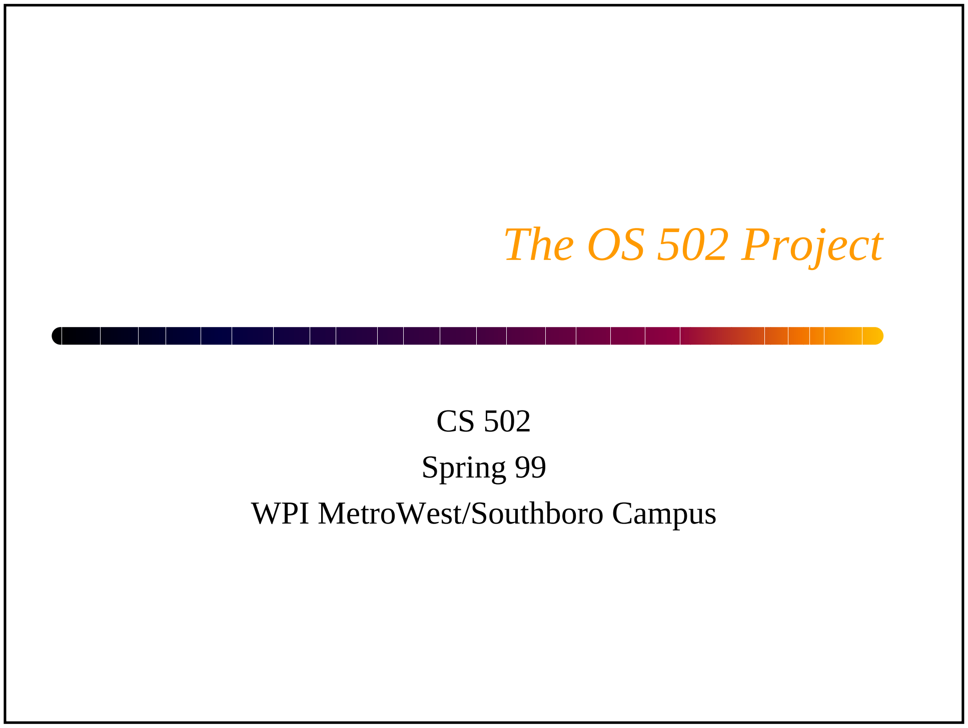
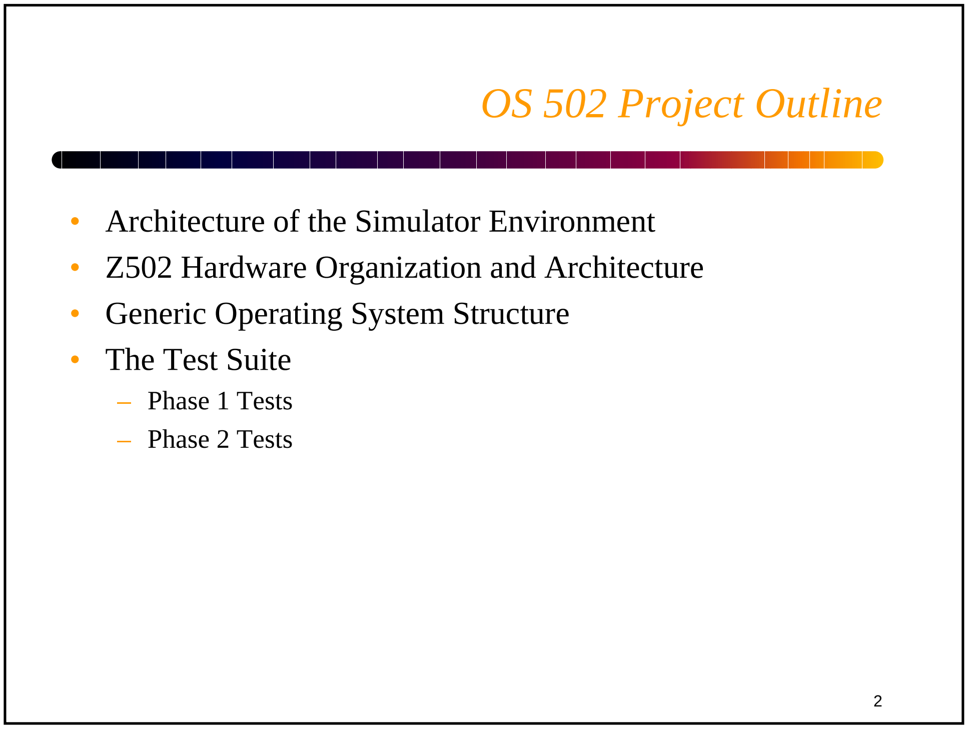


