5a. Implementing Better Locks

Files:

xv6_sync.c

xv6_sync.h

lock_benchmark.c

1. spinlock_t:
    - A simple spinlock implementation that uses an atomic integer to represent the lock state.
    - spinlock_init initializes the lock to an unlocked state.
    - spinlock_acquire spins in a loop while trying to acquire the lock until it succeeds.
    - spinlock_release releases the lock by setting the lock state to 0.
2. ticket_lock_t:
    - A ticket lock implementation that uses two atomic integers to represent the ticket number and the currently serving ticket number.
    - ticket_lock_init initializes the lock to an initial state.
    - ticket_lock_acquire increments the ticket number and spins in a loop while waiting for the currently serving ticket number to match the thread's ticket number.
    - ticket_lock_release increments the currently serving ticket number.
3. rw_lock_t:
    - A reader-writer lock implementation that uses a combination of a spinlock and a ticket lock for writers, and an atomic integer for readers.
    - rw_lock_init initializes the lock to an initial state.
    - rw_lock_acquire_reader increments the number of active readers.

- rw_lock_release_reader decrements the number of active readers.
- rw_lock_acquire_writer increments the number of waiting writers, acquires the ticket lock, and spins in a loop while waiting for all active readers to finish. Then it sets the writer active flag to 1.
- rw_lock_release_writer sets the writer active flag to 0 and releases the ticket lock.
- rw_lock_try_acquire_writer attempts to acquire the writer lock without blocking and returns a success or failure status.

4. queue_lock_t:
    - A lock-free queue implementation that uses an atomic integer for the lock state and atomic pointers for the queue head and tail.
    - queue_lock_init initializes the queue and the lock to an initial state.
    - queue_lock_acquire adds a new node to the queue and spins in a loop while waiting for the lock to become available. When the lock is available, it sets the lock state to 1 and updates the queue head.
    - queue_lock_release removes the first node from the queue and updates the queue head and tail. If the queue is now empty, it sets the lock state to 0.
5. main functions for performance testing:
    - These are main functions that create multiple threads and perform a large number of lock acquisitions and releases, measuring the time taken by the lock operations.
    - There are three main functions, each for testing a different type of lock: ticketlock, non-starving reader-writer lock, and queue lock.

This lock_benchmark.c creates a specified number of threads and measures the time taken for each thread to acquire and release the lock a specified number of iterations.

To add the park(), unpark(), and setpark() system calls to xv6, you need to follow these steps:

1. Add the function prototypes to the system call table in syscall.h.
2. Add the function definitions in sysproc.c.
3. Implement the actual functionality in proc.c.
4. Update the user-space library in usys.S and user.h.