

CS 236: Operating Systems Lab Spring 2025-26 Labquiz4 (20 marks)	A
--	-----------------

Instructions

- No Internet. No phone. No devices. No handwritten notes. No books.
- No friends (strictly for the duration of the exam only!).
- Unethical practices will be an automatic recommendation for the FR grade.
- **Hardcoding of outputs will result in -5 marks per instance.**
- Please read the questions and submission instructions carefully.
- The labquiz consists of 3 problems.
- **All problems are compulsory.**

Submission Guidelines

- **Untar and cd into the labquiz2 directory on your Desktop. Use:**
 - **`./run.sh` to run xv6 in containerized environment**
 - **`submit-xv6` to submit (ask a TA to enter password)**
- **Ensure that your final xv6 submission compiles without any errors.**
Submissions that do not compile will not be considered for evaluation.

1. locks meet priority

Implement a simplified synchronization mechanism using a **futex**-like interface for user space threads (and possibly processes with shared memory setups). The goal is to use the kernel to support user space threads and synchronization.

Implement the following **user-space** function:

int futex(int *uaddr, int ops, int val)

- The **uaddr** argument points to the futex variable, i.e., the shared memory location used for synchronization.
- **ops** specifies the type of operation
 - 0 (FUTEX_LOCK)
 - 1 (FUTEX_UNLOCK)
- **val** is an operation specific value and is used for the specified operation
- The **return** value is **-1** in case of invalid arguments, and **0** otherwise.

Supported operations:

- FUTEX_LOCK
 - **val == 0**: Acquire lock using spinning (spinlock behavior)

- **val == 1**: Acquire the lock, but sleep using **futex_wait** if it is not immediately available and retry on wakeup till success
- **FUTEX_UNLOCK**
 - Release the lock
 - If **val == 0**: Simply release the spinlock.
 - If **val == 1**: Wake up threads waiting on specified lock using **futex_wake**
 - If **val == 2**: Wake up any **one** thread waiting on specified lock using **futex_wake**
 - If **val == 3**: Wake up any **one** thread waiting on specified lock based on priority (higher priority threads wake up before lower priority threads)

Implement the following system calls that will be used in the **futex** function.

Note that the system call helper functions need to handle mutual exclusion cases, so that there is no deadlock with sleep and wakeup interspersing across threads.

1. **futex_wait(int* uaddr)**
puts the thread to sleep if the value at **uaddr** matches **1** (checks it atomically)
2. **futex_wake(int* uaddr, int arg)**
wakes up the threads waiting on the **uaddr** specified lock.
The number and which process(es) to wake up depends on the **arg** argument.
 - If **arg == 0**: Wake up all threads sleeping on the specified lock.
 - If **arg == 1**: Wake up any one thread sleeping on the specified lock.
 - If **arg == 2**: Wake up any one thread waiting on specified lock based on priority (higher priority threads wakeup before lower priority threads).

Hint:

1. Implementation of the **clone**, **join** and **set_priority** system calls for xv6 threads is provided as part of the lab.
2. Refer to `spinlock.c`, `spinlock.h`, `sleeplock.c`, `sleeplock.h`
3. **futex** is user space function you may want to implement it in **ulib.c**

futex does several synchronization related operations in user space — lock checking, issuing system calls for sleep and wakeup, re-checking of locks, spinning for lock etc.

TASKS

1. Implementation of the **futex** user space synchronization function
2. Implementation of the **futex_wait** and **futex_wake** system calls
3. Implement a user-space program **ufutex.c** that has the following command-line usage

Shell

Usage: `ufutex <nthreads> <thread_order[0]> ... <thread_order[nthreads-1]>`

nthreads: the number of threads

Followed by **nthreads** integers (**0** to **nthreads-1**) that specify the thread order for wakeup. Assume thread index starts at **0**.

E.g.,

```
$ ufutex 5 0 1 2 3 4
```

5 threads to be woken up in the order of threads ids 0, 1, 2, 3, 4.

Thread 1 wakes up to do its task after thread 0 is done (releases the lock), and so on ...

Notes:

- Use the provided print statements from the template to show the execution order.
- The solution needs to use **exactly one userspace lock**.
- Each worker thread acquires the lock **exactly** once.
- The **set_priority(int pid, int priority)** system call is provided as part of the template code.
- Solutions that use additional synchronization primitives such as:
 - semaphores
 - extra locks
 - barriers**are not allowed**.
- The ordering of threads must be implemented only using the priority-based futex mechanism. Do not use any other method to enforce thread ordering.