The OS 502 Project

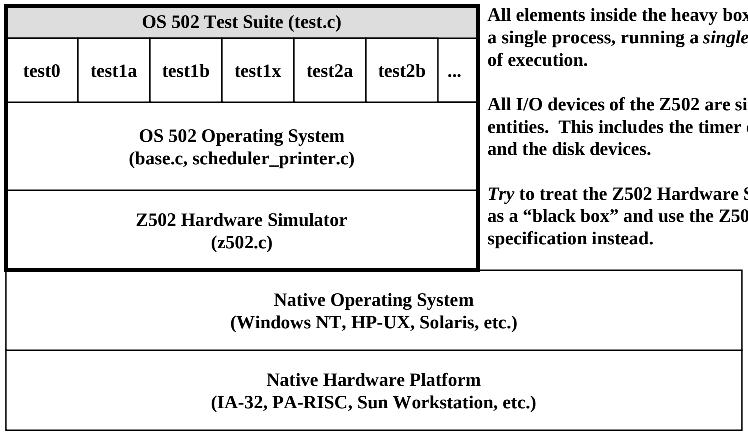
CS 502
Spring 99
WPI MetroWest/Southboro Campus



OS 502 Project Outline

- Architecture of the Simulator Environment
- Z502 Hardware Organization and Architecture
- Generic Operating System Structure
- The Test Suite
  - Phase 1 Tests
  - Phase 2 Tests

Simulator Environment



All elements inside the heavy box are in a single process, running a *single* thread of execution.

All I/O devices of the Z502 are simulated entities. This includes the timer device and the disk devices.

***Try* to treat the Z502 Hardware Simulator as a “black box” and use the Z502 architecture specification instead.**

Z502 Architecture

- Dual-Mode architecture
  - User mode (see A.4)
    - High level language, augmented with
      - Z502 General Purpose Registers
      - Macros for simplifying reentrant programs
      - Systems Calls, provided as macros (do ***not*** rewrite!)
    - Z502 “Programs” are written as C functions taking a void parameter and having a void return.
    - Example Program:

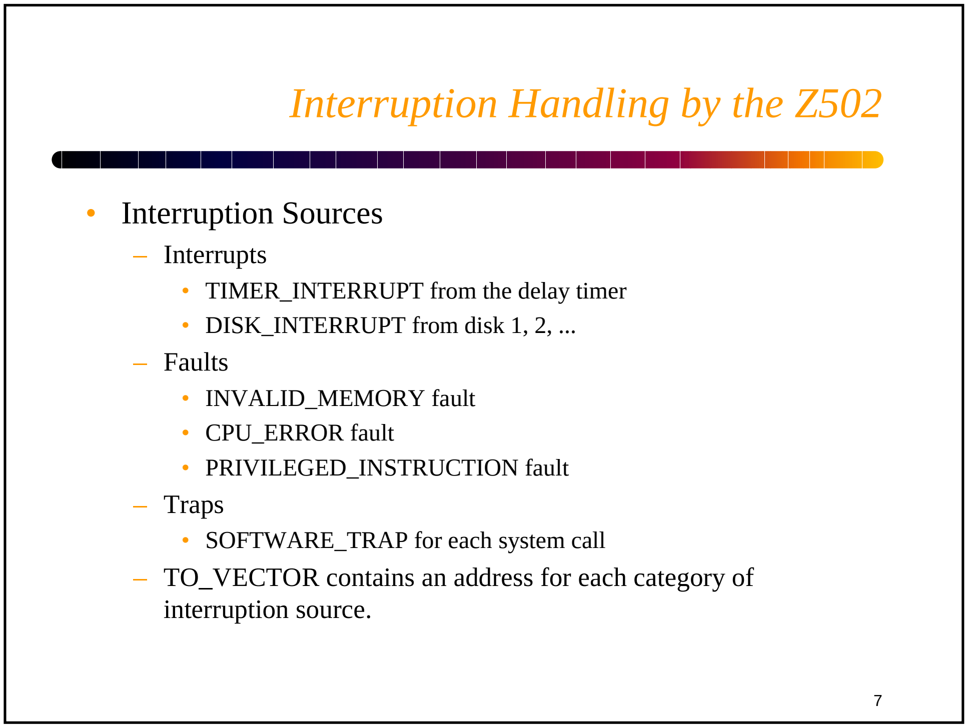
```
void test0( void )
{
   SELECT_STEP
   {
      STEP( 0 )
         printf(“This is test 0”);
         GET_TIME_OF_DAY( &Z502_REG_1 );
      STEP( 1 )
         printf(“Time of day is %d\n”, Z502_REG_1);
         TERMINATE_PROCESS( -1, &Z502_REG_9 );
      STEP( 2 )
         printf(“Error: Test should be terminated, but isn’t\n”);
         break;
   }
}
```

