An execution stream in the context of a particular state”
 - a piece of code along with all the things the code can affect or be affected by.
 - this is a bit too general. It includes all files and transitively all other processes
 - only one thing happens at a time within a process
- What's not a process?
 - program on a disk - a process is an active object, but a program is just a file

CMSC 412

6

Multi-programming

- Systems that permit more than one process at once
 - virtually all computers today
- Permits more efficient use of resources
 - while one process is waiting another can run
- Provides natural abstraction of different activities
 - windowing system
 - editor
 - mail daemon
- Preemptive vs. non-preemptive multi-programming
 - preemptive means that a process can be forced off the processor by the OS
 - provides processor protection

CMSC 412

7

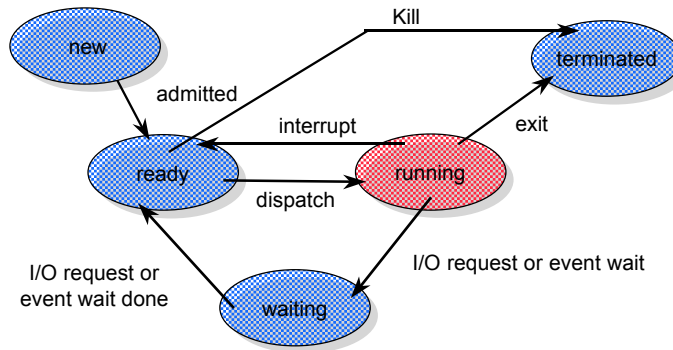
Process State

- Processes switch between different states based on internal and external events
- Each process is in exactly one state at a time
- Typical States of Processes (varies with OS)
 - New: The process is just being created
 - Running: Instructions are being executed
 - only one process per processor may be running
 - Waiting: The process is waiting for an event to occur
 - examples: I/O events, signals
 - Ready: The process is waiting to be assigned to a processor
 - Terminated: The process has finished execution

CMSC 412

8

Process State Transitions



CMSC 412

9

Components of a Process

- **Memory Segments**
 - Program - often called the text segment
 - Data - global variables
 - Stack - contains activation records
- **Processor Registers**
 - program counter - next instruction to execute
 - general purpose CPU registers
 - processor status word
 - results of compare operations
 - floating point registers

CMSC 412

10