Test cases

- p1a.c — Test correctness of futex spinlock (race condition)
- p1b.c — Test correctness of futex sleeplock (race condition + blocking behavior)
- p1c.c — Test correctness of futex_wake (arg == 0 and arg == 1)
- p1d.c — Test correctness of futex_wake (arg == 3)
- Variations to test implementation of ufutex

Sample outputs — Appendix A

2. only if you write,you can read or is it the other way round?

Implement the reader writer synchronization primitive using the pthread API.

Each thread is either a reader or a writer and needs to synchronize appropriately.

Readers: Only read shared data. Multiple readers can safely read concurrently.

Writers: Modify shared data. A writer must have exclusive access.

To implement at least 4 functions — ReaderLock, ReaderUnLock, WriteLock and WriterUnlock

Constraints

Design a synchronization mechanism using the pthread API to meet the following requests —

1. Multiple readers can read the shared resource concurrently.
2. Writers must have **exclusive access** (no readers or other writers while writing).
If a writer is active, no new readers or other writers should enter the critical section.
3. If readers are active, writers must wait.
4. **If writers and readers are both waiting, readers get preference!**
5. No busy waiting. All blocking must use sleep-wakeup (wait/signal) based synchronization.

TASKS

1. Implement the functions **ReaderLock**, **ReaderUnLock**, **WriterLock** and **WriterUnlock** defined as empty functions in **rwlock.c**
2. You may want to update the **read_write_lock** structure definition in **rwlock.h**
3. Look for helper files/templates in the labquiz tar file.

Test cases

- p2a.c — only writers based synchronization
- p2b.c — only readers based synchronization
- p2c.c — many readers and writers queue up, and readers must be given priority
- pd2.c — multiple busy writers with waves of readers

Sample outputs — Appendix B

Appendix A

Sample outputs for Q1 – locks meet priority

(p1a.c): Test correctness of futex spinlock (race condition):

```
Shell
$ p1a
Usage: p1a <num_threads> <iterations>
$ p1a 5 100000
[main] VAR value: 0, nthreads: 5
[T-1] 25000/100000 iterations done
[T-0] 25000/100000 iterations done
[T-2] 25000/100000 iterations done
[T-1] 50000/100000 iterations done
[T-3] 25000/100000 iterations done
[T-1] 75000/100000 iterations done
[T-0] 50000/100000 iterations done
[T-2] 50000/100000 iterations done
[T-3] 50000/100000 iterations done
[T-1] Finished
[T-0] 75000/100000 iterations done
[T-4] 25000/100000 iterations done
[T-2] 75000/100000 iterations done
[T-3] 75000/100000 iterations done
[T-0] Finished
[T-4] 50000/100000 iterations done
[T-2] Finished
[T-3] Finished
[T-4] 75000/100000 iterations done
[T-4] Finished
[main] All threads joined, VAR value: 500000, expected: 500000
```

(p1b.c): Test correctness of futex sleeplock (race condition + blocking behavior)

```
Shell
$ p1b
Usage: p1b <num_threads> <iterations>
$ p1b 2 10000
[main] === Test A: sleeplock correctness ===
[main] initial VAR = 0, nthreads = 2
[T-0] 2500/10000 iterations done
[T-1] 2500/10000 iterations done
```

```
[T-1] 5000/10000 iterations done
[T-0] 5000/10000 iterations done
[T-1] 7500/10000 iterations done
[T-0] 7500/10000 iterations done
[T-1] Finished
[T-0] Finished
[main] all counter threads joined
[main] VAR = 20000, expected = 20000

[main] === Test B: sleeplock scheduling / blocking behavior ===
[main] Launching 2 threads
[sched] Scheduling pid 7
[sched] Scheduling pid 8
[pid 7] T-0 started
[sched] Scheduling pid 8
[pid 8] T-1 started
[sched] Scheduling pid 7
[sched] Scheduling pid 8
[sched] Scheduling pid 7
[sched] Scheduling pid 8
===== [pid 8] T-1 ACQUIRED LOCK ===== +
[sched] Scheduling pid 8
[sched] Scheduling pid 8
[sched] Scheduling pid 8
[sched] Scheduling pid 8
[sched] Scheduling pid 8
[sched] Scheduling pid 8
[sched] Scheduling pid 8
[sched] Scheduling pid 8
===== [pid 8] T-1 RELEASING LOCK ===== -
[sched] Scheduling pid 7
===== [pid 7] T-0 ACQUIRED LOCK ===== +
[sched] Scheduling pid 7
[sched] Scheduling pid 8
[sched] Scheduling pid 7
[sched] Scheduling pid 8
[sched] Scheduling pid 7
[sched] Scheduling pid 7
[sched] Scheduling pid 7
[sched] Scheduling pid 7
[sched] Scheduling pid 7
[sched] Scheduling pid 7
===== [pid 7] T-0 RELEASING LOCK ===== -
```

```
[sched] Scheduling pid 8
===== [pid 8] T-1 ACQUIRED LOCK ===== +
[sched] Scheduling pid 7
[sched] Scheduling pid 8
[sched] Scheduling pid 7
[sched] Scheduling pid 8
[sched] Scheduling pid 8
[sched] Scheduling pid 8
[sched] Scheduling pid 8
[sched] Scheduling pid 8
[sched] Scheduling pid 8
[sched] Scheduling pid 8
===== [pid 8] T-1 RELEASING LOCK ===== -
[sched] Scheduling pid 7
===== [pid 7] T-0 ACQUIRED LOCK ===== +
[sched] Scheduling pid 7
[sched] Scheduling pid 8
[sched] Scheduling pid 7
[sched] Scheduling pid 8
[sched] Scheduling pid 7
[sched] Scheduling pid 7
[sched] Scheduling pid 7
[sched] Scheduling pid 7
[sched] Scheduling pid 7
[sched] Scheduling pid 7
===== [pid 7] T-0 RELEASING LOCK ===== -
[sched] Scheduling pid 8
[pid 8] T-1 exiting
[sched] Scheduling pid 4
[sched] Scheduling pid 7
[sched] Scheduling pid 7
[pid 7] T-0 exiting
[sched] Scheduling pid 4

[main] all threads joined
```