Z502 Architecture (cont.)

- User Mode (cont.)
  - Address space for user programs is divided into
    - C code “program” memory for instructions and for local variables. This, for all intents and purposes, is not constrained in size.
    - User “data” memory, referenced through a virtual address space, and called MEMORY, and accessed from user space through the MEM_XXXX macros. No programs in phase 1 access this user memory.
- Kernel Mode
  - Instruction set includes C language instructions, plus
    - access to all the Z502 registers
    - access to Z502 physical memory (MEMORY)
    - access to the privileged instructions of the Z502 instruction set
      - I/O primitives
      - memory primitives
      - context switching primitives
    - These are all available through provided macros

Z502 Registers and Vectors

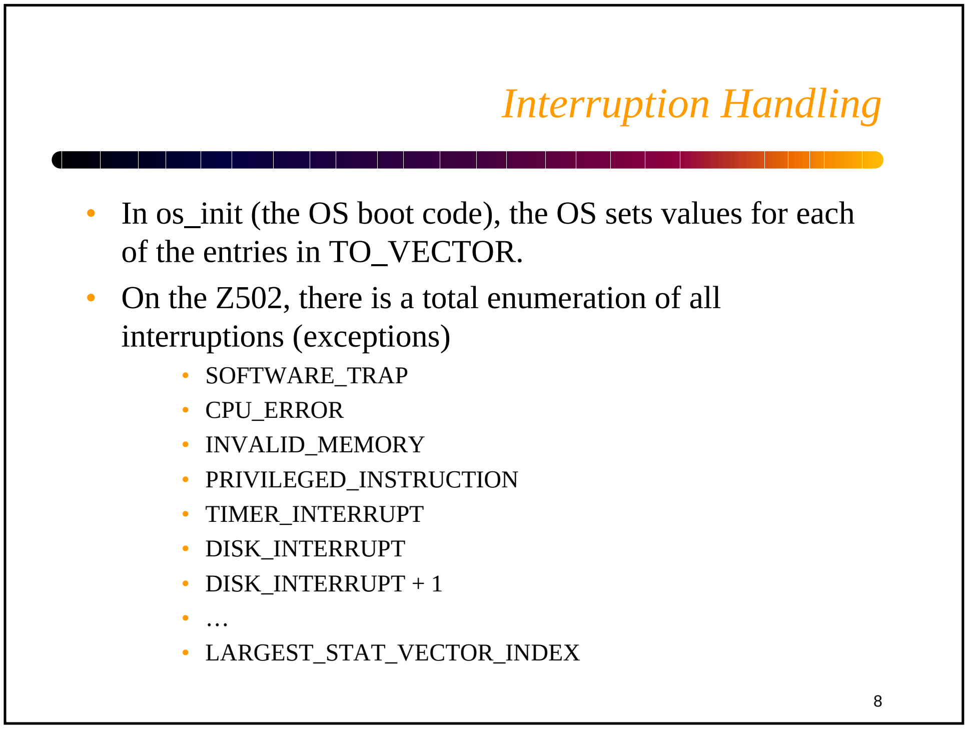
Name	Bits	Usage
Z502_REG_ARG1 … Z502_REG_ARG6	32	For passing system call parameter values
Z502_REG_1 … Z502_REG_9	32	General purpose
Z502_REG_PROGRAM_COUNTER	32	Points to next location in user program
Z502_REG_PAGE_TABLE_ADDR	32	Points to page table
Z502_REG_PAGE_TABLE_LENGTH	32	Length of page table in 32 bit entries
Z502_REG_CURRENT_CONTEXT	32	Handle for current context
Z502_REG_INTERRUPT_MASK	32	Interrupt enable/disable
TO_VECTOR	3 x 32	Addresses of interruption handlers
STAT_VECTOR	2 x N x 32	Exception statuses



