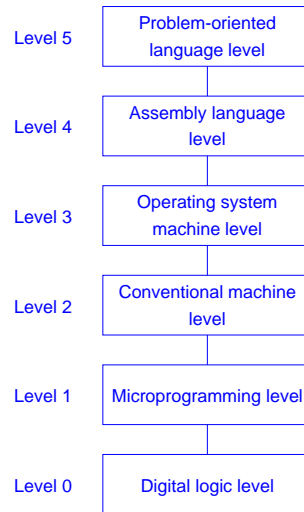


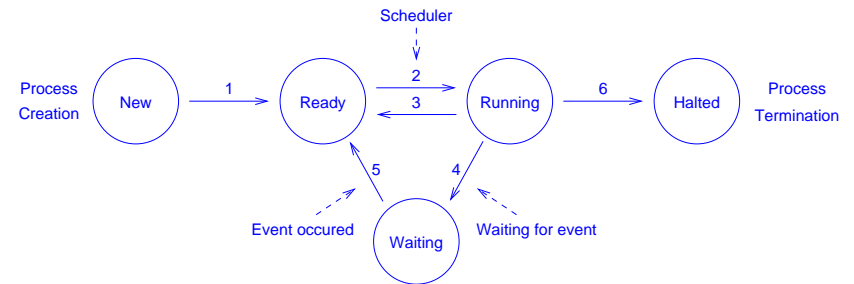


## Review of Abstract Machine Levels



## Fair CPU Usage and Scheduling

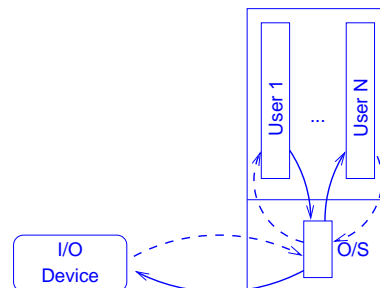
- how can the CPU be shared fairly between various processes?
- is the scheme of allowing other users in when waiting on I/O fair?
  - depends on the frequency and nature of such I/O requests
- better: enforce limits on processes using the CPU by using time slicing
  - typically 1–10 ms
  - O/S (with hardware support) generate a special timer interrupt when the time is up to change process (other processes wait in a queue)



- note: processes can be assigned priorities

## Processes

- we can think of a process as being the execution of a single program
  - program code, program counter, contents of processor registers, data allocated on stack, global data
- why have multiple processes?
  - a single process may not utilise the CPU well
  - much time may be wasted with I/O (see diagram)
  - can let other processes use CPU while waiting for external activities, like I/O
  - enables a computer to be *multi-functional* (even a modern PC)!



## Process State

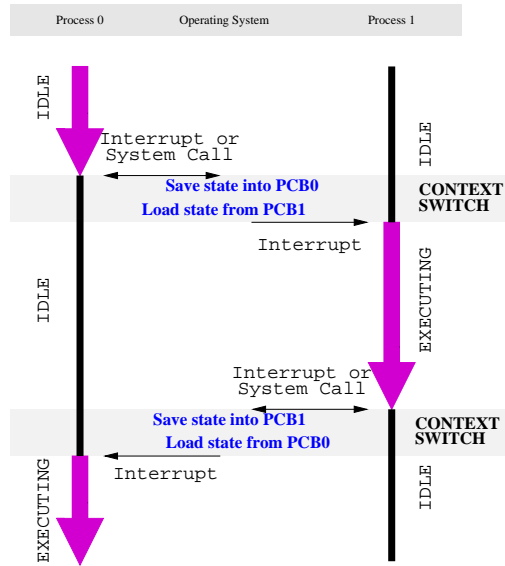
- O/S must be able to interrupt a process and later restore it to exactly the same state
- the O/S must save the process's state information
  - kept in the process control block
- the memory associated with one process must not be modified by another
- upon stopping a process, the O/S must save the values of all registers
- prior to restarting a process, the O/S must
  - recover the values of all registers
  - ensure that its code/data is in (or can be paged into) main memory

(what's the difference between an interrupt and trap?)

process control block:

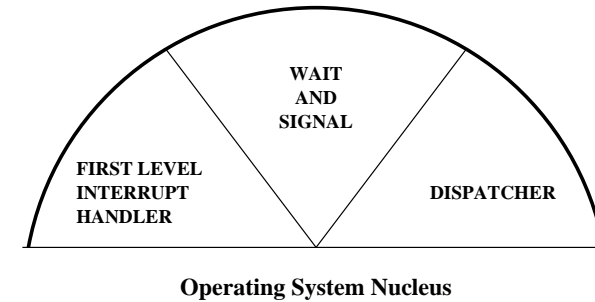
Process State
Process Number
Program Counter
Registers
List of Open Files
Memory Limits
possible other info

## Context Switching: Process State Manipulation

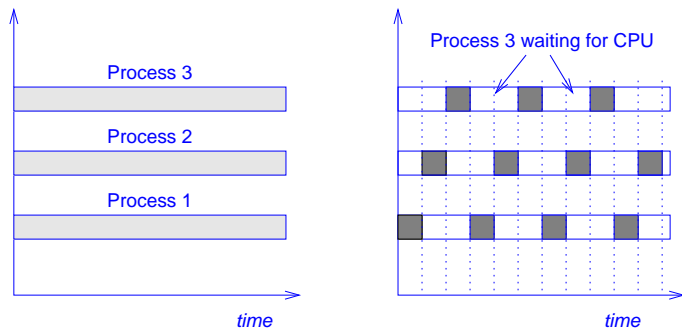


## Core Operating System Functions: Process Management

- within the level 3 (O/S level) abstract machine, OS functions may be arranged at several levels
- process management is largely handled at the innermost level



## Multiprocessing and Multiusers



- effect of multiprocessing is that a single processor can create the (virtual) effect of multiple processors
- several users may simultaneously use the same machine and consider themselves to be the sole user
- memory protection: must ensure that user1 cannot 'clobber' (processes of) user2 or the operating system (example)

## How the PeANUt was fixed: lessons for SE and Computer Systems

- code base of  $\approx 22K$  lines of C and Tcl/Tk/Tix code
  - don't understand Tcl/Tk/Tix (or want to!); don't have much time!
- step 0: (in despair) download latest Tix libraries, install & run demos (2 hrs)
- step 1: get an overall understanding of main Peanut execution path (2 hrs)
- step 2: debug!
  - (04/07) assemble max.ass bug: Error: Start address identifier not a label (58)
  - (01/08) bugs arising from evolution of Tcl/Tk/Tix: (2 hrs)
    - ◆ search demos / source code for offending procedure calls!
  - (01/08) Tcl exit procedure fails! what to do?? (0.5 hrs)
  - (05/08) segmentation violation when starting VM mode (3 hrs)
    - ◆ first trace execution to find where it crashes (how do you trace in Tcl?)
    - ◆ another 'sleeping' coding error (several others spotted and fixed too)
- Q: how can a program stop working when re-compiled? when *not* re-compiled?
- lesson: be very careful with variables on the stack (any harm in over-allocating?)
- don't have to understand *all* of a system to fix it! (right approach; tenacity)