(p1c.c): Test correctness of `futex_wake` (`arg == 0` and `arg == 1`):

```
Shell
$ p1c
Usage: p1c <num_threads>
$ p1c 3
```

```
=== Test A: wake all waiters ===
[pid 4 main] launching 3 threads
[sched] Scheduling pid 5
[sched] Scheduling pid 6
[sched] Scheduling pid 7
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[pid 4[sched] Scheduling pid 4
  main] waking all threads
[[sched] Scheduling pid 5
[sched] Scheduling pid 6
pi[sched] Scheduling pid 7
d 4 main] all threads woken

=== Test B: wake one waiter at a time ===
[pid 4 main] launching 3 threads again
[sched] Scheduling pid 8
[sched] Scheduling pid 9
[sched] Scheduling pid 10
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[pid 4 main] waking one thread
[sched] Scheduling pid 8
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[pid 4 main] waking one thread
[sched] Scheduling pid 9
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[pid 4 main] waking one thread
[sched] Scheduling pid 10
```



```
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4
[sched] Scheduling pid 4

[pid 4 main] all threads joined
```

(p1d.c): Test correctness of `futex_wake (arg == 3)`

```
Shell
$ p1d
Usage: p1d <num_threads>
$ p1d 5
[pid 4 main] launching 5 threads
[pid 4 main] launching thread-0 with priority 0
[pid 4 main] launching thread-1 with priority 1
[pid 4 main] launching thread-2 with priority 0
[pid 4 main] launching thread-3 with priority 1
[pid 4 main] launching thread-4 with priority 0
[thread-1] woke up and acquired lock!
[thread-3] woke up and acquired lock!
[thread-0] woke up and acquired lock!
[thread-2] woke up and acquired lock!
[thread-4] woke up and acquired lock!
[pid 4 main] all threads joined
$ p1d 8
[pid 10 main] launching 8 threads
[pid 10 main] launching thread-0 with priority 0
[pid 10 main] launching thread-1 with priority 1
[pid 10 main] launching thread-2 with priority 0
[pid 10 main] launching thread-3 with priority 1
[pid 10 main] launching thread-4 with priority 0
[pid 10 main] launching thread-5 with priority 1
[pid 10 main] launching thread-6 with priority 0
[pid 10 main] launching thread-7 with priority 1
[thread-1] woke up and acquired lock!
[thread-3] woke up and acquired lock!
[thread-5] woke up and acquired lock!
[thread-7] woke up and acquired lock!
[thread-0] woke up and acquired lock!
[thread-2] woke up and acquired lock!
```

```
[thread-4] woke up and acquired lock!  
[thread-6] woke up and acquired lock!  
[pid 10 main] all threads joined
```

(ufutex.c): Expected output for ufutex implementation.

```
Shell  
$ ufutex  
Usage: ufutex <nthreads> <thread_order[0]> ... <thread_order[nthreads-1]>  
$ ufutex 5 0 1 2 3 4  
[pid 5 main] launching 5 threads  
[pid 5 main] thread order: 0 1 2 3 4  
[pid 5 main] waking thread-0 first  
[pid 6 thread-0] acquired lock  
[pid 7 thread-1] acquired lock  
[pid 8 thread-2] acquired lock  
[pid 9 thread-3] acquired lock  
[pid 10 thread-4] acquired lock  
[pid 5 main] all threads joined  
$ ufutex 5 0 4 2 3 1  
[pid 11 main] launching 5 threads  
[pid 11 main] thread order: 0 4 2 3 1  
[pid 11 main] waking thread-0 first  
[pid 12 thread-0] acquired lock  
[pid 16 thread-4] acquired lock  
[pid 14 thread-2] acquired lock  
[pid 15 thread-3] acquired lock  
[pid 13 thread-1] acquired lock  
[pid 11 main] all threads joined
```

Appendix B

This will take around a minute or two.