Interruption Handling by the Z502

- Interruption Sources
  - Interrupts
    - TIMER_INTERRUPT from the delay timer
    - DISK_INTERRUPT from disk 1, 2, ...
  - Faults
    - INVALID_MEMORY fault
    - CPU_ERROR fault
    - PRIVILEGED_INSTRUCTION fault
  - Traps
    - SOFTWARE_TRAP for each system call
  - TO_VECTOR contains an address for each category of interruption source.

7



Interruption Handling

- In os_init (the OS boot code), the OS sets values for each of the entries in TO_VECTOR.
- On the Z502, there is a total enumeration of all interruptions (exceptions)
    - SOFTWARE_TRAP
    - CPU_ERROR
    - INVALID_MEMORY
    - PRIVILEGED_INSTRUCTION
    - TIMER_INTERRUPT
    - DISK_INTERRUPT
    - DISK_INTERRUPT + 1
    - …
    - LARGEST_STAT_VECTOR_INDEX

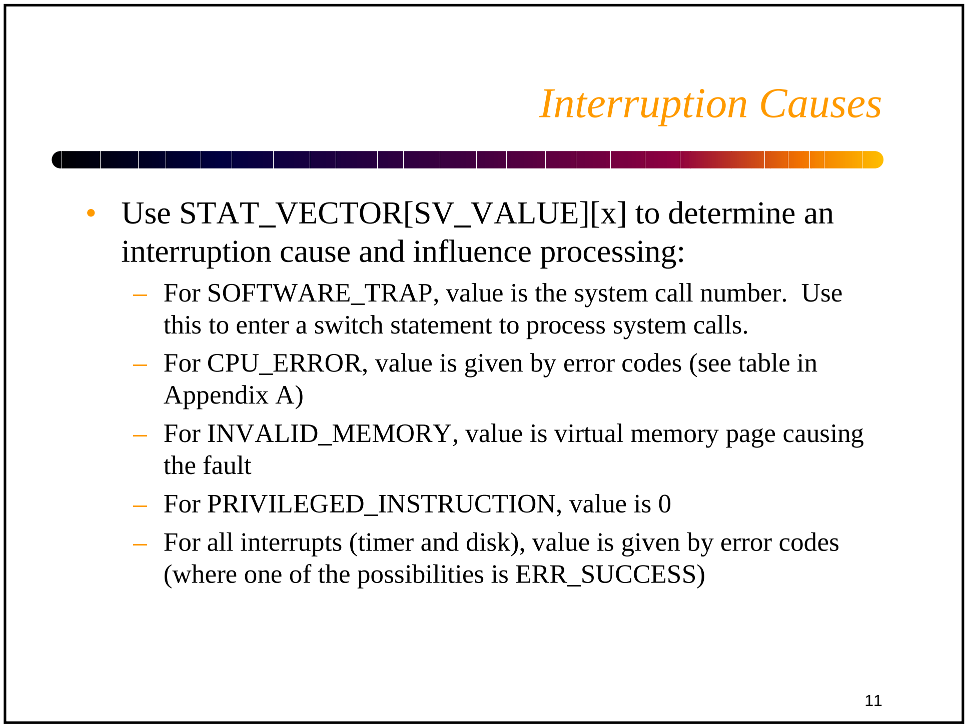
8

Z502 Hardware Actions on Interruption

- Let the *interruption number* (called *exception* in Appendix A) be *x*.
- User registers are saved in Z502 *Hardware Context*
- Hardware sets
  - STAT_VECTOR[SV_ACTIVE][*x*] = TRUE
  - STAT_VECTOR[SV_VALUE][*x*] = *interruption specific info*
- Execution mode is set to *kernel*
- Hardware begins execution at Interrupt, Fault, or Trap entry point as defined by TO_VECTOR
- Note that INTERRUPT_MASK is not set to TRUE. The operating system must do this if that is the desired mode of operation.

