Deadlock Wrapup & Memony Management 4/22/24 request / chopstick / at a time - max 2 requests 3 Chopsticks, 3 philosophers Deadlock Awidance A B C sate 0⁽²⁾ 0⁽²⁾ 0⁽²⁾ Unsafe avail= 3 awail=2 starts out $\begin{array}{c} can finish \\ A & B \\ safe \\ 1 \\ \end{array} \begin{pmatrix} (1) \\ 0 \\ 0 \\ \end{array} \begin{pmatrix} (2) \\ 0 \\ 0 \\ 0 \\ \end{array} \begin{pmatrix} (2) \\ 0 \\ 0 \\ 0 \\ \end{array} \end{pmatrix}$ to be avail = 2 2-1+2 = 3 Safe (A request 1 more) (A nill finish a tetum both) $\begin{array}{c|c} A & B & C \\ Safe & 1^{(1)} & 0^{(2)} & C^{(1)} \end{array}$ avail =1 to be avail = 1 1-1+2=2 2 A can finish 2-0+2=22 B can finish 2-1+2=32 c can finish $unsole A \times B C$ avail = 0 to be avoil = 0 no one Can Finish their future request

Deadlock Detection

-> rare event, let it happen, detect & recover when it happens

-> Resource Allecation Graph

A high false positive Pi Pz if there's a cycle : · Single instance resource => deadback · multi instance resource => potential deadlock held R2 P3 (it R2 has multiple instance) Ostrich Algorithm -> pretend nothing's wring, let user deal w/z

-> recover from dead lock -> abort / terminate a process in the cycle



Physical Menony Management -> volastile, byte addressable, order of GBs -> ~ ZOO cydes alless lateny Problem: namy processes, limited physical memory -> Attemp #1 = let one process use all of physical memory -> no need for translation -> constact suitch A scheduling -> write A's VAS to disk overhead => load B's VAS into Memory Physical Mems 2 -> how do we enforce kernel/user meniony separation

-> Attempt #2: Divide up physical memory among processes o base A -> low context suitch overhead -> how do we support -> virtual to physical address translation 100 bound pr. memory growth? -> base & bound registers (hu support) -> variable sized PA = VA + base (VA < bound) allocation leads B to external bounds base register [_____] for process A bound register [____] 200 Vare fragmentation D dolon't 300 bounde / fit, can't physical memory Processor's View Physical Implementation allocate C Memory Virtual Base Memory Virtual Virtual Physical Address Address Address Processor Processor Base+ Bound Bound

Compaction

update base & bound A A registers for each fragmantation UNUSED moved (compacted) B \Rightarrow B miless C t fragmentation expensive operation, C UNUSED causes variability physical memory physical memory in performance.

-> Can we share library code or support can fork with this approach? -> translation done on the entire VAS, can't share part of it

-> Attempt # 3 = finer grained translation & permission

sharing page A (library Lode) divide $\begin{array}{c|c} A & A & A \\ \hline B & E \\ \hline C & B \\ \hline B \\ \hline B \\ \hline B \\ \hline \end{array}$ the VAS -> better control for sharing -> fixed sized allocation (page (4)(B) into fixed size pages. D E no external fragmentation not yet E (may see internal "instead) accessed, not paged in D process 1's VAS process 2's VAS -> easy to support memory growth physical -> only load pages in use into memory menony (demand paging) What should be translated 64 bits offset within a page virtual address 1 52 12 page # offsets = offset within a physical page. only need to translate page #