Checks if only one writer can write at a time

```
Shell
> ./p2a

Final shared_data = 500000 (expected 500000)
```

```
Shell
> ./p2b

All readers finished. shared_data unchanged = 42
```

Readers should hold preference over writers

```
Shell
> ./p2c

=== Writer 0 holds lock, others queue up ===
Writer 0 acquired lock

=== Writer 0 releasing lock ===
Reader 4 acquired lock
Reader 1 acquired lock
Reader 5 acquired lock
Reader 9 acquired lock
Reader 10 acquired lock
Reader 3 acquired lock
Reader 7 acquired lock
Reader 2 acquired lock
Reader 8 acquired lock
Reader 6 acquired lock
Writer 1 acquired lock
Writer 2 acquired lock
Writer 3 acquired lock
Writer 4 acquired lock
Writer 5 acquired lock
```

```
Writer 6 acquired lock
Writer 7 acquired lock
Writer 8 acquired lock
Writer 9 acquired lock
Writer 10 acquired lock
Writer 11 acquired lock
Writer 12 acquired lock
Writer 13 acquired lock
Writer 14 acquired lock
Writer 15 acquired lock
Writer 16 acquired lock
Writer 17 acquired lock
Writer 18 acquired lock
Writer 19 acquired lock
Writer 20 acquired lock

=== Acquisition order ===
R4 R6 R1 R5 R2 R3 R9 R7 R8 R10 W1 W2 W3 W4 W5 W6 W7 W8 W9 W10 W11 W12 W13 W14
W15 W16 W17 W18 W19 W20

Readers before any writer: 10 / 10
PASS: Reader preference verified!
```

10 busy writers, and waves of readers

```
Shell
> ./p2d
=== Starting 10 writers looping ===

=== Reader wave 1 (spawning 3 readers among busy writers) ===
Reader 1 acquired lock (waited 4714 us)
Reader 3 acquired lock (waited 4743 us)
Reader 2 acquired lock (waited 4795 us)

=== Reader wave 2 (spawning 3 readers among busy writers) ===
Reader 101 acquired lock (waited 463 us)
Reader 103 acquired lock (waited 417 us)
Reader 102 acquired lock (waited 449 us)

=== Reader wave 3 (spawning 3 readers among busy writers) ===
Reader 201 acquired lock (waited 6326 us)
Reader 203 acquired lock (waited 6297 us)
Reader 202 acquired lock (waited 6408 us)
```

```
=== Reader wave 4 (spawning 3 readers among busy writers) ===
Reader 302 acquired lock (waited 4768 us)
Reader 303 acquired lock (waited 4814 us)
Reader 301 acquired lock (waited 4864 us)

=== Reader wave 5 (spawning 3 readers among busy writers) ===
Reader 403 acquired lock (waited 3278 us)
Reader 401 acquired lock (waited 3354 us)
Reader 402 acquired lock (waited 3324 us)

=== Results ===
Total lock acquisitions logged: 180
Total readers served: 15 / 15

=== Log around reader acquisitions ===
[... W9 W10 R1* R3 R2 ...]
[... W10 R1 R3* R2 W2 ...]
[... R1 R3 R2* W2 W1 ...]
[... W6 W8 R101* R103 R102 ...]
[... W8 R101 R103* R102 W7 ...]
[... R101 R103 R102* W7 W9 ...]
[... W5 W4 R201* R203 R202 ...]
[... W4 R201 R203* R202 W6 ...]
[... R201 R203 R202* W6 W8 ...]
[... W2 W1 R302* R303 R301 ...]
[... W1 R302 R303* R301 W3 ...]
[... R302 R303 R301* W3 W5 ...]
[... W7 W9 R403* R401 R402 ...]
[... W9 R403 R401* R402 W10 ...]
[... R403 R401 R402* W10 W2 ...]

PASS: All 15 readers were served despite 10 busy writers!
Final shared_data = 165
```

The time and order waited might not be the same, but the readers must appear grouped