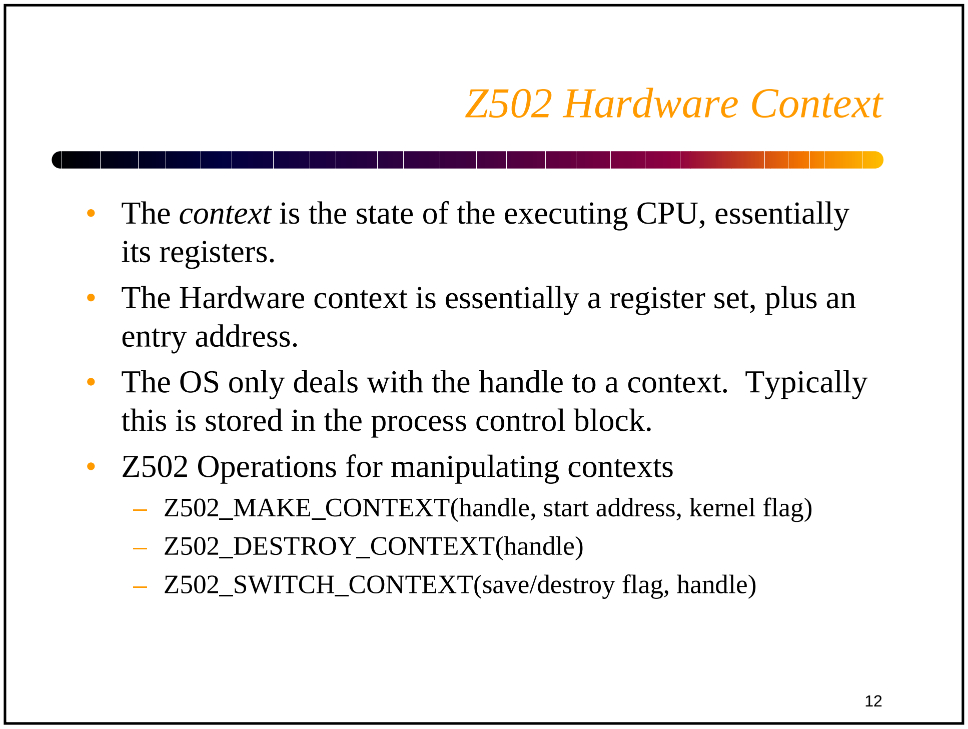
OS Responsibilities on an Interruption

- On Entry
  - Mask interrupts (if desired)
  - Clear the Interruption Source
    - set STAT_VECTOR[SV_ACTIVE][x] to FALSE
  - Determine the cause of the interruption and process accordingly
- On Exit
  - Unmask interrupts (if not already done).
  - For Interrupts, simply *return*
  - For traps and faults, ultimately exit the OS by performing a context switch (even if that switches back to the original process). This operation restores the user registers from the Z502 *Hardware Context* and sets the execution mode back to *user*.



