Processes can execute concurrently ????Returns the original value of passed parameter "value" 3. That is, the swap takes place only under this condition. 5.22 Silberschatz, Galvin and Gagne (C)2013 Operating System Concepts – 9th Edition Solution using compare_and_swap Shared integer "lock" initialized to 0; Solution: do { while (compare_and_swap(&lock, 0, 1) != 0) ; /* do nothing */ /* critical section */ lock = 0; /* remainder section */ } while (true); 5.23 Silberschatz, Galvin and Gagne (C)2013 Operating System Concepts – 9th Edition int compare _and_swap(int *value, int expected, int new_value) { int temp = *value; if (*value == expected) *value = new_value; return temp; } ... lock = 0; do { while (compare_and_swap(&lock, 0, 1) != 0) ; /* do nothing */ /* critical section */ lock = 0; /* remainder section */ } while (true); Solution using compare_and_swap lock value 0 expected 0 new_value 1 temp P0 0 1 Animated by Sarah Al-Shareef (C) 2018 5.24 Silberschatz, Galvin and Gagne (C)2013 Operating System Concepts – 9th Edition int compare _and_swap(int *value, int expected, int new_value) { int temp = *value; if (*value == expected) *value = new_value; return temp; } ... lock = 0; do { while (compare_and_swap(&lock, 0, 1) != 0) ; /* do nothing */ /* critical section */ lock = 0; /* remainder section */ } while (true); Solution using compare_and_swap lock value 0 expected 0 new_value 1 temp P0 0 1 P1 Animated by Sarah Al-Shareef (C) 2018 1 5.25 Silberschatz, Galvin and Gagne (C)2013 Operating System Concepts – 9th Edition Mutex Locks Previous solutions are complicated and generally inaccessible to application programmers OS designers build software tools to solve critical section problem Simplest is mutex lock (mutual exclusion lock) Protect a critical section by first acquire() a lock then release() the lock Boolean variable indicating if lock is available or not Calls to acquire() and release() must be atomic Usually implemented via hardware atomic instructions One disadvantage of this solution: it requires busy waiting While the process is in CS, any other process tries to enter must loop continuously in the call to acquire() Busy waiting wastes CPU cycles.Solved via priority-inheritance protocol5.18 Silberschatz, Galvin and Gagne (C)2013 Operating System Concepts – 9th Edition Solution using test_and_set() Shared Boolean variable lock, initialized to FALSE Solution: do { while (test_and_set(&lock)) ; /* do nothing */ /* critical section */ lock = false; /* remainder section */ } while (true); 5.19 Silberschatz, Galvin and Gagne (C)2013 Operating System Concepts – 9th Edition Solution using test_and_set() boolean test_and_set (boolean *target) { boolean rv = *target; *target = TRUE; return rv: } ... lock = false; do { while (test_and_set(&lock)) /wait/ ; /* critical section */ lock = false; /* remainder section */ } while (true); lock rv target P0 Animated by Sarah Al-Shareef (C) 2018 5.20 Silberschatz, Galvin and Gagne (C)2013 Operating System Concepts – 9th Edition Solution using test_and_set() boolean test_and_set (boolean *target) { boolean rv = *target; *target = TRUE; return rv: } ... lock = false; do { while (test_and_set(&lock)) /wait/ ; /* critical section */ lock = false; /* remainder section */ } while (true); lock rv target P0 P1 Animated by Sarah Al-Shareef (C) 2018 5.21 Silberschatz, Galvin and Gagne (C)2013 Operating System Concepts – 9th Edition compare_and_swap Instruction Definition: int compare _and_swap(int *value, int expected, int new_value) { int temp = *value; if (*value == expected) *value = new_value; return temp; } 1.Originally called P() and V() Definition of the wait() operation wait(S) { while (S