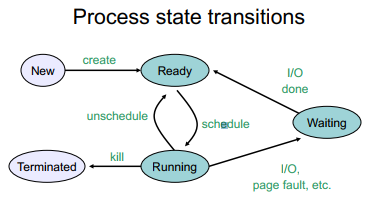
Var: Stack ; Global: Data/BSS ; Function : code/text

|  |  |
| --- | --- |
| User-level threads: Many-to-one thread mapping  – Implemented by user-level runtime libraries  • Create, schedule, synchronize threads at user-level  – OS is not aware of user-level threads  • OS thinks each process contains only a single thread of control  Advantages  – Does not require OS support; Portable  – Can tune scheduling policy to meet application demands  – Lower overhead thread operations since no system call  Disadvantages?  – Cannot leverage multiprocessors  – Entire process blocks when one thread blocks | Kernel-level threads: One-to-one thread mapping  – OS provides each user-level thread with a kernel thread  – Each kernel thread scheduled independently  – Thread operations (creation, scheduling, synchronization)  performed by OS  Advantages  – Each kernel-level thread can run in parallel on a multiprocessor  – When one thread blocks, other threads from process can be  scheduled  Disadvantages  – Higher overhead for thread operations  – OS must scale well with increasing number of threads |

A program consists of: – Code: machine instructions – Data: variables stored and manipulated in memory • initialized variables (globals) • dynamically allocated variables (malloc, new) • stack variables (C automatic variables, function arguments) • What is added by a process? – DLLs: libraries that were not compiled or linked with the program • containing code & data, possibly shared with other programs – mapped files: memory segments containing variables (mmap()) • used frequently in database programs – OS resources: open files • Whats the relationship between a prog

 a. Process A: trap into OS via int $80

b. Process A: calls read()

c. Process B: continues execution

d. OS: return from trap (into B)

e. OS: handle trap

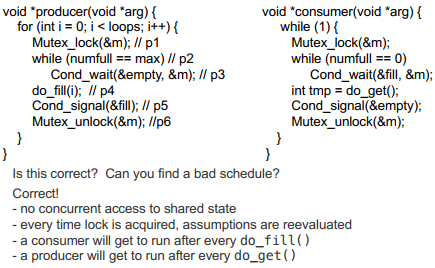
f. OS: call context switch routine

g. OS: send read to disk

h. OS: restore registers(B) from PCB(B)

i. OS: save registers(A) to PCB(A)

j. OS: put A to sleep (i.e. A’s state = blocked)

k. Hardware: jump to B’s PC

l. Hardware: timer interrupt

m. Hardware: move to user mode

n. Hardware: jump to trap handler

o. Hardware: move to kernel mode

Process A switches to B, because it calls read() and blocks.

Your Answer:

b---a---o---n---e---g---j---f---i---h---d---m---k---c

Process A switches to B, because a timer interrupt happens.

Your Answer:

l---o---n---e---f---i---h---d---m---k---c