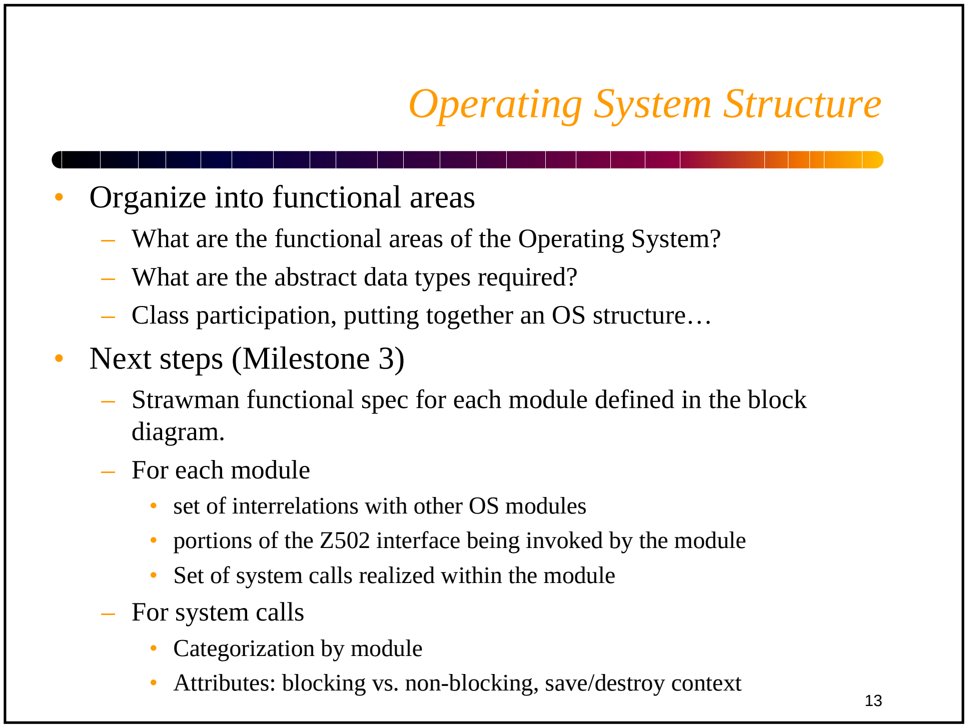
Interruption Causes

- Use STAT_VECTOR[SV_VALUE][x] to determine an interruption cause and influence processing:
  - For SOFTWARE_TRAP, value is the system call number. Use this to enter a switch statement to process system calls.
  - For CPU_ERROR, value is given by error codes (see table in Appendix A)
  - For INVALID_MEMORY, value is virtual memory page causing the fault
  - For PRIVILEGED_INSTRUCTION, value is 0
  - For all interrupts (timer and disk), value is given by error codes (where one of the possibilities is ERR_SUCCESS)



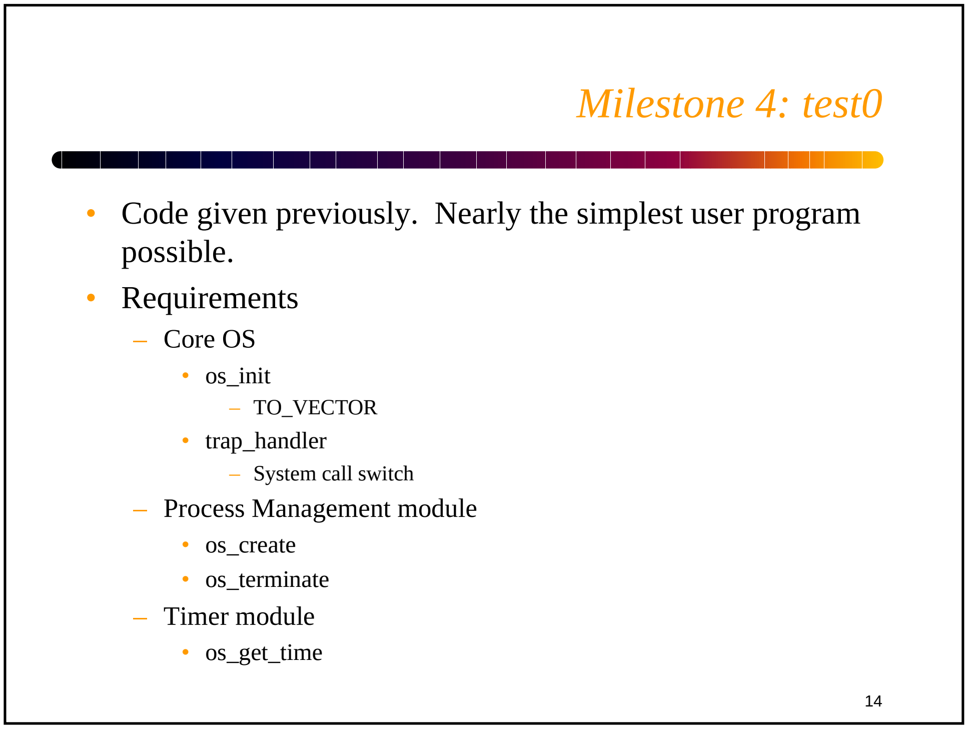
Z502 Hardware Context

- The *context* is the state of the executing CPU, essentially its registers.
- The Hardware context is essentially a register set, plus an entry address.
- The OS only deals with the handle to a context. Typically this is stored in the process control block.
- Z502 Operations for manipulating contexts
  - Z502_MAKE_CONTEXT(handle, start address, kernel flag)
  - Z502_DESTROY_CONTEXT(handle)
  - Z502_SWITCH_CONTEXT(save/destroy flag, handle)



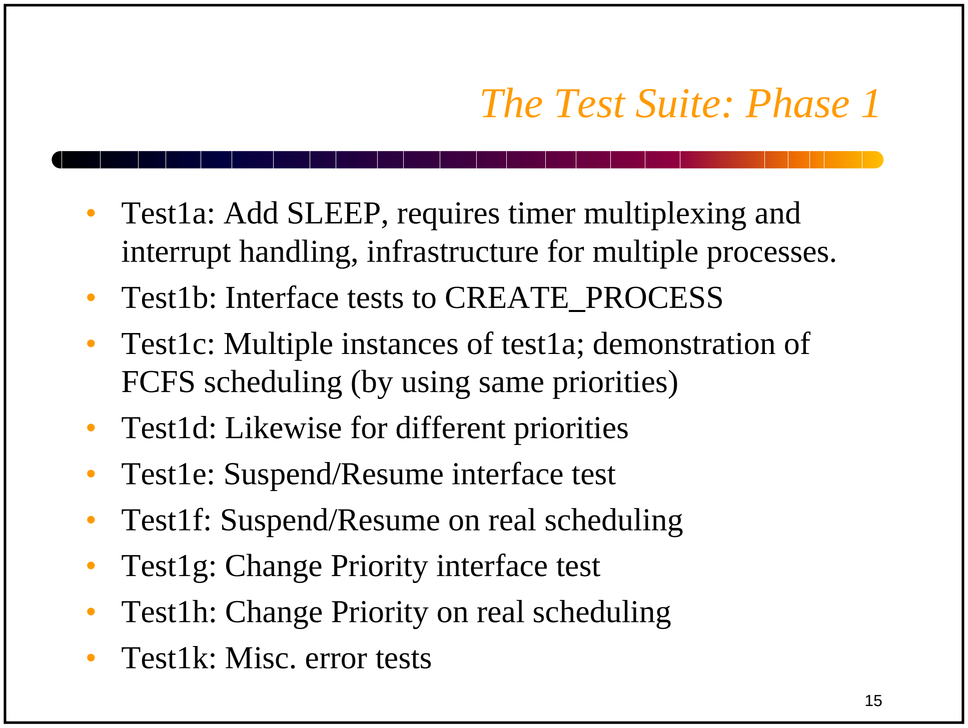
Operating System Structure

- Organize into functional areas
  - What are the functional areas of the Operating System?
  - What are the abstract data types required?
  - Class participation, putting together an OS structure…
- Next steps (Milestone 3)
  - Strawman functional spec for each module defined in the block diagram.
  - For each module
    - set of interrelations with other OS modules
    - portions of the Z502 interface being invoked by the module
    - Set of system calls realized within the module
  - For system calls
    - Categorization by module
    - Attributes: blocking vs. non-blocking, save/destroy context



Milestone 4: test0

- Code given previously. Nearly the simplest user program possible.
- Requirements
  - Core OS
    - os_init
      - TO_VECTOR
    - trap_handler
      - System call switch
  - Process Management module
    - os_create
    - os_terminate
  - Timer module
    - os_get_time



The Test Suite: Phase 1

- Test1a: Add SLEEP, requires timer multiplexing and interrupt handling, infrastructure for multiple processes.
- Test1b: Interface tests to CREATE_PROCESS
- Test1c: Multiple instances of test1a; demonstration of FCFS scheduling (by using same priorities)
- Test1d: Likewise for different priorities
- Test1e: Suspend/Resume interface test
- Test1f: Suspend/Resume on real scheduling
- Test1g: Change Priority interface test
- Test1h: Change Priority on real scheduling
- Test1k: Misc. error tests